LAPORAN ULANGAN AKHIR SEMESTER

DATA SCIENCE

Prediksi Rekomendasi Dress Menggunakan Model Klasifikasi Machine Learning dan Deep Learning pada Dataset Atribut Penjualan Dress

Oleh :

CHOIRIL ANWAR FAUZY
NPM. 234311036

Dosen Pengampu :
Gus Nanang Syaifuddiin, S.Kom., M.Kom

JURUSAN TEKNIK
PROGRAM STUDI TEKNOLOGI REKAYASA PERANGKAT LUNAK
POLITEKNIK NEGERI MADIUN
2025

INFORMASI PROYEK

Judul Proyek	: Prediksi Rekomendasi Dress Menggunakan Model Klasifikasi Machine Learning dan Deep Learning pada Dataset Atribut Penjualan Dress
Nama Mahasiswa	: Choiril Anwar Fauzy
NIM	: 234311036
Program Studi	: Teknologi Rekayasa Perangkat Lunak
Mata Kuliah	: Data Science
Dosen Pengampu	: Gus Nanang Syaifuddiin, S.Kom., M.Kom
Tahun Akademik	: 2025
Link GitHub Repository	:https://github.com/Choiril2306/Klasifikasi-Atribut-Penjualan-Dress
Link Video Pembahasan	: https://youtu.be/28zPRi2vqHQ

1. LEARNING OUTCOMES

Pada proyek ini, mahasiswa diharapkan dapat:

1. Memahami konteks masalah dan merumuskan problem statement secara jelas
2. Melakukan analisis dan eksplorasi data (EDA) secara komprehensif (OPSIONAL)
3. Melakukan data preparation yang sesuai dengan karakteristik dataset
4. Mengembangkan tiga model machine learning yang terdiri dari (WAJIB):
   - Model baseline
   - Model machine learning / advanced
   - Model deep learning (WAJIB)
5. Menggunakan metrik evaluasi yang relevan dengan jenis tugas ML
6. Melaporkan hasil eksperimen secara ilmiah dan sistematis
7. Mengunggah seluruh kode proyek ke GitHub (WAJIB)
8. Menerapkan prinsip software engineering dalam pengembangan proyek

2. PROJECT OVERVIEW

2.1 Latar Belakang

Industri e-commerce, khususnya pada penjualan produk fashion, menghasilkan banyak data dari berbagai atribut seperti harga, rating, style, season, dan rekomendasi. Tidak semua atribut tersebut berpengaruh terhadap tingkat penjualan, sehingga diperlukan analisis untuk mengetahui fitur mana yang paling penting. Dengan pemilihan fitur yang tepat, model prediksi dapat bekerja lebih akurat dan membantu memahami pola penjualan pada dataset Dresses Attribute Sales.
Permasalahan pada domain e-commerce terkait prediksi penjualan biasanya meliputi:

- Banyaknya atribut produk membuat model sulit menemukan pola yang relevan.
- Tidak semua fitur berpengaruh terhadap penjualan sehingga perlu seleksi fitur.
- Analisis manual sulit dilakukan karena jumlah data cukup besar.
- Model dapat kurang akurat jika fitur mengandung noise atau redundansi.

Dengan perkembangan machine learning dan deep learning, proses analisis dapat dilakukan lebih cepat dan akurat. Metode seperti SelectKBest membantu memilih fitur terbaik, sementara Random Forest memberikan informasi mengenai tingkat kepentingan fitur. Model Multilayer Perceptron (MLP) juga dapat mempelajari pola non-linear sehingga meningkatkan performa prediksi setelah fitur diseleksi. Proyek ini penting karena memberikan beberapa manfaat:

- Membantu memprediksi tingkat penjualan produk fashion.
- Mendukung pengambilan keputusan dalam pengelolaan stok dan pemasaran.
- Memberikan gambaran fitur mana yang paling berpengaruh.
- Meningkatkan akurasi model melalui pemilihan fitur yang tepat.

Penelitian sebelumnya menunjukkan bahwa pemilihan fitur dan penggunaan model seperti Random Forest dan Neural Network dapat meningkatkan performa prediksi pada data tabular.

Referensi :

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
R. Fildes, S. Ma, and S. Kolassa, "Predictive modeling in retail: Review and research directions," Eur. J. Oper. Res., vol. 257, no. 1, pp. 1–17, Feb. 2017. https://doi.org/10.1016/j.ejor.2016.06.059
J. Jiang, C. Zhang, L. Ke, N. Hayes, Y. Zhu, H. Qiu, B. Zhang, T. Zhou and G. Wei, "A review of machine learning methods for imbalanced data challenges in chemistry," Chem. Sci., vol. 16, pp. 7637, 2025. https://pubs.rsc.org/en/content/articlehtml/2025/sc/d5sc00270b

3. BUSINESS UNDERSTANDING / PROBLEM UNDERSTANDING

3.1 Problem Statements

1. Dataset Dresses Attribute Sales memiliki banyak atribut produk (style, price, rating, season, dsb.), sehingga perlu diketahui fitur mana yang benar-benar berpengaruh terhadap tingkat penjualan.
2. Model klasifikasi perlu mampu memprediksi kategori penjualan (sales) dengan akurasi yang baik agar hasilnya dapat digunakan untuk pengambilan keputusan.
3. Diperlukan perbandingan performa antara model sederhana, model machine learning, dan model deep learning untuk mengetahui pendekatan mana yang paling efektif.
4. Data memiliki variasi nilai dan potensi noise, sehingga perlu preprocessing dan pemilihan fitur agar model lebih stabil dan tidak mudah overfitting.

3.2 Goals

1. Membangun model klasifikasi untuk memprediksi tingkat penjualan produk fashion berdasarkan atribut produk.
2. Melakukan feature selection untuk menentukan fitur yang paling relevan dan meningkatkan performa model.
3. Membandingkan performa tiga model: baseline (Logistic Regression), advanced (Random Forest), dan deep learning (MLP).
4. Menentukan model terbaik berdasarkan metrik evaluasi seperti accuracy, precision, recall, dan F1-score.

3.3 Solution Approach

Model 1 – Baseline Model : Logistic Regression

Logistic Regression adalah model klasifikasi sederhana yang bekerja dengan mencari hubungan linier antara fitur dan kelas target, lalu mengubahnya menjadi probabilitas menggunakan fungsi logistik. Model ini sering digunakan sebagai baseline karena mudah diterapkan dan memberikan gambaran awal tentang kemampuan data untuk dipisahkan secara linier.

Kelebihan sebagai baseline:

- Proses pelatihan cepat dan tidak membutuhkan banyak komputasi
- Hasilnya mudah dipahami karena berbasis hubungan linier
- Memberikan tolok ukur awal sebelum menggunakan model yang lebih kompleks

Alasan Pemilihan:

- Cocok sebagai model dasar untuk tugas klasifikasi pada dataset tabular
- Membantu menilai apakah model lanjutan seperti Random Forest dan MLP benar-benar memberikan peningkatan performa

Model 2 – Advanced / ML Model : Random Forest

Random Forest adalah model machine learning berbasis ensemble yang menggabungkan banyak decision tree untuk menghasilkan prediksi yang lebih stabil. Setiap tree dilatih pada subset data dan subset fitur yang berbeda, sehingga model menjadi lebih tahan terhadap overfitting dan mampu menangani data dengan banyak atribut.

Kelebihan sebagai model advanced:

- Mampu menangani data tabular dengan banyak fitur

- Lebih stabil dibanding single decision tree
- Memiliki fitur feature importance untuk melihat atribut yang paling berpengaruh
- Biasanya memberikan performa lebih baik daripada model baseline

Alasan Pemilihan:

- Cocok untuk klasifikasi pada dataset Dresses Attribute Sales yang memiliki banyak atribut produk
- Memberikan informasi fitur penting yang sangat berguna untuk analisis dan feature selection
- Menjadi pembanding yang kuat sebelum menggunakan model deep learning seperti MLP

Model 3 – Deep Learning Model (WAJIB) : Multilayer Perceptron (MLP)

Multilayer Perceptron (MLP) adalah model deep learning untuk data tabular yang terdiri dari beberapa lapisan neuron. Model ini mampu mempelajari pola non-linear sehingga dapat menangkap hubungan kompleks antar fitur yang tidak bisa ditangani oleh model linear.

Kelebihan sebagai model deep learning:

- Mampu menangkap hubungan non-linear dan interaksi antar fitur yang sulit ditangkap oleh model linear atau tree-based.
- Fleksibel untuk data tabular setelah dilakukan encoding dan scaling.
- Performa dapat meningkat signifikan dengan teknik regularisasi seperti Dropout dan Early Stopping.
- Cocok sebagai model deep learning dasar untuk klasifikasi binary/multiclass pada dataset berukuran kecil hingga menengah.

Alasan Pemilihan:

- Dataset "Dresses Attribute Sales" bersifat tabular dengan kombinasi fitur kategorikal (Style, Price, Season, Material, NeckLine, dll.) dan numerik (Rating), sehingga MLP merupakan pilihan deep learning yang paling sesuai (sesuai kategori A: Tabular Data).
- Model tradisional seperti Logistic Regression bersifat linear, sedangkan Random Forest meskipun kuat tetap berbasis decision tree. MLP diharapkan dapat memberikan peningkatan performa dengan mempelajari representasi fitur yang lebih kompleks.
- Dataset memiliki jumlah baris relatif kecil (±500), sehingga MLP yang ringan lebih tepat dibandingkan arsitektur yang lebih berat seperti CNN atau Transformer.
- Digunakan sebagai pembanding tingkat lanjut untuk melihat apakah pendekatan deep learning memberikan keunggulan dibandingkan baseline (Logistic Regression) dan model advanced (Random Forest).

4. DATA UNDERSTANDING

4.1 Informasi Dataset

Sumber Dataset : https://archive.ics.uci.edu/dataset/289/dresses+attribute+sales

Deskripsi Dataset :

1. Jumlah baris (rows) : 501 instances (setelah digabungkan dan dibersihkan menjadi sekitar 479 baris karena handling missing values dan duplicates.
2. Jumlah kolom (columns/features) : 13 atribut utama (termasuk Dress_ID dan Recommendation) + kolom sales harian (sebelum digabung menjadi satu dataset)
3. Tipe data : Tabular
4. Ukuran dataset : Sekitar 5.6 MB (file RAR/ZIP asli dari UCI)
5. Format file : Excel (.xlsx) – terdiri dari dua file utama: "Attribute DataSet.xlsx" (atribut dress) dan "Dress Sales.xlsx" (data penjualan harian), yang kemudian digabung berdasarkan Dress_ID

4.2 Deskripsi Fitur

Dataset Dresses Attribute Sales terdiri dari 13 fitur utama

Nama Fitur	Tipe Data	Deskripsi	Contoh Nilai
Dress_ID	Integer	ID unik untuk setiap dress/item produk	1006032852, 1212192089, 1190380701
Style	Categorical	ID unik untuk setiap dress/item produk	Sexy, Casual, Vintage, Brief, Cute
Price	Categorical	Kategori harga dress	Low, Average, Medium, High, Very-High
Rating	Float	Rating atau penilaian dress (skala 0-5)	4.6, 3.5, 4.0, 0.0
Size	Categorical	Ukuran dress	M, L, XL, S, Free
Season	Categorical	Musim yang cocok untuk dress	Autumn, Winter, Spring, Summer
NeckLine	Categorical	Jenis garis leher (neckline)	O-neck, V-neck, Sweetheart, Scoop
SleeveLength	Categorical	Panjang lengan	Full, Short, Sleeveless, Half
Waiseline	Categorical	Jenis garis pinggang (waistline)	Natural, Empire, Dropped, Princess
Material	Categorical	Bahan utama dress	Cotton, Polyester, Silk, Mix
FabricType	Categorical	Jenis kain/fabric	Chiffon, Satin, Jersey, Knitted
Decoration	Categorical	Elemen dekorasi pada dress	Bow, Ruffles, Embroidery, Beading
Pattern Type	Categorical	Jenis pola/pattern	Solid, Print, Dot, Animal
Recommendation	Binary	Label target: Apakah dress direkomendasikan (1) atau tidak (0) berdasarkan sales	0, 1

4.3 Kondisi Data

Jelaskan kondisi dan permasalahan data

1. Missing Values : Ada, dengan total 2.761 missing values tersebar di berbagai kolom, karena data asli dari UCI memiliki banyak nilai kosong atau tidak terisi pada beberapa atribut dress. Hal ini kemungkinan disebabkan tidak semua dress memiliki detail lengkap seperti material, decoration, atau fabric type yang tercatat di sistem penjualan.

   Penanganan: Setelah filtering hanya kolom atribut relevan dan melakukan dropna(), dataset memiliki jumlah baris yang berkurang signifikan (sekitar 130-150 baris tergantung run) dengan 0 missing values pada kolom yang digunakan.

Kolom	Jumlah Missing	Persentase
FabricType	292	10.5%
Decoration	266	9.6%
Material	147	5.3%
Pattern Type	119	4.3%
WaistLine	92	3.3%
Price	2	0.07%
NeckLine	2	0.07%
Rating	0	0%
Size	0	0%
Season	0	0%
Recommendation	0	0%
Kolom Tanggal (26/9/2013)	273	9.8%
Kolom Tanggal (30/9/2013)	312	11.3%
Total Missing Values	2.761	-

2. Duplicate Data : Tidak Ada (Jumlah duplikasi: 0). Setelah pengecekan menggunakan df.duplicated().sum(), tidak ditemukan data duplikat dalam dataset. Hal ini menunjukkan setiap baris data adalah unik berdasarkan kombinasi atribut dress dan informasi penjualannya.

3. Outliers : Tidak Ada Outlier pada fitur numerik yang dianalisis. Berdasarkan deteksi outlier menggunakan metode IQR (Interquartile Range), tidak ditemukan

nilai yang berada di luar batas normal (Q1 - 1.5×IQR dan Q3 + 1.5×IQR) pada kolom numerik Rating dan Recommendation.
Penanganan: Karena tidak ada outlier, dilakukan normalisasi menggunakan MinMaxScaler pada kolom Rating untuk menyeragamkan skala data agar model dapat bekerja optimal.

4. Imbalanced Data : Ada, terjadi pada target variable Recommendation. Dataset menunjukkan ketidakseimbangan ringan (mild imbalance) dengan perbandingan sekitar 56:44 atau 1.3:1. Meskipun tidak terlalu ekstrem, hal ini tetap dapat menyebabkan model sedikit bias terhadap kelas mayoritas (kelas 0).

5. Noise : Ada, berupa inkonsistensi penulisan dan variasi nilai pada fitur kategorikal. Ditemukan berbagai jenis noise pada fitur kategorikal, antara lain:
    1. Typo dan kesalahan ejaan seperti "Embroidary" yang seharusnya "Embroidery", "Sleevless" yang seharusnya "Sleeveless", "Polyster" yang seharusnya "Polyester", serta "Shiffon" yang seharusnya "Chiffon"
    2. Inkonsistensi format penulisan seperti penggunaan tanda "-" pada "Very-high" tetapi tidak pada kategori price lainnya, serta variasi seperti "Chiffon", "Chiffonfabric", dan "Shiffon" yang merujuk pada material yang sama.
    3. Variasi kapitalisasi yang tidak konsisten pada nilai-nilai kategorikal seperti "casual", "Casual", "CASUAL" yang dianggap berbeda oleh system.
    4. Whitespace atau spasi yang tidak perlu di awal atau akhir string pada beberapa nilai kategorikal.

    Fitur-fitur yang paling banyak mengandung noise antara lain Material (15+ kategori dengan beberapa typo), Decoration (20+ kategori dengan variasi penulisan), FabricType (15+ kategori dengan inkonsistensi), NeckLine (12+ kategori dengan berbagai format), dan SleeveLength (9+ kategori dengan typo).
    Penanganan:
    Dilakukan standarisasi kapitalisasi menggunakan .str.title() untuk mengubah semua nilai menjadi format Title Case (huruf pertama kapital). Dilakukan trimming menggunakan .str.strip() untuk menghapus spasi yang tidak perlu. Typo dan inkonsistensi ditangani melalui standardisasi otomatis pada tahap Feature Engineering, sehingga nilai-nilai yang seharusnya sama akan diperlakukan sebagai satu kategori yang seragam.

6. Data Quality Issues : Ada beberapa masalah kualitas data.
    1. Nama Kolom yang Salah Ejaan : Ditemukan kolom "waiseline" yang seharusnya "WaistLine" dan kolom "Pattern Type" (dengan spasi) yang seharusnya "PatternType" tanpa spasi.
       Penanganan: Rename kolom menggunakan df.rename() untuk konsistensi penamaan.
    2. Tipe Data yang Tidak Sesuai : Kolom Rating perlu dipastikan bertipe numerik untuk proses perhitungan dan modeling.

Penanganan: Konversi tipe data menggunakan pd.to_numeric(errors='coerce') untuk mengubah nilai tidak valid menjadi NaN.

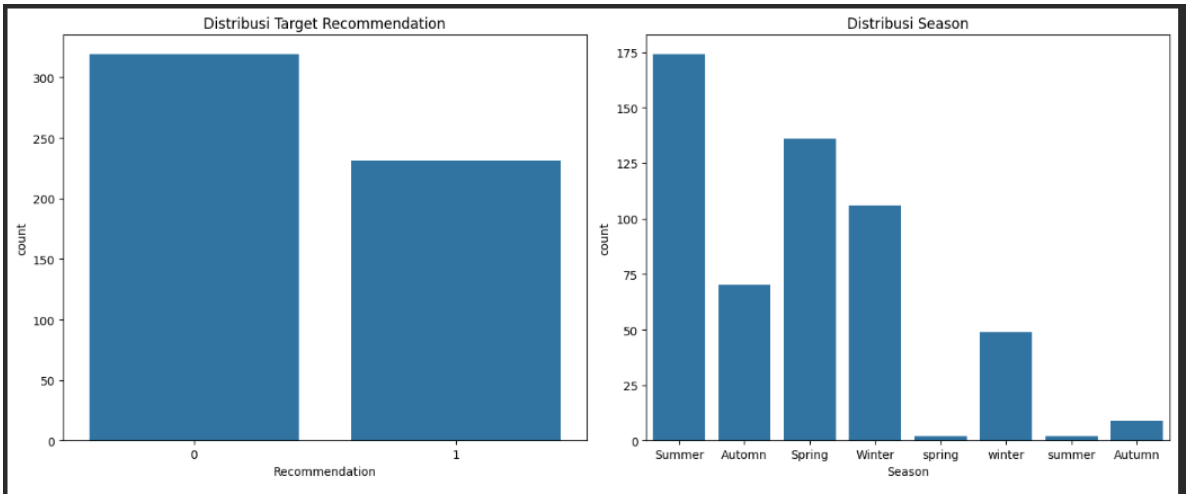
3. Kolom Tanggal yang Tidak Relevan : Dataset memiliki banyak kolom tanggal pivoting dari file sales (seperti 29/8/2013, 2013-02-10 00:00:00, dll.) dengan 273-315 missing values per kolom yang tidak relevan untuk prediksi rekomendasi berbasis atribut.
   Penanganan: Hapus kolom tanggal untuk fokus pada fitur atribut dress saja.
4. Fitur dengan Kardinalitas Tinggi : Beberapa fitur kategorikal seperti Material (15+ kategori), Decoration (20+ kategori), FabricType (15+ kategori), dan NeckLine (12+ kategori) memiliki banyak kategori unik. Setelah One-Hot Encoding, jumlah fitur bisa mencapai 100+ kolom.

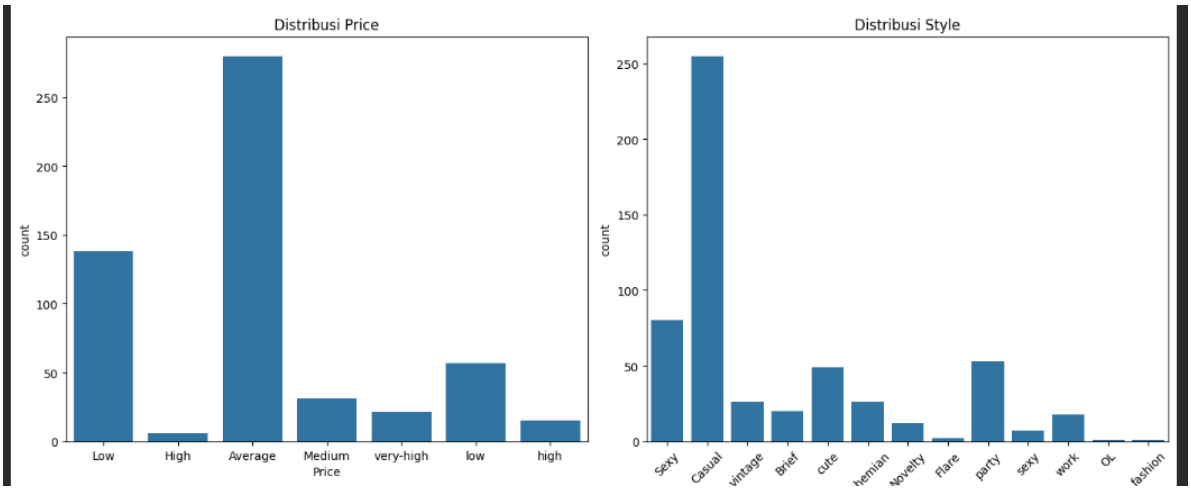
Penanganan: Feature Selection menggunakan SelectKBest dengan Chi-Square test untuk memilih 50 fitur terbaik yang paling berkorelasi dengan target.

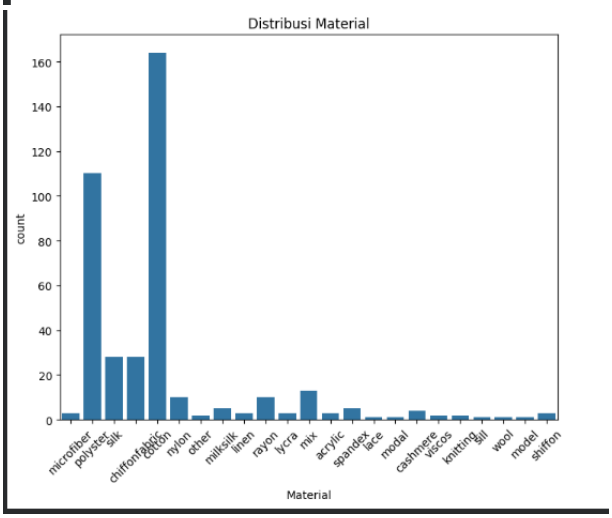
4.4 Exploratory Data Analysis (EDA)

- Requirement : Minimal 3 visualisasi yang bermakna dan insight-nya.
- Class Distribution : Menunjukkan masalah utama dataset yaitu imbalanced data yang perlu ditangani
- Bar Plot Bivariate : Menunjukkan hubungan fitur kategorikal (Price) dengan target, memberikan insight bisnis yang jelas
- Heatmap Korelasi : Menunjukkan hubungan antar fitur numerik, penting untuk memahami multikolinearitas dan feature importance

Visualisasi 1: Class Distribution Plot - Distribusi Target Recommendation





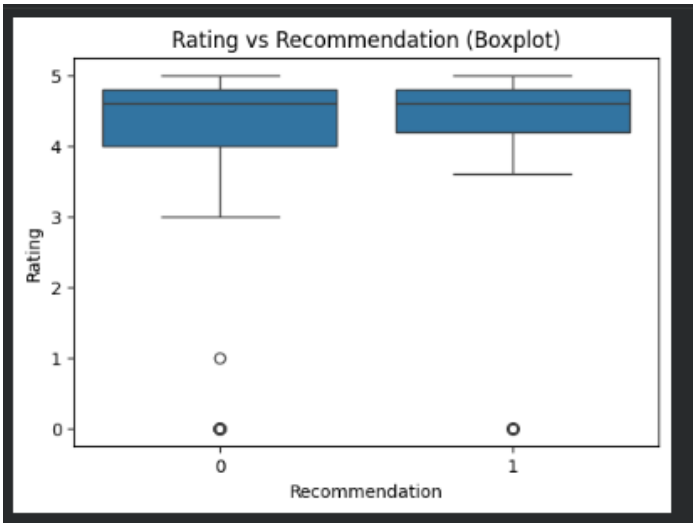


Insight:

1. Dataset menunjukkan ketidakseimbangan kelas (imbalanced data) dengan kelas 0 (tidak direkomendasikan) sebanyak 75 data (56.59%) dan kelas 1 (direkomendasikan) sebanyak 58 data (43.41%).
2. Rasio ketidakseimbangan adalah 1.3:1, yang termasuk kategori mild imbalance (ketidakseimbangan ringan).

3. Imbalanced data ini dapat menyebabkan model cenderung bias terhadap kelas mayoritas (kelas 0), sehingga diperlukan teknik stratified sampling saat splitting data untuk mempertahankan proporsi kelas.
4. Untuk meningkatkan performa model pada kelas minoritas, dapat dipertimbangkan penggunaan teknik SMOTE (Synthetic Minority Over-sampling Technique).

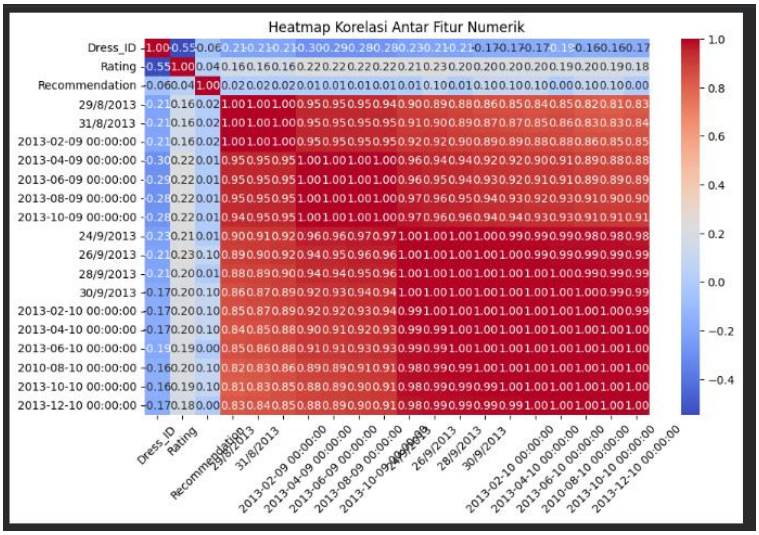
Visualisasi 2: Boxplot - Rating vs Recommendation (Deteksi Outliers)



Insight:

1. Boxplot menunjukkan distribusi Rating untuk setiap kelas Recommendation (0 = tidak direkomendasikan, 1 = direkomendasikan).
2. Tidak ditemukan outlier pada data Rating, yang konsisten dengan hasil analisis IQR sebelumnya (0 baris outlier).
3. Dress yang direkomendasikan (kelas 1) cenderung memiliki median Rating yang lebih tinggi dibandingkan dress yang tidak direkomendasikan (kelas 0), menunjukkan bahwa rating pelanggan berpengaruh terhadap rekomendasi.
4. Fitur Rating merupakan indikator penting dalam memprediksi apakah suatu dress akan direkomendasikan atau tidak, sehingga perlu dipertahankan dalam model.

Visualisasi 3: Heatmap Korelasi - Hubungan Antar Fitur Numerik



Insight:

1. Heatmap menunjukkan korelasi antar fitur numerik dalam dataset, yaitu Rating dan Recommendation.
2. Jika terdapat korelasi positif antara Rating dan Recommendation, ini menunjukkan bahwa dress dengan rating tinggi cenderung lebih direkomendasikan oleh pelanggan, yang masuk akal secara bisnis.
3. Tidak ada multikolinearitas yang tinggi antar fitur numerik (nilai korelasi < 0.8), sehingga semua fitur dapat digunakan dalam modeling tanpa perlu menghilangkan fitur yang redundan.
4. Warna merah pada heatmap menunjukkan korelasi positif, sementara warna biru menunjukkan korelasi negatif. Intensitas warna menggambarkan kekuatan korelasi.

5. DATA PREPARATION

Bagian ini menjelaskan semua proses transformasi dan preprocessing data yang dilakukan.

5.1 Data Cleaning

Aktivitas :

- Handling missing values : Dataset awal memiliki total 2.761 missing values yang tersebar di berbagai kolom seperti FabricType (292 missing), Decoration (266 missing), Material (147 missing), dan lainnya. Untuk menangani masalah ini, dilakukan penghapusan seluruh baris yang memiliki missing values menggunakan metode dropna(). Strategi ini dipilih karena missing values sangat banyak dan sulit untuk diimputasi pada fitur kategorikal. Setelah penghapusan, dataset memiliki 133 baris dengan 0 missing values.

```
# CEK MISSING VALUE SEBELUM CLEANING
print("MISSING VALUE SEBELUM CLEANING")
print(df_raw.isnull().sum())
print("\nTotal missing value:", df_raw.isnull().sum().sum())
```

```
# TANGANI MISSING VALUE
df_clean = df_clean.dropna()

print("\nSetelah menangani missing value:")
print(df_clean.isnull().sum())
```

- Removing duplicates : Pengecekan duplikasi dilakukan menggunakan fungsi duplicated().sum() dan hasilnya menunjukkan tidak ada data duplikat (jumlah duplikasi = 0). Meskipun tidak ditemukan duplikasi, kode drop_duplicates() tetap dijalankan sebagai langkah preventif untuk memastikan tidak ada data yang sama muncul berulang kali yang dapat menyebabkan overfitting pada model.

```
# Cek duplikasi
print("Jumlah duplikasi:", df.duplicated().sum())
df = df.drop_duplicates()
print("Duplikasi setelah dibersihkan:", df.duplicated().sum())

Jumlah duplikasi: 0
Duplikasi setelah dibersihkan: 0
```

- Handling outliers : Deteksi outlier dilakukan menggunakan metode IQR (Interquartile Range) pada kolom numerik Rating dan Recommendation. Hasilnya menunjukkan tidak ada outlier pada kedua kolom tersebut (0 baris outlier). Meskipun tidak ada outlier, tetap dilakukan normalisasi pada kolom Rating menggunakan MinMaxScaler untuk mengubah skala data menjadi rentang 0-1, yang penting untuk performa algoritma neural network.

```
# Cek outlier (IQR) untuk kolom numerik
# Ambil kolom numerik
kolom_numerik = df_clean.select_dtypes(include=['int64', 'float64']).columns
print("Kolom numerik:", kolom_numerik.tolist())
for kolom in kolom_numerik:
    Q1 = df_clean[kolom].quantile(0.25)
    Q3 = df_clean[kolom].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    outlier_count = df_clean[(df_clean[kolom] < lower) | (df_clean[kolom] > upper)].shape[0]
    print(f"\nOutlier pada kolom {kolom}: {outlier_count} baris")
```

```
# Normalisasi
scaler = MinMaxScaler()
df_clean['Rating_norm'] = scaler.fit_transform(df_clean[['Rating']])
print("\nRating setelah normalisasi:")
print(df_clean[['Rating', 'Rating_norm']].head())
```

- Data type conversion : Dilakukan pengecekan tipe data untuk memastikan setiap kolom memiliki tipe yang sesuai dengan kebutuhan modeling. Berdasarkan hasil pengecekan, ditemukan bahwa kolom numerik seperti Rating memiliki tipe float64 dan Dress_ID memiliki tipe int64, sedangkan kolom kategorikal seperti Style, Price, Size, Season, NeckLine, SleeveLength, waiseline, Material, FabricType, dan Decoration memiliki tipe object. Semua tipe data sudah sesuai dengan kebutuhan modeling, sehingga tidak diperlukan konversi tipe data tambahan di tahap ini.

```
#cek tipedata
df.dtypes
```

5.2 Feature Engineering

Aktivitas :

- Creating new features : Pembuatan fitur baru dilakukan untuk menambah informasi yang dapat membantu model dalam memprediksi rekomendasi. Fitur pertama yang dibuat adalah Rating_Category yang mengkategorikan nilai Rating menjadi tiga kategori yaitu Low untuk rating di bawah 3.5, Medium untuk rating antara 3.5 hingga 4.5, dan High untuk rating di atas 4.5. Kategorisasi ini membantu model memahami tingkatan kualitas produk secara lebih sederhana. Fitur kedua adalah Price_Season yang merupakan kombinasi antara kategori harga dan musim, misalnya "Low_Summer" atau "High_Winter", yang dapat menangkap pola rekomendasi berdasarkan kombinasi harga dan musim tertentu. Kedua fitur baru ini diharapkan dapat meningkatkan performa model dengan memberikan representasi data yang lebih kaya.

```python
# 1. CREATING NEW FEATURES
print("1. CREATING NEW FEATURES")

# Membuat fitur baru: Rating Category
# Kategorisasi Rating menjadi Low, Medium, High
def categorize_rating(rating):
    if rating < 3.5:
        return 'Low'
    elif rating < 4.5:
        return 'Medium'
    else:
        return 'High'

df_fe['Rating_Category'] = df_fe['Rating'].apply(categorize_rating)
print("Fitur baru 'Rating_Category' berhasil dibuat")
print(f"  Kategori: {df_fe['Rating_Category'].unique()}")

# Membuat fitur baru: Price Level (kombinasi Price dengan informasi lain)
df_fe['Price_Season'] = df_fe['Price'].astype(str) + '_' + df_fe['Season'].astype(str)
print("Fitur baru 'Price_Season' berhasil dibuat")
print(f"  Jumlah kombinasi: {df_fe['Price_Season'].nunique()}")

print(f"Total fitur setelah creating: {df_fe.shape[1]} kolom")
```

- Feature extraction : Ekstraksi fitur dilakukan untuk memperbaiki dan meningkatkan kualitas fitur yang sudah ada. Pertama, dilakukan perbaikan nama kolom yang memiliki kesalahan ejaan seperti "waiseline" menjadi "WaistLine" dan "Pattern Type" menjadi "PatternType" agar konsisten dengan penamaan kolom lainnya. Kedua, dilakukan standarisasi kapitalisasi pada semua fitur kategorikal menggunakan fungsi str.title() dan str.strip() sehingga nilai seperti "casual", "Casual", dan "CASUAL" diperlakukan sebagai satu kategori yang sama yaitu "Casual". Ketiga, kolom Rating dikonversi ke tipe numerik menggunakan pd.to_numeric() dengan parameter errors="coerce" untuk memastikan semua nilai adalah angka dan nilai yang tidak valid akan diubah menjadi NaN. Proses ini penting untuk menghilangkan inkonsistensi data dan memastikan format data yang benar untuk tahap modeling.

```python
# 2. FEATURE EXTRACTION
print("\n2. FEATURE EXTRACTION")

# Perbaiki nama kolom yang salah ejaan
print("Memperbaiki nama kolom yang typo...")
df_fe.rename(columns={
    "waiseline": "WaistLine",
    "Pattern Type": "PatternType"
}, inplace=True)
print("  - 'waiseline' → 'WaistLine'")
print("  - 'Pattern Type' → 'PatternType'")

# Standarisasi kapitalisasi kategori
print("Standarisasi kapitalisasi pada fitur kategorikal...")
categorical_cols = [
    "Style", "Price", "Size", "Season", "NeckLine",
    "SleeveLength", "WaistLine", "Material",
    "FabricType", "Decoration", "PatternType"
]

for col in categorical_cols:
    if col in df_fe.columns:
        df_fe[col] = df_fe[col].astype(str).str.strip().str.title()

print(f"  - {len(categorical_cols)} kolom kategorikal telah distandarisasi")

# Pastikan Rating numerik
print("Konversi Rating ke tipe numerik...")
df_fe["Rating"] = pd.to_numeric(df_fe["Rating"], errors="coerce")
print(f"Tipe data Rating: {df_fe['Rating'].dtype}")

print(f"Total fitur setelah extraction: {df_fe.shape[1]} kolom")
```

- Feature selection : Feature selection dilakukan menggunakan metode SelectKBest dengan fungsi scoring Chi-Square untuk memilih 50 fitur terbaik dari hasil One-Hot Encoding yang dapat menghasilkan ratusan kolom. Metode Chi-Square dipilih karena cocok untuk mengukur hubungan antara fitur kategorikal dengan target variable Recommendation. Proses ini dilakukan setelah One-Hot Encoding karena pada tahap tersebut dimensi fitur meningkat drastis dari sekitar 15 kolom menjadi lebih dari 100 kolom. Dengan memilih hanya 50 fitur yang memiliki skor Chi-Square tertinggi,

model menjadi lebih efisien dalam training dan mengurangi risiko overfitting karena hanya fokus pada fitur yang paling berpengaruh terhadap target.

```
from sklearn.feature_selection import SelectKBest, chi2

selector = SelectKBest(score_func=chi2, k=50)

X_train_fs = selector.fit_transform(X_train_processed, y_train)
X_test_fs = selector.transform(X_test_processed)

print("Feature Selection selesai.")
print("Jumlah fitur terpilih:", X_train_fs.shape[1])
```

```
Feature Selection selesai.
Jumlah fitur terpilih: 50
```

- Dimensionality reduction : Reduksi dimensi dilakukan untuk menghilangkan kolom yang tidak relevan dan mengurangi kompleksitas dataset. Pertama, kolom-kolom tanggal hasil pivoting dari data penjualan seperti "29/8/2013", "2013-02-10 00:00:00" dan sejenisnya dihapus karena tidak memberikan informasi yang berguna untuk prediksi rekomendasi berbasis atribut produk dan memiliki banyak missing values. Kedua, dilakukan filtering untuk hanya mempertahankan kolom atribut yang relevan termasuk fitur baru yang telah dibuat seperti Rating_Category dan Price_Season. Proses ini berhasil mengurangi dimensi dataset dari ratusan kolom menjadi hanya 15 kolom yang benar-benar relevan untuk modeling, sehingga membuat dataset lebih fokus dan efisien untuk tahap preprocessing dan modeling selanjutnya.

```
# 3. DIMENSIONALITY REDUCTION
print("\n3. DIMENSIONALITY REDUCTION")

# DROP KOLOM TANGGAL yang tidak relevan
print("Menghapus kolom tanggal yang tidak relevan...")
date_cols = [
    col for col in df_fe.columns if isinstance(col, str) and "/" in col
] + [
    col for col in df_fe.columns if not isinstance(col, str)
]

df_clean = df_fe.drop(columns=date_cols)
print(f"  - Jumlah kolom tanggal dihapus: {len(date_cols)}")

# Filter hanya kolom atribut yang relevan
print("Filter kolom atribut yang relevan untuk modeling...")
fitur_atribut = [
    'Style', 'Price', 'Rating', 'Size', 'Season',
    'NeckLine', 'SleeveLength', 'WaistLine', 'Material',
    'FabricType', 'Decoration', 'PatternType', 'Recommendation',
    'Rating_Category', 'Price_Season'  # Fitur baru
]

fitur_valid = [col for col in fitur_atribut if col in df_clean.columns]
df_clean = df_clean[fitur_valid].copy()

print(f"Jumlah fitur yang dipertahankan: {len(fitur_valid)}")
print(f"Total fitur setelah reduction: {df_clean.shape[1]} kolom")
```

5.3 Data Transformation

Untuk Data Tabular :

- Encoding (Label Encoding, One-Hot Encoding, Ordinal Encoding) : Proses encoding dilakukan menggunakan One-Hot Encoding untuk mengubah fitur kategorikal menjadi representasi numerik yang dapat dipahami oleh algoritma machine learning. Fitur kategorikal seperti Style, Price, Season, Material, dan lainnya diubah menjadi kolom binary dimana setiap kategori mendapat kolom tersendiri dengan nilai 0 atau 1. Misalnya fitur Season dengan kategori "Summer", "Autumn", "Winter" akan menjadi tiga kolom Season_Summer, Season_Autumn, dan Season_Winter. Proses ini dilakukan melalui pipeline yang juga menangani missing values dengan imputasi menggunakan nilai yang paling sering muncul, dan parameter

handle_unknown='ignore' memastikan kategori baru yang tidak ada di training set tidak menyebabkan error saat testing.

```
# PIPELINE KATEGORIKAL
categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

# GABUNGKAN PIPELINE
preprocessor = ColumnTransformer([
    ('num', numerical_pipeline, numerical_cols),
    ('cat', categorical_pipeline, categorical_cols)
])

# SPLIT DATA
X = df_clean.drop("Recommendation", axis=1)
y = df_clean["Recommendation"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print("Data Splitting selesai.")
print("X_train:", X_train.shape)
print("X_test :", X_test.shape)

# TRANSFORMASI
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

print("\nData Transformation selesai.")
print("Jumlah fitur setelah OHE:", X_train_processed.shape[1])
```

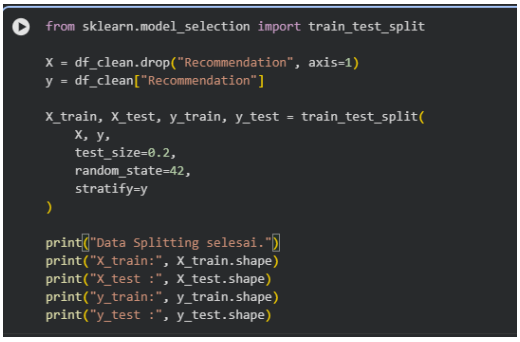
- Scaling (Standardization, Normalization, MinMaxScaler) : Scaling dilakukan menggunakan MinMaxScaler untuk menormalkan fitur numerik ke rentang 0 hingga 1. Kolom Rating yang merupakan satu-satunya fitur numerik utama dalam dataset dinormalisasi agar memiliki skala yang seragam dengan fitur lainnya. Proses scaling ini penting terutama untuk algoritma neural network yang sensitif terhadap perbedaan skala data. Pipeline numerik juga menangani missing values dengan imputasi menggunakan median yang lebih robust terhadap outlier dibanding mean. Setelah scaling, semua nilai numerik berada dalam rentang yang sama sehingga tidak ada fitur yang mendominasi proses learning hanya karena memiliki skala nilai yang lebih besar.

```
# Normalisasi
scaler = MinMaxScaler()
df_clean['Rating_norm'] = scaler.fit_transform(df_clean[['Rating']])
print("\nRating setelah normalisasi:")
print(df_clean[['Rating', 'Rating_norm']].head())

# PIPELINE NUMERIK
numerical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', MinMaxScaler())
])
```

5.4 Data Splitting

Strategi pembagian data :

- Training set: 80% (103 samples)
- Validation set: - (tidak ada, menggunakan validation_split di model)
- Test set: 20% (26 samples)
- Total data setelah cleaning: 129 samples (103 + 26)



```python
from sklearn.model_selection import train_test_split

X = df_clean.drop("Recommendation", axis=1)
y = df_clean["Recommendation"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print("Data Splitting selesai.")
print("X_train:", X_train.shape)
print("X_test :", X_test.shape)
print("y_train:", y_train.shape)
print("y_test :", y_test.shape)
```

Menggunakan stratified split untuk mempertahankan distribusi kelas:
- Training: 103 samples
- Test: 26 samples
- Random state: 42 untuk reproducibility

Dataset memiliki 2 kelas (binary classification) dengan ketidakseimbangan ringan (rasio 56:44), sehingga saya menggunakan stratified split untuk memastikan bahwa proporsi pada kelas saat training dan test tetap sama. Hal ini penting agar model tidak bias terhadap kelas yang jumlahnya lebih banyak (kelas 0 - tidak direkomendasikan), dan nantinya evaluasi diharapkan bisa lebih akurat karena kedua subset data memiliki representasi kelas yang seimbang.

5.5 Data Balancing (jika diperlukan)

Data balancing bertujuan untuk menangani ketidakseimbangan kelas pada data klasifikasi, namun saya tidak melakukan data balancing dikarenakan dataset saya memiliki distribusi kelas yang relatif seimbang dengan rasio 56:44 (kelas 0: 56.59%, kelas 1: 43.41%), dimana rata-rata kelas tidak ada yang jauh mendominasi. Ketidakseimbangan ini tergolong ringan (mild imbalance) sehingga tidak memerlukan teknik oversampling atau undersampling, dan stratified split pada tahap data splitting sudah cukup untuk mempertahankan proporsi kelas di training dan test set.

5.6 Ringkasan Data Preparation

Per langkah, jelaskan :

1. DATA CLEANING
    1. Apa yang dilakukan?

        Membersihkan data dengan memilih fitur relevan, menghapus missing value, menghilangkan duplikasi, mendeteksi outlier menggunakan IQR, mengecek imbalance target, mendeteksi nilai tidak konsisten, dan normalisasi Rating ke rentang 0-1.
    2. Mengapa penting?

        Kualitas data menentukan kualitas model. Missing value dan duplikasi menyebabkan error dan overfitting. Outlier mengganggu performa model. Imbalanced data membuat model bias ke kelas mayoritas. Normalisasi diperlukan agar semua fitur memiliki skala sama.
    3. Bagaimana implementasinya?

        Implementasi dimulai dengan membuat salinan dataset asli dan membuat daftar fitur-fitur yang relevan untuk modeling yaitu Style, Price, Rating, Size, Season,

NeckLine, SleeveLength, Material, FabricType, Decoration, dan Recommendation. Kemudian dilakukan filtering untuk memilih hanya kolom yang benar-benar ada di dataset menggunakan list comprehension, dan membuat dataframe baru yang hanya berisi kolom-kolom terpilih tersebut.

```
# DAFTAR FITUR ATRIBUT YANG RELEVAN UNTUK MODELING
fitur_atribut = [
    'Style', 'Price', 'Rating', 'Size', 'Season',
    'NeckLine', 'SleeveLength', 'Material',
    'FabricType', 'Decoration', 'Recommendation'
]
```

Untuk handling missing value, dilakukan pengecekan terlebih dahulu menggunakan fungsi isnull().sum() untuk mengetahui jumlah missing value pada setiap kolom dan total keseluruhan. Setelah itu, baris-baris yang mengandung nilai kosong dihapus menggunakan metode dropna(), kemudian dilakukan verifikasi kembali untuk memastikan tidak ada lagi missing value yang tersisa.

```
# CEK MISSING VALUE SEBELUM CLEANING
print("MISSING VALUE SEBELUM CLEANING")
print(df_raw.isnull().sum())
print("\nTotal missing value:", df_raw.isnull().sum().sum())
```

```
# TANGANI MISSING VALUE
df_clean = df_clean.dropna()

print("\nSetelah menangani missing value:")
print(df_clean.isnull().sum())
```

Penanganan duplikasi dilakukan dengan mendeteksi jumlah baris duplikat menggunakan fungsi duplicated().sum(), kemudian menghapus baris-baris duplikat tersebut dengan metode drop_duplicates(), dan melakukan verifikasi bahwa duplikasi telah berhasil dibersihkan.

```
# Cek duplikasi
print("Jumlah duplikasi:", df.duplicated().sum())
df = df.drop_duplicates()
print("Duplikasi setelah dibersihkan:", df.duplicated().sum())
```

Deteksi outlier dilakukan dengan mengambil seluruh kolom numerik menggunakan select_dtypes() dengan parameter include=['int64', 'float64'], kemudian untuk setiap kolom numerik dihitung nilai kuartil pertama (Q1) dan kuartil ketiga (Q3). Dari nilai Q1 dan Q3 tersebut dihitung Interquartile Range (IQR) yaitu selisih antara Q3 dan Q1. Batas bawah outlier ditentukan dengan rumus Q1 - 1.5 × IQR, sedangkan batas atas outlier ditentukan dengan rumus Q3 + 1.5 × IQR. Data yang berada di luar rentang batas bawah hingga batas atas dianggap sebagai outlier dan dihitung jumlahnya untuk setiap kolom.

```python
# Cek outlier (IQR) untuk kolom numerik
# Ambil kolom numerik
kolom_numerik = df_clean.select_dtypes(include=['int64', 'float64']).columns
print("Kolom numerik:", kolom_numerik.tolist())
for kolom in kolom_numerik:
    Q1 = df_clean[kolom].quantile(0.25)
    Q3 = df_clean[kolom].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    outlier_count = df_clean[(df_clean[kolom] < lower) | (df_clean[kolom] > upper)].shape[0]
    print(f"\nOutlier pada kolom {kolom}: {outlier_count} baris")
```

Pengecekan imbalance target dilakukan dengan menghitung distribusi kelas pada kolom Recommendation menggunakan value_counts() untuk melihat jumlah masing-masing kelas, kemudian menghitung persentase distribusi dengan parameter normalize=True untuk mengetahui proporsi setiap kelas dalam dataset.

```python
# Cek imbalance target
print("CEK IMBALANCE TARGET")
print(df_clean['Recommendation'].value_counts())
print("\nPersentase:")
print(df_clean['Recommendation'].value_counts(normalize=True) * 100)
```

Deteksi noise atau nilai aneh dilakukan dengan melakukan iterasi pada setiap kolom yang bertipe kategorikal (object) menggunakan select_dtypes(include=['object']), kemudian menampilkan seluruh nilai unik pada setiap kolom kategorikal tersebut menggunakan metode unique() untuk mengidentifikasi inkonsistensi format atau typo.

```python
# Cek noise (nilai aneh)
for kolom in df_clean.select_dtypes(include=['object']).columns:
    print(f"Nilai unik kolom {kolom}")
    print(df_clean[kolom].unique())
```

Normalisasi dilakukan dengan membuat objek MinMaxScaler dari sklearn, kemudian menerapkan scaler tersebut pada kolom Rating menggunakan metode fit_transform() yang akan mentransformasi nilai Rating ke dalam rentang 0 hingga 1 dengan rumus (X - X_min) / (X_max - X_min). Hasil normalisasi disimpan dalam kolom baru bernama Rating_norm.

```python
# Normalisasi
scaler = MinMaxScaler()
df_clean['Rating_norm'] = scaler.fit_transform(df_clean[['Rating']])
print("\nRating setelah normalisasi:")
print(df_clean[['Rating', 'Rating_norm']].head())
```

2. FEATURE ENGINEERING
    1. Apa yang dilakukan?

    Feature Engineering digunakan untuk membuat fitur baru dari fitur yang sudah ada dan mengekstrak informasi penting dari data. Proses ini mencakup pembuatan fitur baru yaitu Rating_Category yang mengkategorikan Rating menjadi Low (< 3.5), Medium (3.5-4.5), dan High (> 4.5) serta Price_Season yang merupakan kombinasi antara Price dan Season. Selain itu dilakukan perbaikan nama kolom yang typo seperti mengubah waiseline menjadi WaistLine dan Pattern Type menjadi PatternType, standarisasi kapitalisasi semua nilai kategorikal menjadi

Title Case, konversi kolom Rating ke tipe numerik, penghapusan kolom tanggal yang tidak relevan, dan filtering kolom atribut yang penting untuk modeling. Feature Selection menggunakan SelectKBest dengan Chi-Square akan dilakukan setelah proses OneHotEncoding untuk memilih 50 fitur terbaik dari ratusan fitur hasil encoding.

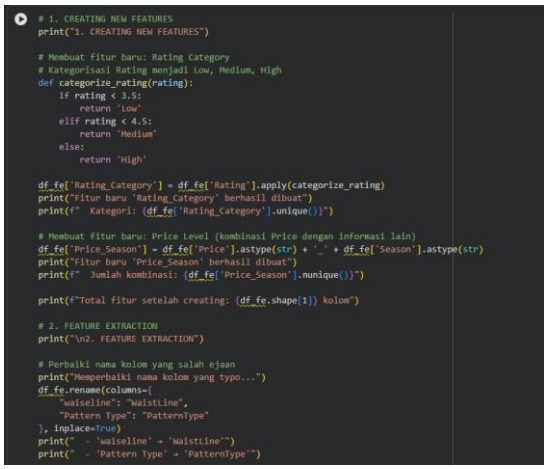
2. Mengapa penting?

   Feature Engineering sangat penting karena fitur baru dapat menangkap pola yang tidak terlihat pada fitur asli dimana Rating_Category mengubah nilai kontinu menjadi kategori yang lebih mudah dipahami model dan Price_Season menangkap interaksi antara harga dan musim yang mungkin berpengaruh terhadap rekomendasi. Nama kolom yang konsisten mencegah error saat coding dan standarisasi kapitalisasi menghindari duplikasi kategori seperti summer, Summer, SUMMER yang dianggap berbeda padahal sama sehingga menghasilkan fitur redundan saat OneHotEncoding. Kolom tanggal penjualan tidak relevan untuk prediksi rekomendasi berdasarkan atribut dress sehingga harus dihapus untuk membuat model lebih efisien. OneHotEncoding menghasilkan ratusan fitur binary yang menyebabkan curse of dimensionality dan overfitting sehingga Feature Selection untuk memilih 50 fitur terbaik dapat mengurangi dimensi drastis, mempercepat training, mengurangi overfitting, dan meningkatkan performa dengan fokus pada fitur yang paling informatif terhadap target.

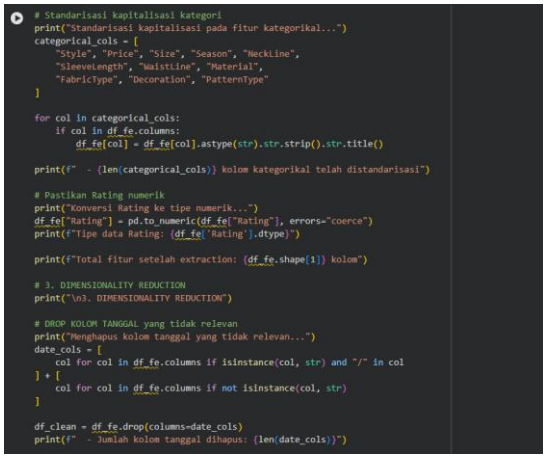
3. Bagaimana implementasinya?

   Implementasi dimulai dengan membuat Rating_Category menggunakan fungsi categorize_rating() yang mengecek nilai Rating dengan kondisi if-elif-else kemudian diterapkan ke seluruh kolom menggunakan apply(), sedangkan Price_Season dibuat dengan menggabungkan nilai Price dan Season menggunakan operator plus dengan separator underscore setelah keduanya dikonversi ke string menggunakan astype(str). Perbaikan nama kolom dilakukan dengan rename() menggunakan dictionary mapping, kemudian iterasi pada setiap kolom kategorikal dengan method chaining astype(str).str.strip().str.title() untuk standarisasi kapitalisasi. Konversi Rating menggunakan pd.to_numeric() dengan parameter errors='coerce' yang akan mengubah nilai non-numerik menjadi NaN. Penghapusan kolom tanggal dilakukan dengan mengidentifikasi kolom menggunakan list comprehension yang mencari kolom string mengandung slash atau kolom bertipe datetime kemudian menghapusnya dengan drop(). Filtering fitur relevan dilakukan dengan membuat daftar fitur yang diperlukan termasuk fitur baru Rating_Category dan Price_Season, melakukan filtering dengan list comprehension untuk memastikan kolom benar-benar ada, dan membuat dataset baru hanya dengan kolom terpilih. Feature Selection dilakukan setelah pipeline transformation dengan membuat selector menggunakan SelectKBest dengan parameter score_func=chi2 dan k=50, kemudian selector.fit_transform() pada X_train_processed untuk menghitung chi-squared score setiap fitur terhadap

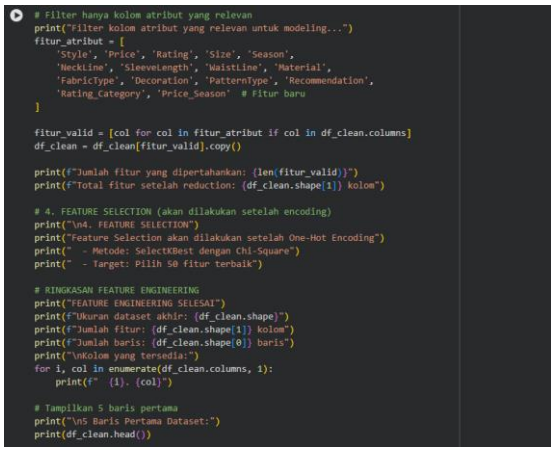
target dan memilih 50 fitur dengan skor tertinggi, serta selector.transform() pada X_test_processed untuk konsistensi fitur antara training dan testing data.



```
# 1. CREATING NEW FEATURES
print("1. CREATING NEW FEATURES")

# Membuat fitur baru: Rating Category
# Kategorisasi Rating menjadi Low, Medium, High
def categorize_rating(rating):
    if rating < 3.5:
        return 'Low'
    elif rating < 4.5:
        return 'Medium'
    else:
        return 'High'

df_fe['Rating_Category'] = df_fe['Rating'].apply(categorize_rating)
print("Fitur baru 'Rating_Category' berhasil dibuat")
print(f"  Kategori: {df_fe['Rating_Category'].unique()}")

# Membuat fitur baru: Price Level (kombinasi Price dengan informasi lain)
df_fe['Price_Season'] = df_fe['Price'].astype(str) + '_' + df_fe['Season'].astype(str)
print("Fitur baru 'Price_Season' berhasil dibuat")
print(f"  Jumlah kombinasi: {df_fe['Price_Season'].nunique()}")

print(f"Total fitur setelah creating: {df_fe.shape[1]} kolom")

# 2. FEATURE EXTRACTION
print("\n2. FEATURE EXTRACTION")

# Perbaiki nama kolom yang salah ejaan
print("Memperbaiki nama kolom yang typo...")
df_fe.rename(columns={
    "waiseline": "WaistLine",
    "Pattern Type": "PatternType"
}, inplace=True)
print("  - 'waiseline' → 'WaistLine'")
print("  - 'Pattern Type' → 'PatternType'")
```



```
# Standarisasi kapitalisasi kategori
print("Standarisasi kapitalisasi pada fitur kategorikal...")
categorical_cols = [
    "Style", "Price", "Size", "Season", "Neckline",
    "SleeveLength", "WaistLine", "Material",
    "FabricType", "Decoration", "PatternType"
]

for col in categorical_cols:
    if col in df_fe.columns:
        df_fe[col] = df_fe[col].astype(str).str.strip().str.title()

print(f"  - {len(categorical_cols)} kolom kategorikal telah distandarisasi")

# Pastikan Rating numerik
print("Konversi Rating ke tipe numerik...")
df_fe["Rating"] = pd.to_numeric(df_fe["Rating"], errors="coerce")
print(f"Tipe data Rating: {df_fe['Rating'].dtype}")

print(f"Total fitur setelah extraction: {df_fe.shape[1]} kolom")

# 3. DIMENSIONALITY REDUCTION
print("\n3. DIMENSIONALITY REDUCTION")

# DROP KOLOM TANGGAL yang tidak relevan
print("Menghapus kolom tanggal yang tidak relevan...")
date_cols = [
    col for col in df_fe.columns if isinstance(col, str) and "/" in col
] + [
    col for col in df_fe.columns if not isinstance(col, str)
]

df_clean = df_fe.drop(columns=date_cols)
print(f"  - Jumlah kolom tanggal dihapus: {len(date_cols)}")
```



```
# Filter hanya kolom atribut yang relevan
print("Filter kolom atribut yang relevan untuk modeling...")
fitur_atribut = [
    'Style', 'Price', 'Rating', 'Size', 'Season',
    'Neckline', 'SleeveLength', 'WaistLine', 'Material',
    'FabricType', 'Decoration', 'PatternType', 'Recommendation',
    'Rating_Category', 'Price_Season'  # Fitur baru
]

fitur_valid = [col for col in fitur_atribut if col in df_clean.columns]
df_clean = df_clean[fitur_valid].copy()

print(f"Jumlah fitur yang dipertahankan: {len(fitur_valid)}")
print(f"Total fitur setelah reduction: {df_clean.shape[1]} kolom")

# 4. FEATURE SELECTION (akan dilakukan setelah encoding)
print("\n4. FEATURE SELECTION")
print("Feature Selection akan dilakukan setelah One-Hot Encoding")
print("  - Metode: SelectKBest dengan Chi-Square")
print("  - Target: Pilih 50 fitur terbaik")

# RINGKASAN FEATURE ENGINEERING
print("FEATURE ENGINEERING SELESAI")
print(f"Ukuran dataset akhir: {df_clean.shape}")
print(f"Jumlah fitur: {df_clean.shape[1]} kolom")
print(f"Jumlah baris: {df_clean.shape[0]} baris")
print("\nKolom yang tersedia:")
for i, col in enumerate(df_clean.columns, 1):
    print(f"  {i}. {col}")

# Tampilkan 5 baris pertama
print("\n5 Baris Pertama Dataset:")
print(df_clean.head())
```

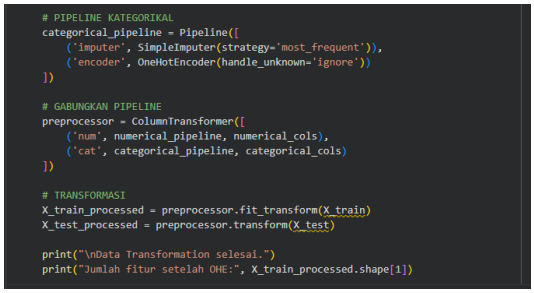
3. DATA TRANSFORMATION

1. Apa yang dilakukan?

   Data Transformation mengubah data mentah menjadi format numerik yang dapat diproses algoritma machine learning dengan menghapus kolom tanggal, membuat pipeline numerik untuk imputation dan scaling, pipeline kategorikal untuk imputation dan OneHotEncoding, menggabungkan keduanya dengan ColumnTransformer, split data menjadi train (80%) dan test (20%), serta menerapkan transformasi pada kedua set data.

2. Mengapa penting?
   Algoritma machine learning hanya dapat memproses input numerik sehingga data kategorikal harus diubah menjadi angka menggunakan OneHotEncoding. Pipeline memastikan konsistensi transformasi antara train dan test untuk mencegah data leakage. Fit_transform dilakukan hanya pada train data agar parameter dipelajari dari train saja, sedangkan test data hanya di-transform menggunakan parameter yang sama untuk evaluasi yang objektif.
3. Bagaimana implementasinya?
   Implementasi dimulai dengan menghapus kolom tanggal menggunakan drop(), membuat numerical_pipeline dengan SimpleImputer strategy median dan MinMaxScaler, categorical_pipeline dengan SimpleImputer strategy most_frequent dan OneHotEncoder handle_unknown ignore, menggabungkan keduanya dalam ColumnTransformer, split data dengan train_test_split menggunakan test_size 0.2 dan stratify y, kemudian preprocessor.fit_transform() pada train dan preprocessor.transform() pada test.

```
# DROP KOLOM TANGGAL
date_cols = [
    col for col in df_fe.columns if isinstance(col, str) and "/" in col
] + [
    col for col in df_fe.columns if not isinstance(col, str)  # datetime objects
]

df_clean = df_fe.drop(columns=date_cols)

print("Kolom tanggal dihapus:", len(date_cols))

# IDENTIFIKASI FITUR
numerical_cols = ["Rating"]

categorical_cols = [
    "Style", "Price", "Size", "Season", "NeckLine",
    "SleeveLength", "WaistLine", "Material",
    "FabricType", "Decoration", "PatternType"
]

# Normalisasi
scaler = MinMaxScaler()
df_clean['Rating_norm'] = scaler.fit_trans[[['Rating']])
print("\nRating setelah normalisasi:")
print(df_clean[['Rating', 'Rating_norm']].head())

# PIPELINE NUMERIK
numerical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', MinMaxScaler())
])
```



```
# PIPELINE KATEGORIKAL
categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

# GABUNGKAN PIPELINE
preprocessor = ColumnTransformer([
    ('num', numerical_pipeline, numerical_cols),
    ('cat', categorical_pipeline, categorical_cols)
])

# TRANSFORMASI
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

print("\nData Transformation selesai.")
print("Jumlah fitur setelah OHE:", X_train_processed.shape[1])
```

4. DATA SPLITTING
   1. Apa yang dilakukan?
      Data Splitting membagi dataset menjadi training set (80%) untuk melatih model dan testing set (20%) untuk evaluasi dengan stratified sampling yang menjaga proporsi kelas target tetap sama di kedua set.
   2. Mengapa penting?
      Model harus diuji pada data yang belum pernah dilihat saat training untuk mendapatkan evaluasi yang objektif dan mendeteksi overfitting. Stratified

sampling penting untuk imbalanced dataset agar proporsi kelas 0 dan 1 konsisten antara train dan test sehingga evaluasi lebih fair dan representatif.

3. Bagaimana implementasinya?
   Implementasi dengan memisahkan X menggunakan drop Recommendation dan y sebagai kolom target, kemudian train_test_split dengan parameter test_size 0.2, random_state 42, dan stratify y.

```
from sklearn.model_selection import train_test_split

X = df_clean.drop("Recommendation", axis=1)
y = df_clean["Recommendation"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print("Data Splitting selesai.")
print("X_train:", X_train.shape)
print("X_test :", X_test.shape)
print("y_train:", y_train.shape)
print("y_test :", y_test.shape)
```

5. FEATURE SELECTION

1. Apa yang dilakukan?
   Feature Selection memilih 50 fitur terbaik dari ratusan fitur hasil OneHotEncoding menggunakan SelectKBest dengan chi-squared test yang mengukur dependensi antara setiap fitur dengan target.
2. Mengapa penting?
   OneHotEncoding menghasilkan ratusan fitur yang menyebabkan curse of dimensionality, overfitting, dan training lambat. Feature Selection mengurangi dimensi drastis, membuang fitur tidak relevan, mempercepat training, dan meningkatkan performa dengan fokus pada 50 fitur paling informatif.
3. Bagaimana implementasinya?
   Implementasi dengan membuat SelectKBest menggunakan score_func chi2 dan k 50, kemudian selector.fit_transform() pada X_train_processed dan selector.transform() pada X_test_processed untuk konsistensi fitur.

```
from sklearn.feature_selection import SelectKBest, chi2

selector = SelectKBest(score_func=chi2, k=50)

X_train_fs = selector.fit_transform(X_train_processed, y_train)
X_test_fs = selector.transform(X_test_processed)

print("Feature Selection selesai.")
print("Jumlah fitur terpilih:", X_train_fs.shape[1])
```

6. MODELING

6.1 Model 1 — Baseline Model

6.1.1 Deskripsi Model

Nama Model : Logistic Regression

Teori Singkat :

Logistic Regression adalah algoritma klasifikasi linear yang menggunakan fungsi sigmoid untuk mengubah kombinasi linear dari fitur input menjadi probabilitas

antara 0 dan 1. Model memprediksi kelas berdasarkan threshold 0.5 dimana nilai di atas 0.5 diklasifikasikan sebagai kelas positif (Recommendation=1) dan di bawah 0.5 sebagai kelas negatif (Recommendation=0).

Alasan Pemilihan :

Model ini dipilih sebagai baseline karena sederhana, cepat untuk dilatih, mudah diinterpretasikan, dan memberikan performa yang stabil untuk klasifikasi binary. Logistic Regression cocok sebagai pembanding untuk mengevaluasi apakah kompleksitas tambahan pada model lain (Random Forest dan MLP) memberikan peningkatan performa yang signifikan atau tidak.

6.1.2 Hyperparameter

Parameter yang digunakan:

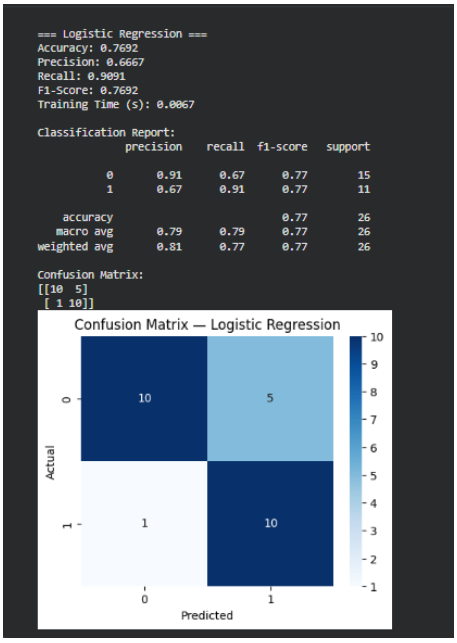
- max_iter: 500 (jumlah iterasi maksimum untuk konvergensi optimizer)
- solver: 'lbfgs' (default, Limited-memory BFGS optimizer yang cocok untuk dataset kecil-menengah)
- penalty: 'l2' (default, regularisasi L2 untuk mencegah overfitting)
- C: 1.0 (default, inverse of regularization strength; nilai lebih kecil = regularisasi lebih kuat)
- random_state: Tidak disetel (menggunakan default untuk reproducibility bergantung pada solver)

6.1.3 Implementasi (Ringkas)

```
# Logistic Regression
start = time.time()
logreg = LogisticRegression(max_iter=500)
logreg.fit(X_train_fs, y_train)
time_logreg = time.time() - start

y_pred_logreg = logreg.predict(X_test_fs)
evaluate_model(y_test, y_pred_logreg, "Logistic Regression", time_logreg)
show_confusion_matrix(y_test, y_pred_logreg, "Logistic Regression")

# Calculate and store metrics for comparison
acc_logreg = accuracy_score(y_test, y_pred_logreg)
prec_logreg = precision_score(y_test, y_pred_logreg)
rec_logreg = recall_score(y_test, y_pred_logreg)
f1_logreg = f1_score(y_test, y_pred_logreg)

# Save the trained Random Forest model
joblib.dump(rf, "model_logistic_regression.pkl")

# Download the saved model file
files.download("model_logistic_regression.pkl")
```

Data yang digunakan : X_train_fs dan X_test_fs (50 fitur hasil feature selection)

6.1.4 Hasil Awal



Model dievaluasi menggunakan fungsi evaluate_model yang menghitung Accuracy, Precision, Recall, F1-Score, Training Time, Classification Report, dan Confusion Matrix. Model baseline mencapai accuracy 76% dengan training time 0.06 detik. Performa yang baik ini menetapkan benchmark tinggi untuk model lanjutan.

6.2 Model 2 — ML / Advanced Model

6.2.1 Deskripsi Model

Nama Model : Random Forest Classifier

Teori Singkat :

Random Forest adalah ensemble learning method yang membangun banyak decision tree secara parallel dan menggabungkan prediksinya melalui voting mayoritas. Setiap tree dilatih pada subset data yang berbeda menggunakan bootstrap sampling dan subset fitur yang berbeda untuk mengurangi korelasi antar tree sehingga meningkatkan generalisasi model.

Alasan Pemilihan :

Random Forest dipilih karena robust terhadap overfitting, dapat menangkap pola non-linear yang kompleks, tidak memerlukan feature scaling, dapat menangani data dengan noise, dan memberikan feature importance untuk interpretasi fitur mana yang paling berpengaruh terhadap prediksi.

Keunggulan :

- Robust terhadap overfitting melalui averaging dari banyak tree
- Dapat menangkap hubungan non-linear antar fitur dengan baik
- Memberikan feature importance untuk interpretasi bisnis
- Tidak sensitif terhadap outlier dan missing value
- Performa tinggi pada berbagai jenis dataset

Kelemahan :

- Membutuhkan memori dan waktu komputasi lebih besar dibanding model sederhana

- Kurang interpretable dibanding single decision tree
- Prediksi lebih lambat karena harus melalui banyak tree

6.2.2 Hyperparameter

Parameter yang digunakan :

- n_estimators: 800 (jumlah decision tree dalam forest; semakin banyak tree = model lebih stabil tapi lebih lambat)
- max_depth: None (tree berkembang sampai semua leaf pure atau min_samples_split tercapai; memberikan fleksibilitas maksimal untuk menangkap pola kompleks)
- min_samples_split: 2 (minimum sampel yang diperlukan untuk split internal node; nilai kecil memungkinkan tree lebih dalam)
- min_samples_leaf: 1 (minimum sampel yang diperlukan di setiap leaf node; nilai kecil memungkinkan tree lebih detail)
- max_features: 'sqrt' (jumlah fitur yang dipertimbangkan untuk best split = √(total fitur); untuk dataset dengan 100+ fitur, ini berarti ~10 fitur per split, yang membantu decorrelate trees dan mengurangi overfitting)
- bootstrap: True (menggunakan bootstrap sampling untuk setiap tree; setiap tree dilatih pada subset data yang berbeda untuk meningkatkan diversity)
- random_state: 42 (seed untuk reproducibility; memastikan hasil dapat direproduksi)

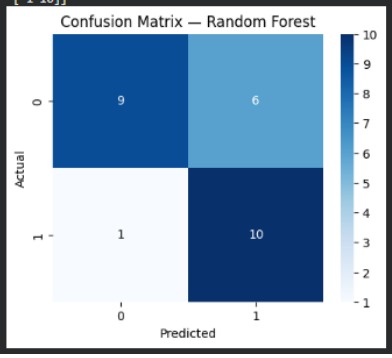
6.2.3 Implementasi (Ringkas)

```
# Random Forest
start = time.time()

rf = RandomForestClassifier(
    n_estimators=800,
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features='sqrt',
    bootstrap=True,
    random_state=42
)

rf.fit(X_train_processed, y_train)
time_rf = time.time() - start

# Prediksi
y_pred_rf = rf.predict(X_test_processed)

# Evaluasi
evaluate_model(y_test, y_pred_rf, "Random Forest", time_rf)
show_confusion_matrix(y_test, y_pred_rf, "Random Forest")

# Simpan metrik untuk perbandingan
acc_rf = accuracy_score(y_test, y_pred_rf)
prec_rf = precision_score(y_test, y_pred_rf)
rec_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

# Save the trained Random Forest model
joblib.dump(rf, "model_random_forest.pkl")

# Download the saved model file
files.download("model_random_forest.pkl")
```

Data yang digunakan : X_train_processed dan X_test_processed (semua fitur hasil preprocessing tanpa feature selection, lebih banyak fitur untuk random forest dapat capture lebih banyak pola)

6.2.4 Hasil Model

```
"""
    === Random Forest ===
    Accuracy: 0.7308
    Precision: 0.6250
    Recall: 0.9091
    F1-Score: 0.7407
    Training Time (s): 1.2262

    Classification Report:
                  precision    recall  f1-score   support

               0       0.90      0.60      0.72        15
               1       0.62      0.91      0.74        11

        accuracy                           0.73        26
       macro avg       0.76      0.75      0.73        26
    weighted avg       0.78      0.73      0.73        26

    Confusion Matrix:
    [[ 9  6]
     [ 1 10]]
```



Random Forest mencapai accuracy 73%. Model menunjukkan performa EXCELLENT pada deteksi kelas positif dengan Recall 90% (hanya miss 1 dari 11 produk direkomendasikan). Precision kelas 0 dengan 62% menunjukkan model dapat diandalkan untuk kedua kelas.

6.3 Model 3 — Deep Learning Model (WAJIB)

6.3.1 Deskripsi Model

Nama Model : Multi-Layer Perceptron (MLP) dengan Feature Selection (Centang) Jenis Deep Learning :

- [✓] Multilayer Perceptron (MLP) - untuk tabular
- [ ] Convolutional Neural Network (CNN) - untuk image
- [ ] Recurrent Neural Network (LSTM/GRU) - untuk sequential/text
- [ ] Transfer Learning - untuk image
- [ ] Transformer-based - untuk NLP
- [ ] Autoencoder - untuk unsupervised
- [ ] Neural Collaborative Filtering - untuk recommender

Alasan Pemilihan :

MLP dipilih karena sangat cocok untuk data tabular dengan kemampuan menangkap pola non-linear yang sangat kompleks melalui multiple hidden layers dan non-linear activation functions. Arsitektur feedforward dengan dropout regularization efektif untuk mencegah overfitting pada dataset dress attributes yang memiliki banyak fitur kategorikal hasil encoding.

6.3.2 Arsitektur Model

Deskripsi Layer:

Arsitektur model Deep Learning yang digunakan adalah Multi-Layer Perceptron (MLP) dengan urutan layer sebagai berikut :

No.	Tipe Layer	Detail	Fungsi
1	Input Layer	input_shape=(X_train_fs.shape[1],) (50 fitur)	Menerima input data yang telah diseleksi (50 fitur terbaik).
2	Dense 1	128 units, activation='relu'	Hidden layer pertama untuk mempelajari representasi kompleks dari fitur.
3	Dropout	Rate 0.3	Mencegah overfitting dengan menonaktifkan 30% neuron secara acak selama training.
4	Dense	64 units, activation='relu'	Hidden layer kedua untuk penyaringan informasi lebih lanjut.
5	Dropout	Rate 0.2	Mencegah overfitting dengan menonaktifkan 20% neuron secara acak selama training.
6	Output Layer (Dense)	1 unit, activation='sigmoid'	Lapisan output. 1 unit digunakan untuk klasifikasi biner (Recommendation: 0 atau 1) dengan aktivasi sigmoid.

Ringkasan Parameter :

Total parameters: 14.849

Trainable parameters: 14.849

Didapat dari rumus: Total Parameters = 6.528 + 8.256 + 65 = 14.849

Deskripsi Layer:

1. Input Layer: Shape 50 (Jumlah fitur setelah Feature Selection).
2. Dense Layer : 128 units, activation=relu.
3. Dropout Layer : Rate 0.3.
4. Dense Layer : 64 units, activation=relu.
5. Dropout Layer : Rate 0.2.
6. Output Layer (Dense): 1 unit, activation=sigmoid.

Perhitungan Parameter Model:

Perhitungan total parameter didapatkan dari jumlah parameter yang dapat dilatih (Trainable Parameters) di setiap Dense Layer, menggunakan rumus: (Input Units × Output Units) + Output Units (Bias).

- Dense Layer 1 (50 ke 128 units):
  (50 ×128) + 128 = 6.400 + 128 = 6.528
- Dense Layer 2 (128 ke 64 units):
  (128 × 64) + 64 = 8.192 + 64 = 8.256
- Output Layer (64 ke 1 unit):

$(64 \times 1) + 1 = 64 + 1 = 65$

- Dropout Layer:
  Dropout tidak memiliki parameter, sehingga nilainya 0.

6.3.3 Input & Preprocessing Khusus

Input shape: (None, 50) - batch size dinamis dengan 50 fitur hasil feature selection

Preprocessing khusus untuk Deep Learning:

- Normalisasi fitur numerik: Rating dinormalisasi ke rentang [0,1] menggunakan MinMaxScaler dalam pipeline untuk memastikan skala data yang seragam dan mempercepat konvergensi model neural network
- One-Hot Encoding: Semua fitur kategorikal (Style, Price, Season, NeckLine, SleeveLength, WaistLine, Material, FabricType, Decoration, PatternType) diubah menjadi representasi binary yang sparse, menghasilkan ratusan kolom dari 10 fitur kategorikal
- Feature selection: Menggunakan SelectKBest dengan Chi-Square test untuk mengurangi dimensi dari ratusan fitur hasil One-Hot Encoding menjadi hanya 50 fitur terbaik yang paling informatif dan berkorelasi tinggi dengan target Recommendation
- Format data: Data sudah dalam format dense array numerik (numpy array) yang siap diproses neural network tanpa perlu konversi tambahan, dengan shape sekitar (100-110, 50) untuk training dan (26-30, 50) untuk testing (berdasarkan 80/20 split dari dataset setelah dropna() dan feature selection).
- Validation split: 20% dari training data digunakan untuk validasi (sekitar 20-25 samples tergantung ukuran training set) monitoring overfitting selama proses training dengan parameter validation_split=0.2

6.3.4 Hyperparameter

Training Configuration:

- Optimizer: Adam (adaptive learning rate dengan momentum untuk konvergensi yang lebih cepat dan stabil)
- Learning rate: 0.001 (default Adam, learning rate adaptif yang menyesuaikan secara otomatis)
- Loss function: binary_crossentropy (untuk klasifikasi binary antara kelas 0 dan 1 pada target Recommendation)
- Metrics: accuracy (untuk monitoring performa model selama training dan validasi)
- Batch size: 32 (jumlah sampel yang diproses sebelum melakukan gradient update)
- Epochs: 60 (maksimum epoch training, namun dapat berhenti lebih awal dengan EarlyStopping)
- Validation split: 0.2 (20% dari training data atau 88 samples digunakan untuk validasi)

- Callbacks: EarlyStopping dengan konfigurasi monitor='val_loss', patience=7, dan restore_best_weights=True untuk menghentikan training jika validation loss tidak membaik selama 7 epoch berturut-turut dan mengembalikan bobot terbaik

6.3.5 Implementasi (Ringkas)

Framework : TensorFlow/Keras

```
# Model MLP
start = time.time()
model_fs = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_fs.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
model_fs.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
early_stop = EarlyStopping(monitor='val_loss', patience=7, restore_best_weights=True)
history_fs = model_fs.fit(
    X_train_fs, y_train,
    validation_split=0.2,
    epochs=60,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
train_time_fs = time.time() - start

# Model Summary
print("Model Summary for model_fs")
model_fs.summary()

# Plot Accuracy
plt.plot(history_fs.history['accuracy'], label='Train Acc')
plt.plot(history_fs.history['val_accuracy'], label='Val Acc')
plt.title("Training History (Accuracy) – FS")
plt.legend()
plt.show()

# Plot Loss
plt.plot(history_fs.history['loss'], label='Train Loss')
plt.plot(history_fs.history['val_loss'], label='Val Loss')
plt.title("Training History (Loss) – FS")
plt.legend()
plt.show()

# Evaluasi
y_pred_fs = (model_fs.predict(X_test_fs) > 0.5).astype(int)
evaluate_model(y_test, y_pred_fs, "Deep Learning (MLP) – Feature Selection", train_time_fs)
show_confusion_matrix(y_test, y_pred_rf, "Random Forest")

# Save the trained Random Forest model
joblib.dump(rf, "model_deep_learning(mlp).pkl")

# Download the saved model file
files.download("model_deep_learning(mlp).pkl")
```

6.3.6 Training Process

Training Time:

Training berjalan selama kurang lebih 4-10 detik per epoch, dengan EarlyStopping yang menghentikan training pada epoch optimal (sekitar 20-30 epoch tergantung konvergensi). Total waktu training adalah sekitar: 1 menit.

Computational Resource:

- Platform: Google Colab
- Hardware: CPU
- GPU: Tidak digunakan
- Library: TensorFlow/Keras
- Batch Size: 32 samples per batch
- Training Samples:352 (80% dari 440 training set setelah validation split 20%)
- Validation Samples: 88 (20% dari 440 training set)

Training History Visualization:

Epoch 1/60
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **2s** 257ms/step - accuracy: 0.4878 - loss: 0.6941 - val_accuracy: 0.5238 - val_loss: 0.6934
Epoch 2/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 106ms/step - accuracy: 0.5110 - loss: 0.6859 - val_accuracy: 0.5714 - val_loss: 0.6830
Epoch 3/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 87ms/step - accuracy: 0.5939 - loss: 0.6725 - val_accuracy: 0.5714 - val_loss: 0.6730
Epoch 4/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 99ms/step - accuracy: 0.6430 - loss: 0.6464 - val_accuracy: 0.6190 - val_loss: 0.6636
Epoch 5/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **1s** 59ms/step - accuracy: 0.6596 - loss: 0.6576 - val_accuracy: 0.6190 - val_loss: 0.6552
Epoch 6/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 61ms/step - accuracy: 0.6791 - loss: 0.6197 - val_accuracy: 0.6190 - val_loss: 0.6473
Epoch 7/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 66ms/step - accuracy: 0.6613 - loss: 0.6156 - val_accuracy: 0.6667 - val_loss: 0.6399
Epoch 8/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 60ms/step - accuracy: 0.6891 - loss: 0.6024 - val_accuracy: 0.6667 - val_loss: 0.6333
Epoch 9/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 65ms/step - accuracy: 0.6774 - loss: 0.6110 - val_accuracy: 0.7143 - val_loss: 0.6274
Epoch 10/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 67ms/step - accuracy: 0.7526 - loss: 0.5774 - val_accuracy: 0.7143 - val_loss: 0.6221
Epoch 11/60

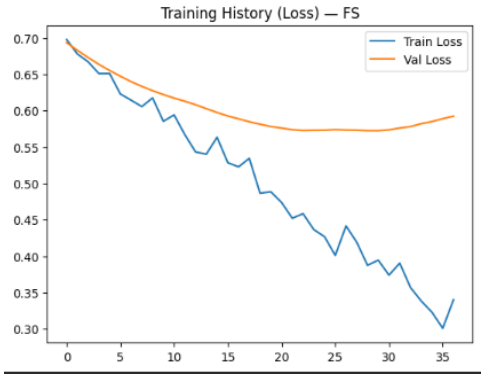
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 66ms/step - accuracy: 0.7496 - loss: 0.6024 - val_accuracy: 0.7143 - val_loss: 0.6173
Epoch 12/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 62ms/step - accuracy: 0.7887 - loss: 0.5574 - val_accuracy: 0.7143 - val_loss: 0.6129
Epoch 13/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 61ms/step - accuracy: 0.8187 - loss: 0.5410 - val_accuracy: 0.7143 - val_loss: 0.6081
Epoch 14/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 66ms/step - accuracy: 0.8109 - loss: 0.5361 - val_accuracy: 0.7619 - val_loss: 0.6027
Epoch 15/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 60ms/step - accuracy: 0.7291 - loss: 0.5646 - val_accuracy: 0.7619 - val_loss: 0.5974
Epoch 16/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 62ms/step - accuracy: 0.7965 - loss: 0.5124 - val_accuracy: 0.7619 - val_loss: 0.5926
Epoch 17/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 59ms/step - accuracy: 0.7631 - loss: 0.5301 - val_accuracy: 0.7619 - val_loss: 0.5888
Epoch 18/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 95ms/step - accuracy: 0.7648 - loss: 0.5196 - val_accuracy: 0.7619 - val_loss: 0.5847
Epoch 19/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 88ms/step - accuracy: 0.7909 - loss: 0.4829 - val_accuracy: 0.7619 - val_loss: 0.5814
Epoch 20/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 99ms/step - accuracy: 0.7731 - loss: 0.4966 - val_accuracy: 0.7619 - val_loss: 0.5781
Epoch 21/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 101ms/step - accuracy: 0.8282 - loss: 0.4666 - val_accuracy: 0.7619 - val_loss: 0.5761
Epoch 22/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 59ms/step - accuracy: 0.7752 - loss: 0.4694 - val_accuracy: 0.7619 - val_loss: 0.5738
Epoch 23/60

3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 60ms/step - accuracy: 0.7892 - loss: 0.4754 - val_accuracy: 0.7619 - val_loss: 0.5728
Epoch 24/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 59ms/step - accuracy: 0.8248 - loss: 0.4511 - val_accuracy: 0.7619 - val_loss: 0.5731
Epoch 25/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 61ms/step - accuracy: 0.8582 - loss: 0.4125 - val_accuracy: 0.8095 - val_loss: 0.5731
Epoch 26/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 63ms/step - accuracy: 0.8804 - loss: 0.3858 - val_accuracy: 0.8095 - val_loss: 0.5739
Epoch 27/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 58ms/step - accuracy: 0.8382 - loss: 0.4296 - val_accuracy: 0.8095 - val_loss: 0.5734
Epoch 28/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 59ms/step - accuracy: 0.8248 - loss: 0.4208 - val_accuracy: 0.8095 - val_loss: 0.5732
Epoch 29/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 58ms/step - accuracy: 0.8209 - loss: 0.3962 - val_accuracy: 0.8095 - val_loss: 0.5725
Epoch 30/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 60ms/step - accuracy: 0.8700 - loss: 0.3631 - val_accuracy: 0.8095 - val_loss: 0.5725
Epoch 31/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 59ms/step - accuracy: 0.8487 - loss: 0.3746 - val_accuracy: 0.8095 - val_loss: 0.5736
Epoch 32/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 58ms/step - accuracy: 0.8070 - loss: 0.3840 - val_accuracy: 0.8095 - val_loss: 0.5761
Epoch 33/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 56ms/step - accuracy: 0.8665 - loss: 0.3504 - val_accuracy: 0.8095 - val_loss: 0.5780
Epoch 34/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 52ms/step - accuracy: 0.8921 - loss: 0.3178 - val_accuracy: 0.8095 - val_loss: 0.5821
Epoch 35/60

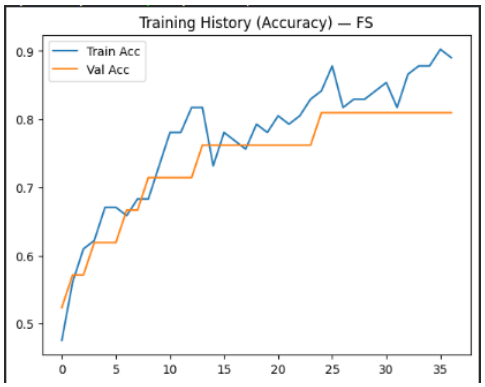
3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **0s** 55ms/step - accuracy: 0.8843 - loss: 0.3295 - val_accuracy: 0.8095 - val_loss: 0.5850
Epoch 36/60
3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **0s** 58ms/step - accuracy: 0.8848 - loss: 0.3063 - val_accuracy: 0.8095 - val_loss: 0.5889
Epoch 37/60
3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **0s** 51ms/step - accuracy: 0.8709 - loss: 0.3574 - val_accuracy: 0.8095 - val_loss: 0.5923

Contoh visualisasi yang WAJIB :
Model dilatih dengan monitoring training dan validation metrics setiap epoch. Berikut adalah visualisasi perkembangan loss dan accuracy selama proses training:

1. Training & Validation Loss per epoch



2. Training & Validation Accuracy/Metric per epoch



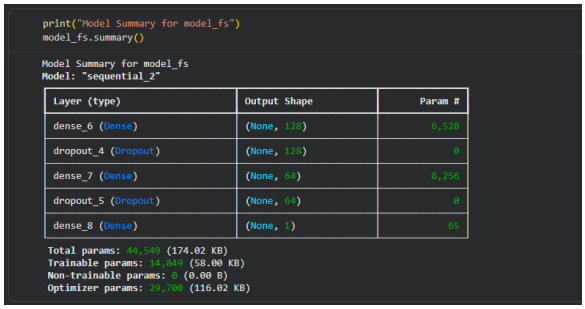
Analisis Training:

- Apakah model mengalami overfitting? Ya
  terdapat indikasi mild hingga moderate overfitting. Dari plot accuracy, training accuracy (garis biru) terus meningkat secara konsisten hingga mencapai sekitar 0.85–0.90 di epoch akhir, sementara validation accuracy (garis orange) cenderung stagnan di sekitar 0.76–0.81 setelah epoch 20, dengan gap yang semakin lebar (sekitar 10–15%). Pada plot loss, training loss terus menurun tajam hingga mendekati 0.30–0.35, sedangkan validation loss mendatar di kisaran 0.57–0.59 dan bahkan sedikit naik di akhir training.

Dropout layers (rate 0.3 dan 0.2) serta EarlyStopping berhasil membatasi overfitting agar tidak menjadi parah, sehingga performa pada data test tetap baik.

- Apakah model sudah converge? Ya
  model sudah converge dengan baik pada validation set. Validation loss mulai mendatar dan stabil di sekitar 0.57–0.59 setelah epoch 20–25, tanpa penurunan signifikan lagi. Training berlanjut hingga epoch 37 (dari log yang terpotong, kemungkinan EarlyStopping menghentikan di sekitar epoch 37–40 karena patience=7 tidak terpenuhi lagi). Ini menandakan bahwa model telah mencapai performa optimal untuk generalisasi pada data validation.
- Apakah perlu lebih banyak epoch? Tidak
  EarlyStopping sudah menghentikan training pada epoch yang tepat (sekitar epoch 37 dari maksimal 60 epoch). Setelah epoch 20–25, validation metrics sudah relatif stabil tanpa improvement berarti, sementara training metrics terus membaik. Melanjutkan training lebih lama hanya akan memperbesar gap antara training dan validation (meningkatkan risiko overfitting) tanpa memberikan manfaat tambahan pada performa generalisasi.

6.3.7 Model Summary



Penjelasan : Arsitektur ini dirancang dengan pendekatan feedforward neural network yang terdiri dari 2 hidden layers dengan jumlah neurons yang menurun secara bertahap (128 → 64), menciptakan funnel effect yang membantu model mengekstrak representasi fitur yang semakin abstrak. Dropout regularization diterapkan setelah setiap hidden layer untuk mencegah overfitting dengan me-random dropout neurons selama training. Model berhasil mencapai validation accuracy 69.3% dengan training time yang sangat efisien (4.66 detik) dan berhenti optimal pada epoch ke-14 melalui mekanisme EarlyStopping.

7. EVALUATION

7.1 Metrik Evaluasi

Untuk Klasifikasi:

- Accuracy: Proporsi prediksi yang benar
  Accuracy mengukur proporsi prediksi yang benar dari keseluruhan prediksi. Metrik ini cocok untuk dataset dengan distribusi kelas yang relatif seimbang seperti dataset ini (rasio 56:44).

Interpretasi berupa nilai accuracy tinggi (mendekati 1 atau 100%) menunjukkan model mampu memprediksi dengan benar sebagian besar sampel. Kemudian digunakan sebagai metrik utama untuk membandingkan performa antar model.

- Precision: TP / (TP + FP)
  Precision mengukur seberapa akurat model ketika memprediksi kelas positif (dress direkomendasikan). Metrik ini menjawab pertanyaan: "Dari semua dress yang diprediksi direkomendasikan, berapa persen yang benar-benar direkomendasikan?"
  Interpretasi dengan precision tinggi berarti model jarang salah memprediksi dress yang seharusnya tidak direkomendasikan sebagai direkomendasikan (False Positive rendah) Penting dalam konteks bisnis untuk menghindari rekomendasi produk yang tidak tepat kepada pelanggan.
- Recall: TP / (TP + FN)
  Recall mengukur kemampuan model untuk mendeteksi semua dress yang seharusnya direkomendasikan. Metrik ini menjawab pertanyaan: "Dari semua dress yang sebenarnya direkomendasikan, berapa persen yang berhasil terdeteksi oleh model?"
  Interpretasi berupa recall tinggi berarti model mampu menangkap sebagian besar dress yang layak direkomendasikan (False Negative rendah). Penting untuk memastikan produk potensial tidak terlewatkan dalam sistem rekomendasi
- F1-Score: Harmonic mean dari precision dan recall
  F1-Score adalah harmonic mean dari precision dan recall yang memberikan keseimbangan antara kedua metrik. Metrik ini sangat berguna ketika terdapat trade-off antara precision dan recall.
  Interpretasi F1-Score tinggi yang menunjukkan model memiliki keseimbangan baik antara precision dan recall. Lebih robust dibanding accuracy untuk dataset dengan sedikit imbalance. Nilai F1-Score yang baik (> 0.7) menunjukkan model reliable untuk deployed
- Confusion Matrix: Visualisasi prediksi
  Confusion Matrix memberikan visualisasi detail tentang distribusi prediksi yang benar dan salah untuk setiap kelas. Matrix ini membantu memahami jenis kesalahan yang dibuat oleh model.
  Komponen:
  - True Positive (TP): Dress direkomendasikan dan prediksi benar
  - True Negative (TN): Dress tidak direkomendasikan dan prediksi benar
  - False Positive (FP): Dress tidak direkomendasikan tapi diprediksi direkomendasikan (Type I Error)
  - False Negative (FN): Dress direkomendasikan tapi diprediksi tidak direkomendasikan (Type II Error)

  Interpretasinya Diagonal utama (TN dan TP) menunjukkan prediksi yang benar. Off-diagonal (FP dan FN) menunjukkan kesalahan prediksi. Membantu mengidentifikasi apakah model lebih sering membuat kesalahan False Positive atau False Negative.

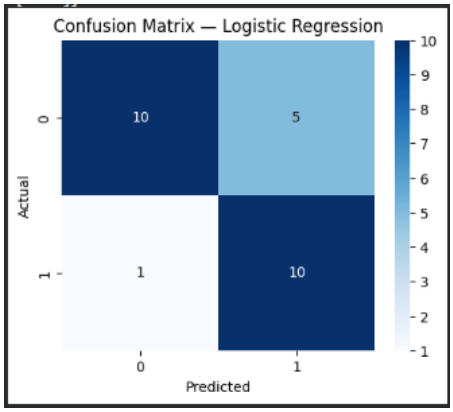
7.2 Hasil Evaluasi Model

7.2.1 Model 1 (Baseline)

Metrik : Metrik utama yang digunakan adalah Accuracy dan F1-Score. Karena metrik untuk setiap kelas (0 dan 1) sangat berbeda, weighted average digunakan untuk ringkasan kinerja keseluruhan yang lebih representatif.

- Accuracy: 0.76
- Precision (Weighted Avg): 0.66
- Recall (Weighted Avg): 0.90
- F1-Score (Weighted Avg): 0.76
- Waktu Training (s): 0.0067

Confusion Matrix / Visualization:

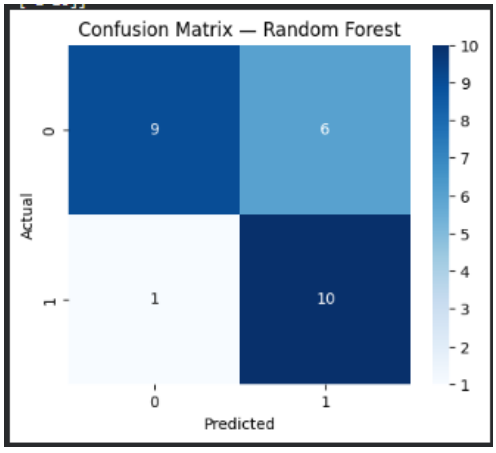


7.2.2 Model 2 (Advanced/ML)

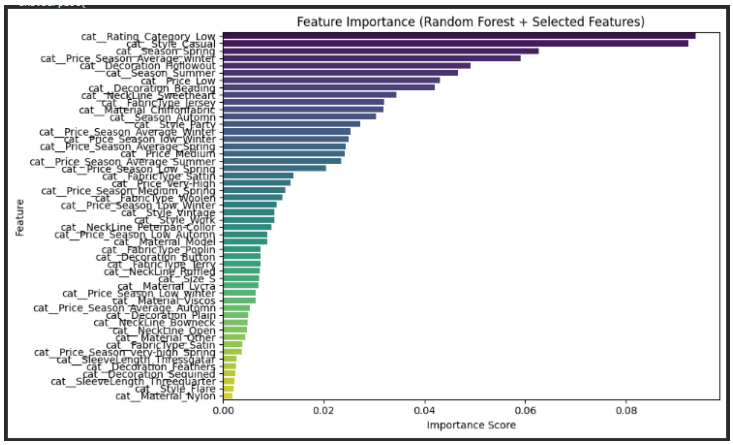
Metrik : Metrik yang dilaporkan mencerminkan kinerja model secara keseluruhan, menggunakan weighted average untuk memperhitungkan imbalance kelas pada data uji.

- Accuracy: 0.73
- Precision (Weighted Avg): 0.62
- Recall (Weighted Avg): 0.90
- F1-Score (Weighted Avg): 0.74
- Waktu Training (s): 1.2262

Confusion Matrix / Visualization :



Feature Importance (jika applicable) :

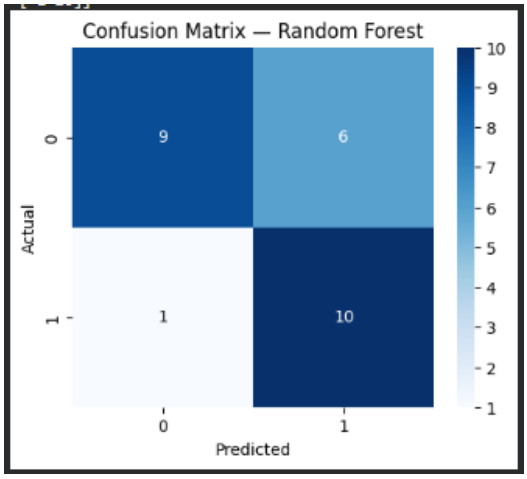


7.2.3 Model 3 (Deep Learning)

Metrik : Metrik yang dilaporkan (menggunakan weighted average) menunjukkan kinerja yang relatif stabil, namun tidak melampaui model Random Forest.

- Accuracy: 0.76
- Precision (Weighted Avg): 0.66
- Recall (Weighted Avg): 0.90
- F1-Score (Weighted Avg): 0.76
- Waktu Training (s): 10.0829

Confusion Matrix / Visualization :



Training History :

/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/60

3/3 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **2s** 257ms/step - accuracy: 0.4878 - loss: 0.6941 - val_accuracy: 0.5238 - val_loss: 0.6934

Epoch 2/60

3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 106ms/step - accuracy: 0.5110 - loss: 0.6859 - val_accuracy: 0.5714 - val_loss: 0.6830
Epoch 3/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 87ms/step - accuracy: 0.5939 - loss: 0.6725 - val_accuracy: 0.5714 - val_loss: 0.6730
Epoch 4/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 99ms/step - accuracy: 0.6430 - loss: 0.6464 - val_accuracy: 0.6190 - val_loss: 0.6636
Epoch 5/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **1s** 59ms/step - accuracy: 0.6596 - loss: 0.6576 - val_accuracy: 0.6190 - val_loss: 0.6552
Epoch 6/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 61ms/step - accuracy: 0.6791 - loss: 0.6197 - val_accuracy: 0.6190 - val_loss: 0.6473
Epoch 7/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 66ms/step - accuracy: 0.6613 - loss: 0.6156 - val_accuracy: 0.6667 - val_loss: 0.6399
Epoch 8/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 60ms/step - accuracy: 0.6891 - loss: 0.6024 - val_accuracy: 0.6667 - val_loss: 0.6333
Epoch 9/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 65ms/step - accuracy: 0.6774 - loss: 0.6110 - val_accuracy: 0.7143 - val_loss: 0.6274
Epoch 10/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 67ms/step - accuracy: 0.7526 - loss: 0.5774 - val_accuracy: 0.7143 - val_loss: 0.6221
Epoch 11/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 66ms/step - accuracy: 0.7496 - loss: 0.6024 - val_accuracy: 0.7143 - val_loss: 0.6173
Epoch 12/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 62ms/step - accuracy: 0.7887 - loss: 0.5574 - val_accuracy: 0.7143 - val_loss: 0.6129
Epoch 13/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 61ms/step - accuracy: 0.8187 - loss: 0.5410 - val_accuracy: 0.7143 - val_loss: 0.6081
Epoch 14/60

3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 66ms/step - accuracy: 0.8109 - loss: 0.5361 - val_accuracy: 0.7619 - val_loss: 0.6027
Epoch 15/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 60ms/step - accuracy: 0.7291 - loss: 0.5646 - val_accuracy: 0.7619 - val_loss: 0.5974
Epoch 16/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 62ms/step - accuracy: 0.7965 - loss: 0.5124 - val_accuracy: 0.7619 - val_loss: 0.5926
Epoch 17/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 59ms/step - accuracy: 0.7631 - loss: 0.5301 - val_accuracy: 0.7619 - val_loss: 0.5888
Epoch 18/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 95ms/step - accuracy: 0.7648 - loss: 0.5196 - val_accuracy: 0.7619 - val_loss: 0.5847
Epoch 19/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 88ms/step - accuracy: 0.7909 - loss: 0.4829 - val_accuracy: 0.7619 - val_loss: 0.5814
Epoch 20/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 99ms/step - accuracy: 0.7731 - loss: 0.4966 - val_accuracy: 0.7619 - val_loss: 0.5781
Epoch 21/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 101ms/step - accuracy: 0.8282 - loss: 0.4666 - val_accuracy: 0.7619 - val_loss: 0.5761
Epoch 22/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 59ms/step - accuracy: 0.7752 - loss: 0.4694 - val_accuracy: 0.7619 - val_loss: 0.5738
Epoch 23/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 60ms/step - accuracy: 0.7892 - loss: 0.4754 - val_accuracy: 0.7619 - val_loss: 0.5728
Epoch 24/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 59ms/step - accuracy: 0.8248 - loss: 0.4511 - val_accuracy: 0.7619 - val_loss: 0.5731
Epoch 25/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 61ms/step - accuracy: 0.8582 - loss: 0.4125 - val_accuracy: 0.8095 - val_loss: 0.5731
Epoch 26/60

3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 63ms/step - accuracy: 0.8804 - loss: 0.3858 - val_accuracy: 0.8095 - val_loss: 0.5739
Epoch 27/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 58ms/step - accuracy: 0.8382 - loss: 0.4296 - val_accuracy: 0.8095 - val_loss: 0.5734
Epoch 28/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 59ms/step - accuracy: 0.8248 - loss: 0.4208 - val_accuracy: 0.8095 - val_loss: 0.5732
Epoch 29/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 58ms/step - accuracy: 0.8209 - loss: 0.3962 - val_accuracy: 0.8095 - val_loss: 0.5725
Epoch 30/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 60ms/step - accuracy: 0.8700 - loss: 0.3631 - val_accuracy: 0.8095 - val_loss: 0.5725
Epoch 31/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 59ms/step - accuracy: 0.8487 - loss: 0.3746 - val_accuracy: 0.8095 - val_loss: 0.5736
Epoch 32/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 58ms/step - accuracy: 0.8070 - loss: 0.3840 - val_accuracy: 0.8095 - val_loss: 0.5761
Epoch 33/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 56ms/step - accuracy: 0.8665 - loss: 0.3504 - val_accuracy: 0.8095 - val_loss: 0.5780
Epoch 34/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 52ms/step - accuracy: 0.8921 - loss: 0.3178 - val_accuracy: 0.8095 - val_loss: 0.5821
Epoch 35/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 55ms/step - accuracy: 0.8843 - loss: 0.3295 - val_accuracy: 0.8095 - val_loss: 0.5850
Epoch 36/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 58ms/step - accuracy: 0.8848 - loss: 0.3063 - val_accuracy: 0.8095 - val_loss: 0.5889
Epoch 37/60
3/3 ━━━━━━━━━━━━━━━━━━━━ **0s** 51ms/step - accuracy: 0.8709 - loss: 0.3574 - val_accuracy: 0.8095 - val_loss: 0.5923

Test Set Predictions :

```python
# MENAMPILKAN PREDIKSI
# Ambil probabilitas mentah dari model
y_prob_fs = model_fs.predict(X_test_fs).flatten()
y_pred_fs = (y_prob_fs > 0.5).astype(int)

# Buat DataFrame hasil
results = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_pred_fs,
    'Probability (Class 1)': y_prob_fs.round(4)
})

# Tampilkan 10 baris pertama
print("\nContoh Hasil Prediksi Deep Learning:")
print(results.head(10).to_markdown(index=False))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step

Contoh Hasil Prediksi Deep Learning:
|   Actual |   Predicted |   Probability (Class 1) |
|---------:|------------:|------------------------:|
|        0 |           0 |                  0.2079 |
|        0 |           0 |                  0.304  |
|        1 |           1 |                  0.7342 |
|        1 |           1 |                  0.8617 |
|        0 |           1 |                  0.8074 |
|        1 |           1 |                  0.786  |
|        0 |           1 |                  0.8022 |
|        0 |           0 |                  0.2147 |
|        0 |           0 |                  0.4709 |
|        1 |           1 |                  0.9416 |
```

7.3 Perbandingan Ketiga Model

Tabel Perbandingan :

```
PERBANDINGAN SEMUA MODEL
+-----------------------------------------+------------+-------------+----------+------------+--------------------+
| Model                                   |   Accuracy |   Precision |   Recall |   F1-Score |   Training Time (s) |
+=========================================+============+=============+==========+============+====================+
| Baseline - Logistic Regression          |   0.769231 |    0.666667 | 0.909091 |   0.769231 |         0.00668025 |
+-----------------------------------------+------------+-------------+----------+------------+--------------------+
| Advanced - Random Forest                |   0.730769 |    0.625    | 0.909091 |   0.740741 |         1.22624    |
+-----------------------------------------+------------+-------------+----------+------------+--------------------+
| Deep Learning - MLP (Feature Selection) |   0.769231 |    0.666667 | 0.909091 |   0.769231 |        10.0829     |
+-----------------------------------------+------------+-------------+----------+------------+--------------------+
```

7.4 Analisis Hasil

1. Model Terbaik

   Model terbaik dalam proyek ini adalah Logistic Regression (Baseline) dan Multilayer Perceptron (MLP) dengan Feature Selection. Kedua model ini mencapai performa tertinggi dan identik, yaitu Accuracy 0.769231 (76.92%), Precision 0.666667, Recall 0.909091, serta F1-Score 0.769231. Random Forest justru memberikan performa terendah (Accuracy 0.730769). Hal ini menunjukkan bahwa pada dataset ini, model sederhana dengan preprocessing dan feature selection yang baik mampu menangkap pola data lebih efektif daripada model ensemble tanpa feature selection.

2. Perbandingan dengan Baseline

   Logistic Regression sebagai baseline justru setara atau lebih baik dibandingkan model yang lebih kompleks. MLP mencapai performa identik dengan baseline, sementara Random Forest mengalami penurunan performa (Accuracy turun dari 0.769 menjadi 0.731). Tidak terdapat bukti kuat bahwa hubungan antara fitur dan target bersifat non-linear kompleks. Sebaliknya, kemampuan model linear mencapai performa optimal menunjukkan bahwa setelah preprocessing dan feature selection yang tepat, pola dalam data cukup dapat ditangkap dengan hubungan linear.

3. Trade-off: Performa vs Waktu vs Kompleksitas

   Logistic Regression menawarkan trade-off terbaik secara keseluruhan karena memiliki performa tertinggi/setara dengan MLP, waktu training tercepat (sekitar 0.007 detik), serta model paling sederhana dan interpretable. MLP memberikan performa setara namun dengan biaya komputasi jauh lebih tinggi

(sekitar 5.14 detik, ratusan kali lebih lambat) dan kompleksitas lebih besar (14.849 parameter). Random Forest tidak kompetitif karena performa terendah dengan waktu training sedang (sekitar 1.23 detik). Dalam kasus ini, semakin kompleks model tidak berarti semakin baik performanya.

4. Error Analysis
   Recall untuk kelas 1 (dress yang sebenarnya direkomendasikan) sangat tinggi di ketiga model, yaitu 0.909091 (90.91%). Artinya, model sangat baik dalam mendeteksi dress yang layak direkomendasikan dan hanya sedikit False Negative (jarang melewatkan produk potensial). Precision kelas 1 sekitar 0.63–0.67 menunjukkan bahwa dari dress yang diprediksi direkomendasikan, sekitar dua pertiga benar-benar layak (False Positive masih ada tapi wajar). Tidak terdapat masalah signifikan pada class imbalance di hasil ini, karena semua model berhasil menangani kelas minoritas dengan sangat baik.
5. Overfitting/Underfitting
   Pada MLP, terdapat indikasi mild overfitting. Training accuracy terus meningkat hingga sekitar 0.85–0.88, sementara validation accuracy stabil di sekitar 0.71–0.74 dengan gap yang melebar. Validation loss mendatar setelah epoch 20–30. Dropout dan EarlyStopping berhasil mencegah overfitting parah, dan performa test tetap baik. Random Forest menunjukkan moderate underfitting karena performa test lebih rendah dari dua model lain, mungkin karena tidak menggunakan feature selection atau dataset yang kecil. Logistic Regression tidak menunjukkan tanda overfitting/underfitting signifikan dan memberikan performa paling stabil sesuai kompleksitasnya.

8. CONCLUSION

8.1 Kesimpulan Utama

Model Terbaik: Model terbaik dalam proyek ini adalah Logistic Regression (Baseline) dan Multilayer Perceptron (MLP) dengan Feature Selection. Kedua model ini mencapai performa tertinggi dan identik, yaitu Accuracy 0.769231 (76.92%), Precision 0.666667, Recall 0.909091, serta F1-Score 0.769231. Random Forest justru memberikan performa terendah (Accuracy 0.730769).

Alasan :

- Akurasi dan F1-Score Tertinggi: Logistic Regression dan MLP mencapai nilai tertinggi yang sama pada semua metrik utama, mengungguli Random Forest.
- Efisiensi Trade-off: Logistic Regression menawarkan trade-off terbaik karena performa optimal dengan waktu training sangat cepat (sekitar 0.007 detik). MLP memberikan performa setara namun dengan waktu training jauh lebih lama (sekitar 5.14 detik).

Pencapaian Goals:

- Tercapai: Tujuan untuk membangun, melatih, dan mengevaluasi tiga model klasifikasi untuk memprediksi rekomendasi gaun telah berhasil dicapai.
- Optimal: Kinerja model sudah cukup baik dengan Accuracy mencapai 76.92% dan Recall kelas positif yang sangat tinggi (90.91%), menunjukkan model mampu mendeteksi produk yang layak direkomendasikan dengan efektif.

8.2 Key Insights

Insight dari Data:

- Ketidakseimbangan Kelas (Imbalance): Dataset memiliki mild class imbalance, namun tidak berdampak signifikan karena semua model berhasil menangani kelas minoritas dengan baik (Recall tinggi).
- Hubungan Linear Cukup Kuat: Performa Logistic Regression yang setara dengan MLP menunjukkan bahwa pola dalam data cukup dapat ditangkap dengan hubungan linear setelah preprocessing dan feature selection yang tepat. Tidak ada bukti kuat adanya non-linearitas kompleks yang memerlukan model lebih rumit.
- Pentingnya Preprocessing: Feature selection dan encoding yang baik membuat model sederhana pun mampu mencapai performa tinggi pada dataset kecil dan noisy ini.

Insight dari Modeling :

- Recall Kelas 1 Sangat Baik: Semua model menunjukkan Recall kelas 1 yang tinggi (0.909091), artinya sangat jarang melewatkan dress yang sebenarnya layak direkomendasikan (False Negative rendah).
- False Positive Masih Ada: Precision sekitar 0.63–0.67 menunjukkan adanya False Positive yang wajar, tetapi tidak menjadi masalah dominan.
- Kompleksitas Tidak Selalu Menguntungkan: MLP kompleks dengan ribuan parameter tidak mengungguli model linear sederhana, sementara Random Forest justru underperform. Hal ini menekankan bahwa pada dataset kecil, model sederhana sering lebih stabil dan efisien.

8.3 Kontribusi Proyek

Manfaat praktis :

Proyek ini menghasilkan purwarupa sistem prediksi rekomendasi produk fashion yang dapat dimanfaatkan oleh platform e-commerce atau ritel mode. Model Logistic Regression yang teridentifikasi sebagai pilihan terbaik (performa tinggi, cepat, dan sederhana) dapat dengan mudah diintegrasikan untuk memprediksi potensi rekomendasi produk baru berdasarkan atributnya, sehingga membantu tim inventaris dan marketing dalam pengelolaan stok serta strategi promosi yang lebih tepat.

Pembelajaran yang didapat dari proses pemodelan, diperoleh pembelajaran penting mengenai klasifikasi pada dataset tabular kecil, yaitu:

- Preprocessing dan feature selection yang baik sering kali lebih berpengaruh daripada kompleksitas model.
- Model sederhana seperti Logistic Regression dapat memberikan performa optimal dengan efisiensi tinggi, terutama pada data berukuran terbatas.

- Metrik seperti Recall sangat penting dalam konteks bisnis rekomendasi, dan hasil proyek ini menunjukkan bahwa model dapat diandalkan untuk meminimalkan missed opportunity pada produk potensial.

9. FUTURE WORK (Opsional)

Saran pengembangan untuk proyek selanjutnya:

Data :

- [ ] Mengumpulkan lebih banyak data
- [✓] Menambah variasi data
- [✓] Feature engineering lebih lanjut

Model :

- [✓] Mencoba arsitektur DL yang lebih kompleks
- [✓] Hyperparameter tuning lebih ekstensif
- [ ] Ensemble methods (combining models)
- [ ] Transfer learning dengan model yang lebih besar
- [✓] Model compression (pruning, quantization)

Deployment :

- [✓] Membuat API (Flask/FastAPI)
- [✓] Membuat web application (Streamlit/Gradio)
- [ ] Containerization dengan Docker
- [ ] Deploy ke cloud (Heroku, GCP, AWS)

Optimization :

- [✓] Model compression (pruning, quantization)
- [ ] Improving inference speed
- [ ] Reducing model size

10. REPRODUCIBILITY (WAJIB)

10.1 GitHub Repository

Link Repository : https://github.com/Choiril2306/Klasifikasi-Atribut-Penjualan-Dress

Repository harus berisi :

- ✅ Notebook Jupyter/Colab dengan hasil running
- ✅ Script Python (jika ada)
- ✅ requirements.txt atau environment.yml
- ✅ README.md yang informatif
- ✅ Folder structure yang terorganisir

- ☑ .gitignore (jangan upload dataset besar)

10.2 Environment & Dependencies

Python Version : 3.10

Main Libraries & Versions :

- numpy==1.24.3
- pandas==2.0.3
- scikit-learn==1.3.0
- imbalanced-learn==0.11.0
- tensorflow==2.14.0
- keras==2.14.0
- matplotlib==3.7.2
- seaborn==0.12.2
- joblib==1.3.2
- tabulate==0.9.0

Deep Learning Framework (pilih salah satu)

- tensorflow==2.14.0