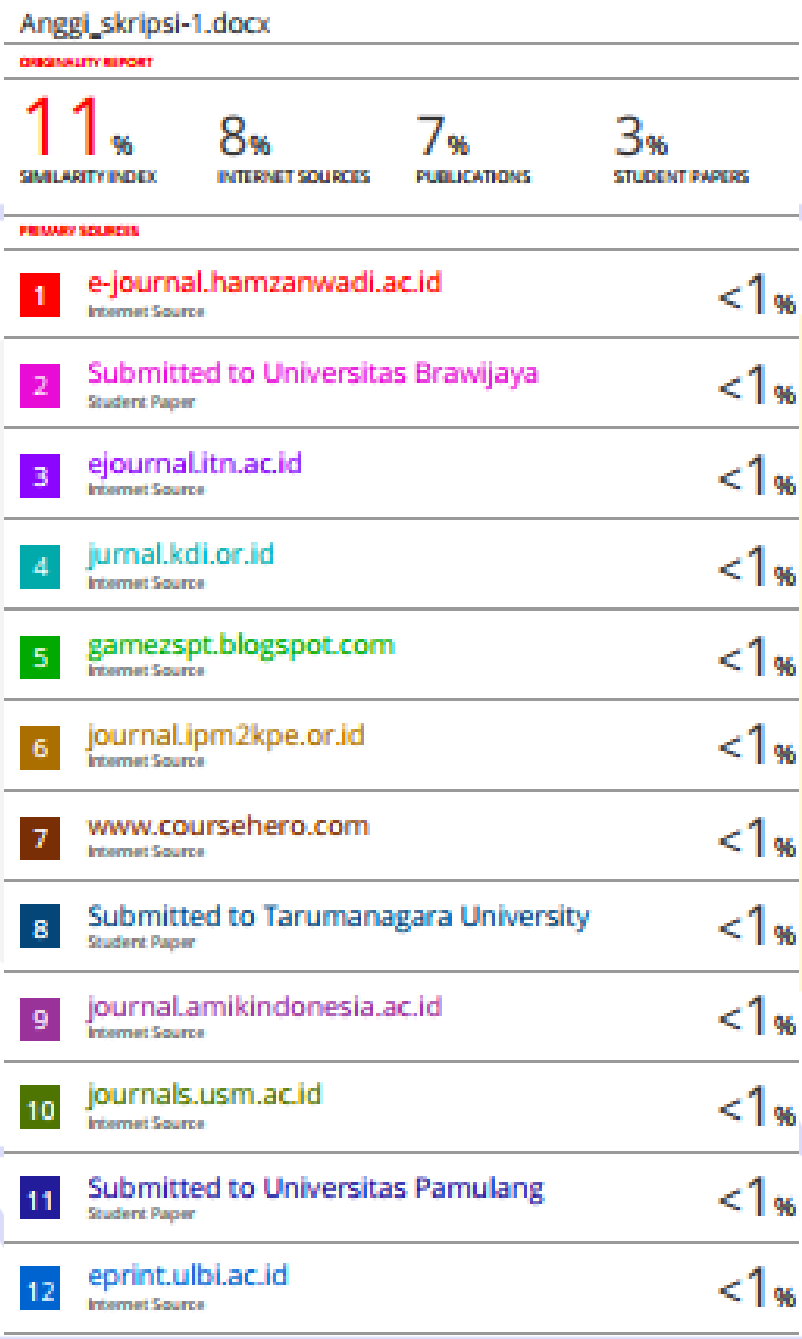
LAMPIRAN



Anggi_skripsi-1.docx

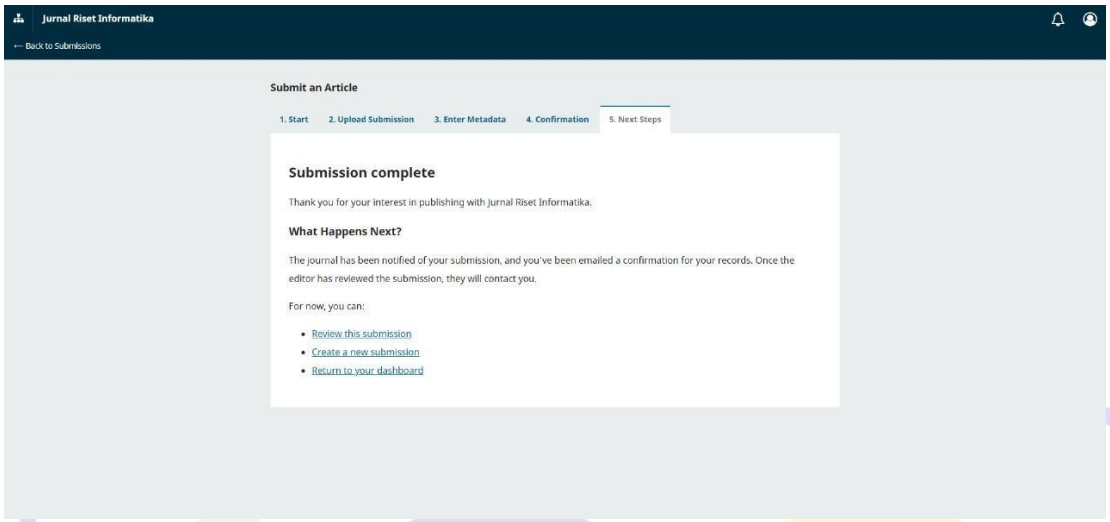
ORIGINALITY REPORT

11%	8%	7%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	e-journal.hamzanwadi.ac.id Internet Source	<1%
2	Submitted to Universitas Brawijaya Student Paper	<1%
3	ejournal.itn.ac.id Internet Source	<1%
4	jurnal.kdi.or.id Internet Source	<1%
5	gamezspt.blogspot.com Internet Source	<1%
6	journal.ipm2kpe.or.id Internet Source	<1%
7	www.coursehero.com Internet Source	<1%
8	Submitted to Tarumanagara University Student Paper	<1%
9	journal.amikindonesia.ac.id Internet Source	<1%
10	journals.usm.ac.id Internet Source	<1%
11	Submitted to Universitas Pamulang Student Paper	<1%
12	eprint.ulbi.ac.id Internet Source	<1%

Lampiran A. Bukti Hasil Pengecekan Plagiarisme



Lampiran B. Bukti Submit Jurnal

SURAT PERNYATAAN KEBENARAN/KEABSAHAN DATA HASIL RISET UNTUK KARYA ILMIAH

Yang bertanda tangan di bawah ini, saya:

Nama	: Rianggi Silvi Anti Butar-Butar
NIM	: 15210018
Mahasiswa Jenjang	: Sarjana (S1)
Program Studi	: Sains Data
Fakultas	: Teknologi Informasi
Perguruan Tinggi	: Universitas Nusa Mandiri

Dengan ini menyatakan bahwa data dan atau informasi yang saya gunakan dalam penulisan karya ilmiah dengan judul **"Impementasi Sistem Rekomendasi *Game* Menggunakan Metode *K-Means Clustering* dan *Content-based Filtering*"** merupakan data dan atau informasi yang saya peroleh berdasarkan hasil Riset Secara Daring (Online) pada:

Nama Repositori	: *Rawg Video Games Database*
URL Repositori	: https://rawg.io/

Saya bersedia untuk bertanggung jawab secara pribadi, tanpa melibatkan pihak Universitas Nusa Mandiri, atas materi/isi karya ilmiah tersebut, termasuk bertanggung jawab atas dampak atau kerugian yang timbul dalam bentuk akibat tindakan yang berkaitan dengan data dan atau informasi yang terdapat pada karya ilmiah saya ini. Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di: Jakarta
Pada tanggal: 4 Agustus 2025

Mengetahui,

Dosen Pembimbing	Asisten Dosen Pembimbing	Yang Menyatakan,
Nanang Ruhyana, M.Kom	Syarah Seimahura, M.Kom	Rianggi Silvi Anti

Lampiran C. Surat Pernyataan Keabsahan Data

SISTEM REKOMENDASI GAME BERBASIS K-MEANS CLUSTERING DAN CONTENT-BASED FILTERING

1. Load dataset

```
#Membaca file Excel dari Google Drive ke DataFrame
df = pd.read_excel('/content/gdrive/MyDrive/Kaggle/Rekomendasi
Game Final.xlsx')
df.head()
```

2. Pemilihan Atribut

```
# Memilih kolom yang relevan untuk analisis dan membangun sistem
rekomendasi
df = df[['Name', 'Rating', 'Genres', 'Platforms', 'ESRB']]
# Menampilkan 5 baris pertama untuk memverifikasi kolom yang
dipilih
df.head()
```

3. Menampilkan struktur Dataframe

```
# Menampilkan informasi struktur DataFrame (kolom, tipe data, dan
jumlah nilai non-null)
print("D STRUKTUR DATAFRAME")
print("="*40)
print(df.info())
print("\n")
```

4. Menampilkan jumlah nilai kosong

```
# Menampilkan jumlah nilai kosong (missing values) pada setiap
kolom DataFrame
print("  JUMLAH NILAI KOSONG PER KOLOM")
print("="*40)
print(df.isnull().sum())
print("\n")
```

5. Cek dan hapus duplikat berdasarkan kolom tertentu

```
# Cek dan hapus duplikat berdasarkan kolom tertentu
jumlah_duplikat = df.duplicated(subset=['Name', 'Genres',
'Platforms', 'Rating', 'ESRB']).sum()
print(" PEMERIKSAAN DAN PENGHAPUSAN DUPLIKAT DATA ")
print("=".center(70, "="))
print(f"Jumlah data sebelum penghapusan duplikat : {len(df)}")
print(f"Jumlah duplikat berdasarkan ['Name', 'Genres',
'Platforms', 'Rating', 'ESRB'] : {jumlah_duplikat}")
# Hapus duplikat
```

```
df.drop_duplicates(subset=['Name', 'Genres', 'Platforms',
'Rating', 'ESRB'], inplace=True)
df.reset_index(drop=True, inplace=True)
print(f"Jumlah data setelah penghapusan duplikat  : {len(df)}")
print("="*70 + "\n")
```

6. Membersihkan spasi

```
# Membersihkan spasi & menyeragamkan format teks kolom
df['Name'] = df['Name'].str.title().str.strip()
df['Genres'] = df['Genres'].str.title().str.strip()
df['Platforms'] = df['Platforms'].str.title().str.strip()
df['ESRB'] = df['ESRB'].str.strip()
```

7. Visualisasi Distribusi Rating *Game*

```
import seaborn as sns
import matplotlib.pyplot as plt
# Pastikan kolom Rating bebas NaN
ratings = df['Rating'].dropna()
plt.figure(figsize=(10, 6))
sns.histplot(ratings, bins=30, kde=True, color='skyblue')
# Tambahkan garis mean dan median
mean_val = ratings.mean()
median_val = ratings.median()
plt.axvline(mean_val, color='red', linestyle='--', label=f'Rata-
rata ({mean_val:.2f})')
plt.axvline(median_val, color='green', linestyle='--',
label=f'Median ({median_val:.2f})')
# Judul dan label
plt.title("Distribusi Rating Game (Histogram + KDE)",
fontsize=14, fontweight='bold')
plt.xlabel("Rating")
plt.ylabel("Frekuensi")
plt.legend()
plt.show()
```

8. Visualisasi jumlah *game* berdasarkan kategori ESRB

```
# Visualisasi jumlah game berdasarkan kategori ESRB
plt.figure(figsize=(10,6))
sns.countplot(data=df, x='ESRB',
order=df['ESRB'].value_counts().index, palette='Set2')
plt.title("Jumlah Game berdasarkan ESRB")
plt.xlabel("ESRB")
plt.ylabel("Jumlah")
plt.show()
```

9. Visualisasi Top 10 Genre *Game* Paling Populer

```
# Menampilkan 10 genre game terpopuler berdasarkan jumlah
kemunculan
top_genres =
df['Genres'].str.split(',').explode().str.strip().value_counts().
head(10)

plt.figure(figsize=(10,6))
sns.barplot(x=top_genres.values, y=top_genres.index,
palette='Blues_r')
plt.title("Top 10 Genre Paling Populer")
plt.xlabel("Jumlah")
plt.ylabel("Genre")
plt.show()
```

10. Visualisasi Top 10 Platform *Game* Paling Populer

```
# Menampilkan 10 genre game terpopuler berdasarkan jumlah
kemunculan
top_platforms =
df['Platforms'].str.split(',').explode().str.strip().value_counts
().head(10)
plt.figure(figsize=(10,6))
sns.barplot(x=top_platforms.values, y=top_platforms.index,
palette='viridis')
plt.title("Top 10 Platform Paling Populer")
plt.xlabel("Jumlah")
plt.ylabel("Platform")
plt.show()
```

11. Feature Engineering (Preprocessing)

a. Standardisasi Fitur Rating

```
# Standardisasi kolom Rating (mean=0, std=1) menggunakan
StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['Rating_scaled'] = scaler.fit_transform(df[['Rating']])
# Menampilkan hasil standardisasi untuk 5 game pertama
print("✓ Rating berhasil dinormalisasi:")
display(df[['Name', 'Rating', 'Rating_scaled']].head())
```

b. One-Hot Encoding Kolom ESRB

```
# Mengonversi kolom ESRB menjadi kode numerik
df['ESRB_cat'] = df['ESRB'].astype('category').cat.codes
# Menampilkan hasil encoding untuk 5 game pertama
print("✓ ESRB berhasil di-encode jadi numerik:")
display(df[['Name', 'ESRB', 'ESRB_cat']].head())
```

c. MultiLabelBinarizer untuk Genres dan Platforms

1) Siapkan list genre

```
# Mengubah kolom Genres menjadi list lalu melakukan multi-hot
encoding
df['Genres_list'] = df['Genres'].str.split(',').apply(lambda x:
[i.strip() for i in x])

from sklearn.preprocessing import MultiLabelBinarizer
#mengubah data teks "Genres" menjadi bentuk numerik
mlb_genres = MultiLabelBinarizer()
genres_encoded = mlb_genres.fit_transform(df['Genres_list'])

df_genres = pd.DataFrame(genres_encoded,
                         columns=[f"Genre_{g.strip()}" for g in
mlb_genres.classes_],
                         index=df.index)
print("✓ Multi-hot encoding untuk Genres selesai:")
display(df_genres.head())
```

2) Siapkan list platform

```
from sklearn.preprocessing import MultiLabelBinarizer
# Ubah string platform menjadi list lalu melakukan multi-hot
encoding
df['Platforms_list'] =
df['Platforms'].str.split(',').apply(lambda x: [i.strip() for i
in x])
from sklearn.preprocessing import MultiLabelBinarizer
#Mengubah kolom data teks "Platforms" menjadi bentuk numerik.
mlb_platforms = MultiLabelBinarizer()
platforms_encoded =
mlb_platforms.fit_transform(df['Platforms_list'])
df_platforms = pd.DataFrame(platforms_encoded,
                            columns=[f"Platform_{p.strip()}" for
p in mlb_platforms.classes_],
                            index=df.index)
print("✓ Multi-hot encoding untuk Platforms selesai:")
display(df_platforms.head())
```

d. Gabungkan Semua Fitur ke Satu DataFrame

```
# Menggabungkan semua fitur hasil preprocessing: Rating, ESRB,
Genres, Platforms
df_encoded = pd.concat([
    df[['Name', 'Rating', 'Genres', 'Platforms', 'ESRB',
'Rating_scaled', 'ESRB_cat']],
    df_genres,
    df_platforms
], axis=1)
print("✅ Data preprocessing selesai. Preview akhir:")
display(df_encoded.head())
```

12. Clustering Preparation (Persiapan KMeans)

a. Ambil Hanya Fitur Numerik

```
# Mengambil hanya kolom fitur numerik untuk proses clustering
features = df_encoded.drop(columns=['Name', 'Rating', 'Genres',
'Platforms', 'ESRB'])
# Menampilkan ringkasan informasi fitur
print(f"✓ Jumlah fitur untuk clustering: {features.shape[1]}")
print(f"✓ Jumlah data: {features.shape[0]}")
print(f"✓ Contoh nama kolom fitur:
{list(features.columns[:10])}")
```

b. Evaluasi Jumlah Cluster dengan Elbow & Silhouette

```
# Menentukan jumlah cluster optimal dengan Elbow Method dan
Silhouette Score
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
k_range = range(2, 11)
inertias = []
silhouette_scores = []
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    cluster_labels = kmeans.fit_predict(features)
    inertias.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(features,
cluster_labels))
# Visualisasi hasil
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))
# Elbow Plot
ax1.plot(k_range, inertias, 'bo-', linewidth=2, markersize=8)
ax1.set_title('Elbow Method untuk Menentukan K Optimal')
ax1.set_xlabel('Jumlah Cluster (k)')
ax1.set_ylabel('Inertia (SSE)')
ax1.grid(True, alpha=0.3)
ax1.axvline(x=3, color='blue', linestyle='--', alpha=0.5)
ax1.text(3, inertias[k_range.index(3)], 'K=3', ha='center',
fontweight='bold')
# Silhouette Score
ax2.plot(k_range, silhouette_scores, 'ro-', linewidth=2,
markersize=8)
ax2.set_title('Silhouette Score vs Jumlah Cluster')
ax2.set_xlabel('Jumlah Cluster (k)')
ax2.set_ylabel('Silhouette Score')
ax2.grid(True, alpha=0.3)
k3_silhouette = silhouette_scores[k_range.index(3)]
ax2.axvline(x=3, color='red', linestyle='--', alpha=0.5)
```

```
ax2.text(3, k3_silhouette, f'Score={k3_silhouette:.3f}',
ha='center', fontweight='bold')
plt.tight_layout()
plt.show()
```

c. Simpan Skor Silhouette K=3

```
# Menetapkan jumlah cluster optimal (K) dan menampilkan nilai
Silhouette Score
optimal_k = 3
max_silhouette = silhouette_scores[k_range.index(optimal_k)]
print(f"✓ Menggunakan K={optimal_k} sesuai evaluasi")
print(f"✓ Silhouette Score: {max_silhouette:.3f}")
```

13. Clustering Implementation (KMeans)

a. Jalankan KMeans Clustering dan Simpan Label

```
# Melakukan clustering dengan KMeans (K=3) dan menambahkan label
cluster ke DataFrame
from sklearn.cluster import KMeans
# Gunakan K=3 (hasil evaluasi sebelumnya)
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
# Jalankan KMeans dan simpan hasil label cluster
clusters = kmeans.fit_predict(features)
# Salin df_encoded untuk menambahkan hasil cluster
df_clustered = df_encoded.copy()
df_clustered['Cluster'] = clusters
# Tampilkan hasil distribusi cluster
print(f"✓ Clustering selesai menggunakan K={optimal_k}")
print("\nD Distribusi cluster:")
print(df_clustered['Cluster'].value_counts().sort_index())
```

b. Analisis Keseimbangan Cluster

```
# Menganalisis keseimbangan jumlah data di setiap cluster
def analyze_cluster_balance(df_clustered):
    cluster_counts =
df_clustered['Cluster'].value_counts().sort_index()
    balance_ratio = cluster_counts.min() / cluster_counts.max()
    print(f"\n☑ Cluster Balance Ratio: {balance_ratio:.3f}")
    print("   (Ratio > 0.4 = Seimbang, 0.2-0.4 = Cukup, < 0.2 =
Tidak Seimbang)")
    for i, count in cluster_counts.items():
        pct = (count / len(df_clustered)) * 100
        print(f"  Cluster {i}: {count:4d} game ({pct:.1f}%)")
    return balance_ratio
# Jalankan analisis dan interpretasi keseimbangan cluster
balance_ratio = analyze_cluster_balance(df_clustered)
# Interpretasi keseimbangan
if balance_ratio > 0.4:
```

```
    print("✅ Cluster SEIMBANG - Baik untuk rekomendasi")
elif balance_ratio > 0.2:
    print("⚠️ Cluster CUKUP SEIMBANG - Masih layak")
else:
    print("❌ Cluster TIDAK SEIMBANG - Perlu revisi K")
```

c. Visualisasi Distribusi Cluster (Bar Plot)

```
# Visualisasi distribusi jumlah game pada setiap cluster
import matplotlib.pyplot as plt
import numpy as np
# Hitung jumlah per cluster
cluster_counts =
df_clustered['Cluster'].value_counts().sort_index()
colors = plt.cm.viridis(np.linspace(0, 1, len(cluster_counts)))
plt.figure(figsize=(10, 6))
bars = plt.bar(cluster_counts.index, cluster_counts.values,
color=colors)
plt.title("Distribusi Jumlah Game per Cluster", fontsize=14,
fontweight='bold')
plt.xlabel("Cluster", fontsize=12)
plt.ylabel("Jumlah Game", fontsize=12)
# Tampilkan jumlah + persentase
for i, (bar, count) in enumerate(zip(bars,
cluster_counts.values)):
    pct = (count / len(df_clustered)) * 100
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() +
1,
             f"{count}\n({pct:.1f}%)", ha='center', va='bottom',
fontweight='bold')
plt.tight_layout()
plt.show()
```

14. Visualisasi Hasil Clustering dalam 2D

```
# Visualisasi sebaran cluster dengan UMAP (reduksi dimensi ke 2D)
!pip install umap-learn
import umap.umap_ as umap
reducer = umap.UMAP(random_state=42)
umap_result = reducer.fit_transform(features)
df_umap = pd.DataFrame(umap_result, columns=['UMAP1', 'UMAP2'])
df_umap['Cluster'] = df_clustered['Cluster']
plt.figure(figsize=(10,6))
sns.scatterplot(data=df_umap, x='UMAP1', y='UMAP2',
hue='Cluster', palette='Set1')
plt.title("Visualisasi Cluster Game dengan UMAP")
plt.show()
```

15. Visualisasi sebaran cluster dengan t-SNE (reduksi dimensi ke 2D)

```
# Visualisasi sebaran cluster dengan t-SNE (reduksi dimensi ke
2D)
from sklearn.manifold import TSNE
# Jalankan t-SNE
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
tsne_result = tsne.fit_transform(features)
# Buat DataFrame dari hasil t-SNE
df_tsne = pd.DataFrame(tsne_result, columns=['Dim1', 'Dim2'])
df_tsne['Cluster'] = df_clustered['Cluster'].values
# Visualisasi t-SNE
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_tsne, x='Dim1', y='Dim2', hue='Cluster',
palette='Set2')
plt.title("Visualisasi Cluster Game dengan t-SNE", fontsize=14,
fontweight='bold')
plt.xlabel("Dimensi 1")
plt.ylabel("Dimensi 2")
plt.legend(title="Cluster")
plt.tight_layout()
plt.show()
```

16. Sistem Rekomendasi Berdasarkan Cluster

a. Fungsi Rekomendasi Cluster

```
# Memberikan rekomendasi game berdasarkan cluster yang sama
dengan game input
def rekomendasi_by_cluster(nama_game, df_clustered, top_n=5):
    nama_game = nama_game.lower()
  # Cari index game berdasarkan nama (pencarian tepat atau
sebagian)
    idx = df_clustered[df_clustered['Name'].str.lower() ==
nama_game].index
    if idx.empty:
        partial =
df_clustered[df_clustered['Name'].str.lower().str.contains(nama_g
ame)]
        if partial.empty:
            print("❌ Game tidak ditemukan.")
            return
        else:
            idx = partial.index[0]
            print(f"✅ Menggunakan hasil terdekat:
{df_clustered.loc[idx, 'Name']}")
    else:
        idx = idx[0]
    # Ambil ID cluster dari game yang ditemukan
    cluster_id = df_clustered.loc[idx, 'Cluster']
    print(f"🎯 Game '{df_clustered.loc[idx, 'Name']}' berada di
cluster {cluster_id}")
```

```
    # Ambil game lain di cluster yang sama (kecuali game input)
    cluster_games = df_clustered[(df_clustered['Cluster'] ==
cluster_id) & (df_clustered.index != idx)]
    # Urutkan berdasarkan rating dan ambil top_n
    rekomendasi = cluster_games.sort_values(by='Rating',
ascending=False).head(top_n)
    rekomendasi = rekomendasi[['Name', 'Rating', 'Genres',
'Platforms', 'ESRB']].reset_index(drop=True)
    print(f"\nD Rekomendasi berdasarkan cluster {cluster_id}:")
    return rekomendasi
```

b. Visualisasi Rekomendasi Berdasarkan Cluster

```
# Membuat visualisasi rekomendasi game (barplot) dari cluster
yang sama dengan game asal
import matplotlib.pyplot as plt
import seaborn as sns
def visualisasi_rekomendasi_cluster(game_name,
hasil_rekomendasi):
    plt.figure(figsize=(10, 6))
    sns.barplot(
        x='Rating',
        y='Name',
        data=hasil_rekomendasi,
        palette='coolwarm',
        edgecolor='black'
    )
    plt.title(f"Visualisasi Rekomendasi Game dari Cluster
Sama\n(Game Asal: {game_name})", fontsize=14, fontweight='bold')
    plt.xlabel("Rating")
    plt.ylabel("Game")
    plt.tight_layout()
    plt.show()
```

17. Sistem Rekomendasi Berbasis Konten (Content-Based Filtering)

a. Persiapan Data Content-Based Filtering

```
# Menyiapkan fitur untuk content-based filtering (Rating, Genre,
Platform, ESRB)
from sklearn.preprocessing import MultiLabelBinarizer,
StandardScaler
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
# Salin dataframe bersih
df_cb = df.copy()


# Normalisasi Rating
df_cb['Rating_scaled'] =
StandardScaler().fit_transform(df_cb[['Rating']])
```

```
# Ubah Genre & Platform menjadi list
df_cb['Genres_list'] =
df_cb['Genres'].str.split(',').apply(lambda x: [i.strip().lower()
for i in x])
df_cb['Platforms_list'] =
df_cb['Platforms'].str.split(',').apply(lambda x:
[i.strip().lower() for i in x])
# One-hot encoding untuk genre & platform
mlb_genres = MultiLabelBinarizer()
genres_encoded = mlb_genres.fit_transform(df_cb['Genres_list'])
mlb_platforms = MultiLabelBinarizer()
platforms_encoded =
mlb_platforms.fit_transform(df_cb['Platforms_list'])
# One-hot encoding untuk ESRB
esrb_encoded = pd.get_dummies(df_cb['ESRB'].str.lower())
# Gabungkan semua fitur ke dalam satu array
content_features = np.hstack([
    df_cb[['Rating_scaled']].values,
    genres_encoded,
    platforms_encoded,
    esrb_encoded.values
])
print("✓ Content-based features siap. Shape:",
content_features.shape)
```

b. Hitung Cosine Similarity

```
# Menghitung matriks kemiripan antar game menggunakan cosine
similarity
cosine_sim = cosine_similarity(content_features)
print("✓ Cosine similarity matrix berhasil dibuat:",
cosine_sim.shape)
```

c. Fungsi Rekomendasi Content-Based

```
# Memberikan rekomendasi game berbasis kemiripan konten (content-
based filtering) menggunakan cosine similarity
from IPython.display import display
def rekomendasi_content_based(nama_game, df_cb, sim_matrix,
top_n=10):
    nama_game = nama_game.lower()
    idx = df_cb[df_cb['Name'].str.lower() == nama_game].index
  # Cek apakah game ditemukan
    if idx.empty:
        print("✗ Game tidak ditemukan.")
        return None
    idx = idx[0]
  # Hitung skor kemiripan dan urutkan
    sim_scores = list(enumerate(sim_matrix[idx]))
```

```
    sim_scores = sorted(sim_scores, key=lambda x: x[1],
reverse=True)
    sim_scores = [i for i in sim_scores if i[0] != idx]  #
exclude diri sendiri
   # Ambil top_n hasil teratas
    top_indices = [i[0] for i in sim_scores[:top_n]]
    top_scores = [i[1] for i in sim_scores[:top_n]]
    # Buat DataFrame hasil rekomendasi
    rekomendasi = df_cb.loc[top_indices, ['Name', 'Rating',
'Genres', 'Platforms', 'ESRB']].copy()
    rekomendasi['Similarity'] = [f"{s:.3f}" for s in top_scores]
    rekomendasi.reset_index(drop=True, inplace=True)
    print(f"\nD Rekomendasi Content-Based untuk:
{df_cb.loc[idx, 'Name']}")
    return rekomendasi
```

d. Visualisasi Rekomendasi Content-Based Filtering

```
# Visualisasi rekomendasi game menggunakan content-based
filtering dalam bentuk barplot
import matplotlib.pyplot as plt
import seaborn as sns
def visualisasi_rekomendasi_content(game_name, df_cb, sim_matrix,
top_n=10):
    game_name_lower = game_name.lower()
    idx = df_cb[df_cb['Name'].str.lower() ==
game_name_lower].index
    # Cek apakah game ditemukan
    if idx.empty:
        print("X Game tidak ditemukan.")
        return
    idx = idx[0]
 # Hitung skor kemiripan dan ambil top_n teratas
    sim_scores = list(enumerate(sim_matrix[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1],
reverse=True)
    sim_scores = [s for s in sim_scores if s[0] != idx]
    top_indices = [i[0] for i in sim_scores[:top_n]]
    top_scores = [i[1] for i in sim_scores[:top_n]]
    top_names = df_cb.loc[top_indices, 'Name']
    # Visualisasi
    plt.figure(figsize=(10, 6))
    sns.barplot(x=top_scores, y=top_names, palette='Blues_d')
    plt.xlabel("Similarity Score")
    plt.ylabel("Recommended Games")
    plt.title(f"🎮 Rekomendasi Content-Based untuk:
{df_cb.loc[idx, 'Name']}", fontsize=14, fontweight='bold')
    plt.xlim(0, 1)
    plt.grid(axis='x', linestyle='--', alpha=0.5)
```

```
    plt.tight_layout()
    plt.show()
```

18. Rekomendasi Hybrid

a. Fungsi Hybrid Recommender

```
import pandas as pd
def rekomendasi_hybrid(nama_game, df_cb, df_clustered,
sim_matrix, top_n=10):
    # --- cari index game target ---
    nama_game = nama_game.lower()
    idx = df_cb[df_cb['Name'].str.lower() == nama_game].index
    if idx.empty:
        partial =
df_cb[df_cb['Name'].str.lower().str.contains(nama_game)]
        if partial.empty:
            print("✗ Game tidak ditemukan.")
            return
        idx = partial.index[0]
        print(f"✓ Menggunakan hasil terdekat: {df_cb.loc[idx,
'Name']}")
    else:
        idx = idx[0]
    cluster_id = df_clustered.loc[idx, 'Cluster']
    # --- content-based ---
    sim_scores = list(enumerate(sim_matrix[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1],
reverse=True)
    sim_scores = [i for i in sim_scores if i[0] != idx]
    cb_idx = [i[0] for i in sim_scores[:top_n // 2]]
    cb_sim = [i[1] for i in sim_scores[:top_n // 2]]
    rec_cb = df_cb.loc[cb_idx,
['Name','Rating','Genres','Platforms','ESRB']].copy()
    rec_cb['Similarity'] = cb_sim
    rec_cb['Source'] = 'Content-Based'
    rec_cb['Cluster'] = df_clustered.loc[cb_idx,
'Cluster'].values
    # --- cluster-based ---
    cluster_candidates = df_clustered[(df_clustered['Cluster'] ==
cluster_id) & (df_clustered.index != idx)]
    cl_pairs = [(i, sim_matrix[idx][i]) for i in
cluster_candidates.index]
    cl_pairs = sorted(cl_pairs, key=lambda x: x[1],
reverse=True)[:top_n // 2]
    cl_idx = [i[0] for i in cl_pairs]
    cl_sim = [i[1] for i in cl_pairs]
    rec_cl = df_cb.loc[cl_idx,
['Name','Rating','Genres','Platforms','ESRB']].copy()
    rec_cl['Similarity'] = cl_sim
```

```
    rec_cl['Source'] = 'Cluster-Based'
    rec_cl['Cluster'] = cluster_id
    # --- normalisasi skor untuk pengurutan seimbang ---
    def minmax(s):
        s = s.astype(float)
        return (s - s.min())/(s.max() - s.min()) if s.max() >
s.min() else pd.Series(0.5, index=s.index)
    rec_cb['cb_norm'] = minmax(rec_cb['Similarity'])
    rec_cb['cl_norm'] = 0.0
    rec_cl['cb_norm'] = 0.0
    rec_cl['cl_norm'] = minmax(rec_cl['Similarity'])
    # --- gabung ---
    df_all = pd.concat([rec_cb, rec_cl], ignore_index=True)
    # Hitung skor gabungan hanya untuk pengurutan (tidak
ditampilkan)
    df_all['sort_score'] = 0.5 * df_all['cb_norm'] + 0.5 *
df_all['cl_norm']
    # --- urutkan & ambil top_n ---
    df_all = df_all.sort_values(by='sort_score',
ascending=False).drop(columns=['cb_norm','cl_norm','sort_score'])
    df_all =
df_all.drop_duplicates(subset='Name').reset_index(drop=True).head
(top_n)
    return df_all
```

b. Visualisasi Rekomendasi Hybrid

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
def visualisasi_rekomendasi_hybrid(game_name, df_cb,
df_clustered, sim_matrix, top_n=10):
    game_name_lower = game_name.lower()
    idx = df_cb[df_cb['Name'].str.lower() ==
game_name_lower].index
    # fallback: cari partial match
    if idx.empty:
        partial =
df_cb[df_cb['Name'].str.lower().str.contains(game_name_lower)]
        if partial.empty:
            print("✗ Game tidak ditemukan.")
            return
        idx = partial.index[0]
        print(f"✓ Menggunakan hasil terdekat: {df_cb.loc[idx,
'Name']}")
    else:
        idx = idx[0]
    # Identifikasi cluster game
    cluster_id = df_clustered.loc[idx, 'Cluster']
```

```
    # === Content-Based ===
    sim_scores = list(enumerate(sim_matrix[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1],
reverse=True)
    sim_scores = [s for s in sim_scores if s[0] != idx]
    k_each = max(1, top_n // 2)
    content_indices = [i[0] for i in sim_scores[:k_each]]
    content_scores  = [i[1] for i in sim_scores[:k_each]]
    df_cb_content = df_cb.loc[content_indices, ['Name']].copy()
    df_cb_content['Score']  = content_scores
    df_cb_content['Source'] = 'Content-Based'
    # === Cluster-Based ===
    same_cluster = df_clustered[(df_clustered['Cluster'] ==
cluster_id) & (df_clustered.index != idx)]
    cluster_indices = same_cluster.index.tolist()
    cluster_sim = [(i, sim_matrix[idx][i]) for i in
cluster_indices]
    cluster_sim_sorted = sorted(cluster_sim, key=lambda x: x[1],
reverse=True)[:k_each]
    clust_idx    = [i[0] for i in cluster_sim_sorted]
    clust_scores = [i[1] for i in cluster_sim_sorted]
    df_cb_clust = df_cb.loc[clust_idx, ['Name']].copy()
    df_cb_clust['Score']  = clust_scores
    df_cb_clust['Source'] = 'Cluster-Based'
    # Gabungkan (untuk ditampilkan)
    df_vis = pd.concat([df_cb_content, df_cb_clust],
ignore_index=True)
    # ==== PENENTUAN URUTAN SEIMBANG (50:50) ====
    # Normalisasi per jalur agar adil
    def minmax(s):
        s = s.astype(float)
        return (s - s.min())/(s.max() - s.min()) if s.max() >
s.min() else pd.Series(0.5, index=s.index)
    # Hitung skor norm per sumber
    tmp = df_vis.copy()
    tmp['cb_norm'] = 0.0
    tmp['cl_norm'] = 0.0
    mask_cb = tmp['Source'] == 'Content-Based'
    mask_cl = tmp['Source'] == 'Cluster-Based'
    tmp.loc[mask_cb, 'cb_norm'] = minmax(tmp.loc[mask_cb,
'Score'])
    tmp.loc[mask_cl, 'cl_norm'] = minmax(tmp.loc[mask_cl,
'Score'])
    # Agregasi ke level Name untuk dapatkan skor gabungan (hanya
untuk sorting)
    order_df = (tmp.groupby('Name', as_index=False)
                   .agg({'cb_norm':'max', 'cl_norm':'max'}))
```

```
    order_df['sort_score'] = 0.5*order_df['cb_norm'] +
0.5*order_df['cl_norm']
    # Merge sort_score ke df_vis lalu urutkan, tapi tetap
tampilkan Score & Source aslinya
    df_vis = df_vis.merge(order_df[['Name','sort_score']],
on='Name', how='left')
    df_vis = df_vis.sort_values(by='sort_score', ascending=True)
    # === Visualisasi ===
    plt.figure(figsize=(10, 6))
    sns.barplot(data=df_vis, x='Score', y='Name', hue='Source',
dodge=False)
    plt.xlabel("Similarity Score")
    plt.ylabel("Recommended Games")
    plt.title(f"🎮 Hybrid Recommendation (Balanced 50:50) untuk:
{df_cb.loc[idx, 'Name']} (Cluster {cluster_id})",
              fontsize=14, fontweight='bold')
    plt.xlim(0, 1)
    plt.legend(loc='lower right', title='')
    plt.grid(axis='x', linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()
```

19. Tahap Akhir: Menyimpan Semua Hasil ke File

a. Persiapan

```
# === PERSIAPAN ===
all_hybrid = pd.DataFrame()
jumlah_gagal = 0
# === LOOP UNTUK SEMUA GAME DALAM df_cb ===
for i, nama in enumerate(df_cb['Name'].tolist()):
    try:
        rekomendasi = rekomendasi_hybrid(nama, df_cb,
df_clustered, cosine_sim, top_n=10)
        rekomendasi['Input_Game'] = nama
        all_hybrid = pd.concat([all_hybrid, rekomendasi],
ignore_index=True)
    except Exception as e:
        print(f"❌ Gagal untuk: {nama} — {e}")
        jumlah_gagal += 1

print(f"\n✅ Proses selesai dengan {jumlah_gagal} kegagalan dari
{len(df_cb)} game.")
```

b. Membuat folder output jika belum ada

```
import os
import joblib
import numpy as np
# Buat folder output jika belum ada
output_dir = 'output'
os.makedirs(output_dir, exist_ok=True)
```

```
all_hybrid.to_csv("output/all_hybrid_recommendations.csv",
index=False)
print("✓ all_hybrid_recommendations.csv berhasil disimpan")
```

c. Simpan semua output

```
import os
import joblib
import numpy as np
# === 1. Buat Folder Output ===
output_dir = 'output'
os.makedirs(output_dir, exist_ok=True)
# === 2. Simpan File Preprocessing ===
df.to_csv(f"{output_dir}/games_preprocessed.csv", index=False)
print("✓ games_preprocessed.csv disimpan")
# === 3. Simpan File Setelah Encoding ===
df_encoded.to_csv(f"{output_dir}/games_encoded.csv", index=False)
print("✓ games_encoded.csv disimpan")
# === 4. Simpan File Setelah Clustering ===
df_clustered.to_csv(f"{output_dir}/games_clustered_k3.csv",
index=False)
print("✓ games_clustered_k3.csv disimpan")
# === 5. Simpan Model KMeans ===
joblib.dump(kmeans, f"{output_dir}/kmeans_model_k3.pkl")
print("✓ Model KMeans disimpan sebagai .pkl")
# === 6. Simpan Cosine Similarity Matrix ===
np.save(f"{output_dir}/cosine_similarity_matrix.npy", cosine_sim)
print("✓ Cosine similarity matrix disimpan (.npy)")
# === 7. Simpan DataFrame Content-Based ===
df_cb.to_csv(f"{output_dir}/df_cb_content_ready.csv",
index=False)
print("✓ df_cb_content_ready.csv disimpan")
# === 8. Simpan Hasil Rekomendasi Hybrid (Semua Game) ===
try:
    all_hybrid.to_csv(f"{output_dir}/all_hybrid_recommendations.c
sv", index=False)
    print("✓ all_hybrid_recommendations.csv disimpan")
except NameError:
    print("⚠ all_hybrid belum dibuat. Jalankan rekomendasi_hybrid
untuk semua game terlebih dahulu.")
```