Catatan Kuliah: Sistem Informasi Cerdas

Data Understanding dan Data Preparation

Disusun oleh: Denny Sihombing
Universitas Katolik Indonesia Atma Jaya

April 2025

Daftar Isi

1 Pendahuluan

Dalam pengembangan Sistem Informasi Cerdas, terutama yang berbasis pada data seperti sistem Machine Learning, kualitas data memegang peranan yang sangat fundamental. Prinsip *Garbage In, Garbage Out* (GIGO) menegaskan bahwa input data yang buruk akan menghasilkan output atau model yang buruk pula, tidak peduli seberapa canggih algoritma yang digunakan.

Oleh karena itu, dua fase awal dalam proses standar penemuan pengetahuan dalam basis data (Knowledge Discovery in Databases - KDD) atau metodologi CRISP-DM (Cross-Industry Standard Process for Data Mining) menjadi sangat krusial: **Data Understanding** dan **Data Preparation**.

- **Data Understanding:** Fase ini berfokus pada pengenalan awal terhadap data. Tujuannya adalah untuk memahami karakteristik data, mengidentifikasi potensi masalah kualitas (seperti data hilang atau tidak konsisten), menemukan pola-pola awal yang menarik, dan mendapatkan wawasan pertama tentang struktur data. Familiarisasi dengan data adalah kunci utama di tahap ini.
- **Data Preparation:** Fase ini mencakup semua aktivitas yang diperlukan untuk membersihkan, mentransformasi, dan menyusun data mentah menjadi dataset final yang siap digunakan untuk tahap pemodelan. Ini seringkali merupakan fase yang paling memakan waktu dalam proyek data science, namun sangat menentukan kualitas model yang akan dihasilkan.

Catatan kuliah ini akan membahas langkah-langkah utama dalam kedua fase tersebut, disertai contoh implementasi menggunakan Python dengan library Pandas, Matplotlib, dan Seaborn, berdasarkan studi kasus dataset `student_depression_dataset.csv`.

2 Tahap 1: Data Understanding

Tujuan utama dari tahap Data Understanding adalah untuk mendapatkan pemahaman mendalam tentang data yang dimiliki sebelum melangkah ke pemodelan. Ini melibatkan serangkaian aktivitas eksplorasi dan verifikasi kualitas data.

2.1 Memuat Dataset Awal

Langkah paling awal adalah memuat data ke dalam lingkungan analisis kita. Kita akan menggunakan library Pandas di Python untuk memuat data dari file CSV.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Tentukan path file dataset Anda
file_path = 'student_depression_dataset.csv'

# Memuat dataset dari file CSV
try:
    df = pd.read_csv(file_path)
    print(f"Dataset '{file_path}' berhasil dimuat.")
    print(f"Dimensi dataset: {df.shape[0]} baris, {df.shape[1]} kolom")
except FileNotFoundError:
    print(f"Error: File '{file_path}' tidak ditemukan.")
    df = None # Set df ke None jika gagal
except Exception as e:
    print(f"Error lain saat memuat dataset: {e}")
    df = None
```

```python

# Menampilkan 5 baris pertama jika berhasil dimuat
if df is not None:
    print("\n5 Baris Pertama Dataset:")
    # Mengatur agar semua kolom ditampilkan (opsional, berguna jika kolom banyak)
    # pd.set_option('display.max_columns', None)
    print(df.head())
```

Listing 1: Kode Python untuk memuat dataset

Output dan Observasi Awal (berdasarkan PDF): Kode di atas berhasil memuat dataset. Output `df.head()` (PDF hlm. 1) menampilkan 5 baris pertama, memberikan gambaran awal tentang kolom-kolom yang ada dan tipe data di dalamnya (misalnya, 'Gender', 'City', 'Profession' tampak kategorikal; 'Age', 'CGPA' tampak numerik). Dimensi dataset adalah 27901 baris dan 18 kolom.

2.2 Menjelajahi Data: Informasi Dasar & Statistik

Setelah data dimuat, kita perlu memahami struktur dasarnya dan mendapatkan ringkasan statistik.

2.2.1 Informasi Dasar (df.info())

Metode `info()` pada DataFrame Pandas memberikan ringkasan ringkas, termasuk jumlah total baris, jumlah kolom, jumlah nilai non-null untuk setiap kolom, tipe data setiap kolom, dan penggunaan memori.

```python
if df is not None:
    # Menampilkan informasi dasar (tipe data, non-null counts, memory usage)
    print("\nInformasi Dasar Dataset (df.info()):")
    df.info()
else:
    print("Dataset belum dimuat.")
```

Listing 2: Kode Python untuk menampilkan informasi dasar

Output dan Observasi (berdasarkan PDF hlm. 2):

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27901 entries, 0 to 27900
Data columns (total 18 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   id                                     27901 non-null  int64
 1   Gender                                 27901 non-null  object
 2   Age                                    27901 non-null  float64
 3   City                                   27901 non-null  object
 4   Profession                             27901 non-null  object
 5   Academic Pressure                      27901 non-null  float64
 6   Work Pressure                          27901 non-null  float64
 7   CGPA                                   27901 non-null  float64
 8   Study Satisfaction                     27901 non-null  float64
 9   Job Satisfaction                       27901 non-null  float64
 10  Sleep Duration                         27901 non-null  object
 11  Dietary Habits                         27901 non-null  object
 12  Degree                                 27901 non-null  object
```

```
 13  Have you ever had suicidal thoughts?  27901 non-null  object
 14  Work/Study Hours                      27901 non-null  float64
 15  Financial Stress                      27901 non-null  object
 16  Family History of Mental Illness     27901 non-null  object
 17  Depression                            27901 non-null  int64
dtypes: float64(7), int64(2), object(9)
memory usage: 3.8+ MB
```

Dari output ini, kita dapat menyimpulkan:

- Dataset memiliki 27901 entri (baris) dan 18 kolom.
- Tidak ada nilai null (*missing values*) pada saat pemuatan awal, karena semua kolom menunjukkan `27901 non-null`.
- Terdapat 7 kolom dengan tipe `float64`, 2 kolom `int64` (termasuk 'id' dan target 'Depression'), dan 9 kolom `object`. Kolom `object` biasanya berisi string dan merupakan kandidat kuat untuk data kategorikal.
- Kolom `Financial Stress` terdeteksi sebagai `object`, meskipun mungkin diharapkan numerik (perlu investigasi lebih lanjut atau konversi tipe nanti).
- Penggunaan memori sekitar 3.8 MB.

2.2.2 Statistik Deskriptif (df.describe())

Metode `describe()` menghitung statistik ringkasan untuk kolom-kolom numerik (secara default) atau kolom kategorikal (jika `include='object'` digunakan).

```python
if df is not None:
    # Menampilkan statistik deskriptif untuk kolom numerik
    print("\nStatistik Deskriptif (Numerik):")
    print(df.describe().T) # .T untuk transpose agar lebih mudah dibaca

    # Menampilkan statistik deskriptif untuk kolom object/kategorikal
    print("\nStatistik Deskriptif (Kategorikal/Object):")
    print(df.describe(include=['object', 'category']).T)
else:
    print("Dataset belum dimuat.")
```

Listing 3: Kode Python untuk menampilkan statistik deskriptif

Output dan Observasi (berdasarkan PDF hlm. 3-4):

- **Numerik:**
  - `id`: Tampaknya hanya identifier, tidak relevan secara statistik.
  - `Age`: Rentang usia dari 18 hingga 59 tahun, dengan rata-rata sekitar 25.8 tahun. 75
  - `Academic Pressure`, `Study Satisfaction`: Skala kemungkinan 0-5.
  - `Work Pressure`, `Job Satisfaction`: Nilai mean, std, 25
  - `CGPA`: Rentang dari 0 hingga 10, rata-rata 7.65. Nilai minimum 0 mungkin perlu diperiksa (apakah valid?).
  - `Work/Study Hours`: Rata-rata 7.15 jam, rentang 0-12 jam.
  - `Depression`: Variabel target biner (0 atau 1). Rata-rata 0.585 menunjukkan bahwa sekitar 58.5% dari sampel memiliki nilai 1 (terindikasi depresi).
- **Kategorikal:**

- `Gender`: 2 kategori unik, 'Male' paling umum (15547 kali).
- `City`: 52 kota unik, 'Kalyan' paling sering muncul (1570 kali). Banyaknya kategori unik bisa menjadi tantangan untuk encoding.
- `Profession`: 14 profesi unik, 'Student' sangat dominan (27870 dari 27901). Ini menunjukkan dataset mayoritas adalah mahasiswa. Kolom ini mungkin kurang informatif jika hampir semua sama.
- `Sleep Duration`: 5 kategori, 'Less than 5 hours' paling umum.
- `Dietary Habits`: 4 kategori, 'Unhealthy' paling umum.
- `Degree`: 28 gelar unik, "'Class 12'" paling umum (6080 kali). Banyak kategori unik.
- `Have you ever had suicidal thoughts?`: 2 kategori ('Yes'/'No'), 'Yes' lebih umum (17656 kali).
- `Financial Stress`: 6 kategori unik, '5.0' paling umum. Meskipun nilainya angka, tipe datanya object, menandakan perlu konversi.
- `Family History of Mental Illness`: 2 kategori ('Yes'/'No'), 'No' sedikit lebih umum (14398 kali).

2.3 Verifikasi Kualitas Data: Pemeriksaan Nilai Hilang

Meskipun `df.info()` menunjukkan tidak ada nilai null, selalu baik untuk secara eksplisit memeriksa jumlah dan persentase nilai hilang per kolom.

```
if df is not None:
    # Menghitung jumlah nilai hilang per kolom
    missing_values = df.isnull().sum()

    if missing_values.sum() == 0:
        print("\nTidak ditemukan nilai hilang dalam dataset.")
    else:
        # Menghitung persentase nilai hilang per kolom
        missing_percentage = (missing_values / len(df)) * 100
        # Menggabungkan hasil ke dalam DataFrame
        missing_info = pd.DataFrame({
            'Jumlah Hilang': missing_values,
            'Persentase Hilang': missing_percentage
        })
        # Menampilkan hanya kolom yang memiliki nilai hilang
        print("\nInformasi Nilai Hilang:")
        print(missing_info[missing_info['Jumlah Hilang'] > 0].sort_values(by='
    Persentase Hilang', ascending=False))
else:
    print("Dataset belum dimuat.")
```

Listing 4: Kode Python untuk memeriksa nilai hilang

Output dan Observasi (berdasarkan PDF hlm. 5): Output kode ini mengkonfirmasi hasil dari `df.info()`, yaitu tidak ada nilai hilang (*missing values*) di seluruh kolom dataset ini. Ini menyederhanakan tahap Data Preparation karena kita tidak perlu melakukan imputasi nilai hilang.

2.4 Eksplorasi dan Visualisasi Data

Visualisasi data adalah alat yang sangat ampuh untuk memahami distribusi, hubungan, dan potensi outlier dalam data.

2.4.1 Distribusi Fitur Numerik

Histogram dapat digunakan untuk melihat bentuk distribusi dari setiap fitur numerik.

```
if df is not None:
    try:
        print("\nMembuat histogram untuk fitur numerik...")
        df.hist(figsize=(12, 10), bins=20) # bins disesuaikan
        plt.tight_layout()
        plt.show()
    except Exception as e:
        print(f"Gagal membuat histogram: {e}")
else:
    print("Dataset belum dimuat.")
```

Listing 5: Kode Python untuk plot histogram fitur numerik

Observasi dari Histogram (PDF hlm. 7):

- `id`: Distribusi tampak seragam, mengkonfirmasi ini adalah identifier.
- `Age`: Sangat miring ke kanan (*right-skewed*), mayoritas besar data terkonsentrasi pada usia muda (sekitar 18-30).
- `Academic Pressure, Study Satisfaction`: Distribusi diskrit pada skala nilai (kemungkinan 0-5).
- `Work Pressure, Job Satisfaction`: Mayoritas absolut nilai berada di 0.
- `CGPA`: Distribusi mendekati normal, sedikit miring ke kiri (*left-skewed*), dengan puncak di sekitar 8-9. Ada beberapa nilai di dekat 0 yang mungkin perlu diperiksa.
- `Work/Study Hours`: Distribusi multimodal dengan beberapa puncak (misalnya sekitar 3-4 jam, 8 jam, 10-11 jam).
- `Depression`: Variabel biner dengan lebih banyak nilai 1 daripada 0.

2.4.2 Distribusi Fitur Kategorikal

Countplot (diagram batang) digunakan untuk melihat frekuensi setiap kategori dalam fitur kategorikal.

```
if df is not None:
    categorical_cols = df.select_dtypes(include=['object']).columns
    print("\nMembuat countplot untuk fitur kategorikal...")
    for column in categorical_cols:
        try:
            plt.figure(figsize=(10, 6)) # Sesuaikan ukuran jika perlu
            # Mengambil 20 kategori teratas jika > 20 agar plot tidak terlalu
    ramai
            top_categories = df[column].value_counts().nlargest(20).index
            sns.countplot(data=df[df[column].isin(top_categories)], y=column,
    order=top_categories, palette='viridis')
            plt.title(f'Distribusi {column} (Top 20 jika > 20)')
            plt.xlabel('Jumlah')
            plt.ylabel(column)
            plt.tight_layout()
            plt.show()
        except Exception as e:
            print(f"Gagal membuat countplot untuk {column}: {e}")
else:
    print("Dataset belum dimuat.")
```

Listing 6: Kode Python untuk plot countplot fitur kategorikal

Observasi dari Countplots (PDF hlm. 8-16):

- `Gender`: Lebih banyak 'Male' daripada 'Female'.
- `City`: Distribusi sangat tidak merata, beberapa kota dominan, banyak kota hanya muncul sedikit (Plot mungkin hanya menampilkan top 20 untuk kejelasan).
- `Profession`: Sangat didominasi oleh 'Student'.
- `Sleep Duration`: Kategori 'Less than 5 hours', '5-6 hours', '7-8 hours' cukup umum. 'More than 8 hours' dan 'Others' lebih jarang.
- `Dietary Habits`: 'Unhealthy' dan 'Moderate' paling umum.
- `Degree`: "'Class 12'" sangat dominan, diikuti oleh beberapa gelar Sarjana dan Magister. Banyak kategori dengan frekuensi rendah (Plot mungkin hanya menampilkan top 20).
- `Have you ever had suicidal thoughts?`: 'Yes' lebih banyak daripada 'No'.
- `Financial Stress`: Terlihat seperti skala ordinal (meskipun tipenya object). Kategori '5.0' paling banyak.
- `Family History of Mental Illness`: 'No' sedikit lebih banyak dari 'Yes'.

2.4.3 Hubungan Antar Variabel

Boxplot dapat digunakan untuk melihat hubungan antara variabel numerik dan target, sementara countplot dengan `hue` dapat melihat hubungan antara variabel kategorikal dan target.

```
if df is not None and 'Depression' in df.columns:
    print("\nMembuat visualisasi hubungan dengan variabel target...")
    # Hubungan Numerik vs Target (contoh: Age, CGPA)
    numerical_to_plot = ['Age', 'CGPA']
    for col in numerical_to_plot:
        if col in df.columns:
            plt.figure(figsize=(8, 5))
            sns.boxplot(data=df, x='Depression', y=col)
            plt.title(f'Hubungan antara {col} dan Depression')
            plt.show()

    # Hubungan Kategorikal vs Target (contoh: Gender, Dietary Habits)
    categorical_to_plot = ['Gender', 'Dietary Habits']
    for col in categorical_to_plot:
        if col in df.columns:
            plt.figure(figsize=(8, 5))
            sns.countplot(data=df, x='Depression', hue=col, palette='pastel')
            plt.title(f'Hubungan antara {col} dan Depression')
            plt.show()
else:
    print("Dataset belum dimuat atau kolom 'Depression' tidak ada.")

# Contoh lain: Heatmap Korelasi (seperti di slide sebelumnya)
# if df is not None: ... (kode heatmap) ...
```

Listing 7: Kode Python untuk visualisasi hubungan dengan target

Observasi dari Visualisasi Hubungan (PDF hlm. 17-20):

- **Boxplot Age vs Depression (hlm. 17):** Distribusi usia tampak sedikit lebih tinggi untuk kelompok yang tidak depresi (0) dibandingkan yang depresi (1).
- **Boxplot CGPA vs Depression (hlm. 18):** Kelompok yang tidak depresi (0) cenderung memiliki CGPA yang sedikit lebih tinggi dibandingkan kelompok yang depresi (1).

- **Countplot Gender vs Depression (hlm. 19):** Baik pada kelompok depresi (1) maupun tidak (0), jumlah 'Male' lebih banyak.
- **Countplot Dietary Habits vs Depression (hlm. 20):** Pola tampak berbeda. Pada kelompok tidak depresi (0), 'Healthy' dan 'Moderate' lebih dominan. Pada kelompok depresi (1), 'Unhealthy' menjadi sangat dominan, menunjukkan potensi hubungan.

Tahap Data Understanding memberikan kita pemahaman yang solid tentang data, kualitasnya, dan pola-pola awal sebelum melangkah ke persiapan data untuk pemodelan.

3 Tahap 2: Data Preparation

Setelah memahami data, tahap selanjutnya adalah mempersiapkannya agar sesuai untuk algoritma machine learning.

3.1 Penanganan Nilai Hilang (Handling Missing Values)

Berdasarkan analisis pada tahap Data Understanding, dataset `student_depression_dataset.csv` ini **tidak memiliki nilai hilang**. Oleh karena itu, langkah imputasi atau penghapusan data hilang tidak diperlukan. Jika ada, strategi seperti penghapusan atau imputasi (mean, median, modus, dll.) akan diterapkan di sini. Kita akan menggunakan `df` asli untuk tahap selanjutnya.

3.2 Konversi Tipe Data & Encoding Variabel Kategorikal

3.2.1 Konversi Tipe Data (Financial Stress)

Kolom `Financial Stress` perlu diubah menjadi tipe numerik.

```
if df is not None and 'Financial Stress' in df.columns:
    # Coba konversi ke numerik, paksa error menjadi NaN
    df['Financial Stress'] = pd.to_numeric(df['Financial Stress'], errors='coerce')
    # Cek apakah ada NaN yang muncul setelah konversi (jika ada nilai non-numerik)
    if df['Financial Stress'].isnull().sum() > 0:
        print("Peringatan: Ada nilai non-numerik di 'Financial Stress', menghasilkan NaN.")
        # Handle NaN jika muncul (misal: imputasi median)
        median_fs = df['Financial Stress'].median()
        df['Financial Stress'].fillna(median_fs, inplace=True)
        print(f"NaN di 'Financial Stress' diimputasi dengan median: {median_fs}")
    else:
        print("Kolom 'Financial Stress' berhasil dikonversi ke numerik.")
    # Verifikasi tipe data baru
    print(f"Tipe data baru 'Financial Stress': {df['Financial Stress'].dtype}")
else:
    print("Dataset atau kolom 'Financial Stress' tidak ditemukan.")
```

Listing 8: Konversi tipe data 'Financial Stress'

3.2.2 Encoding Variabel Kategorikal

Menggunakan **One-Hot Encoding** untuk variabel kategorikal nominal.

```
if df is not None:
    print("\n--- Memulai Encoding Variabel Kategorikal ---")
    # Identifikasi kolom kategorikal (tipe 'object' setelah Financial Stress dikonversi)
```

```
    categorical_cols_to_encode = df.select_dtypes(include=['object', 'category']).columns.tolist()

    # Hapus kolom yang tidak relevan jika ada (misal: id jika tidak dihapus sebelumnya)
    if 'id' in categorical_cols_to_encode:
        categorical_cols_to_encode.remove('id') # Contoh jika 'id' masih object

    if categorical_cols_to_encode:
        print(f"Kolom kategorikal yang akan di-encode: {categorical_cols_to_encode}")

        # Terapkan One-Hot Encoding
        df_encoded = pd.get_dummies(df, columns=categorical_cols_to_encode, drop_first=True, dtype=int)

        # Hapus kolom 'id' asli jika masih ada dan tidak diperlukan
        if 'id' in df_encoded.columns:
            df_encoded = df_encoded.drop('id', axis=1)
            print("Kolom 'id' telah dihapus.")

        print(f"\nDimensi dataset setelah One-Hot Encoding: {df_encoded.shape}")
        print("Contoh beberapa nama kolom baru:")
        print(df_encoded.columns[:15].tolist(), "...") # Tampilkan beberapa awal saja
    else:
        print("Tidak ada kolom kategorikal yang perlu di-encode.")
        df_encoded = df.copy()
        if 'id' in df_encoded.columns: # Hapus id jika tidak ada encoding
            df_encoded = df_encoded.drop('id', axis=1)
            print("Kolom 'id' telah dihapus.")

else:
    print("Dataframe 'df' tidak tersedia.")
    df_encoded = None
```

Listing 9: One-Hot Encoding Variabel Kategorikal

Hasil Encoding: Proses ini menghasilkan DataFrame `df_encoded` dengan jumlah kolom yang bertambah secara signifikan karena banyaknya kategori unik, terutama pada fitur 'City' dan 'Degree'.

3.3 Normalisasi / Standarisasi Fitur Numerik (Scaling)

Menggunakan **Min-Max Scaling** untuk membawa fitur numerik ke rentang [0, 1]. (Ingat best practice: lakukan ini setelah split data).

```
from sklearn.preprocessing import MinMaxScaler

if 'df_encoded' in locals() and df_encoded is not None:
    print("\n--- Memulai Scaling Fitur Numerik ---")
    # Identifikasi fitur numerik ASLI (sebelum OHE) untuk di-scale
    original_numerical_cols = ['Age', 'Academic Pressure', 'Work Pressure', 'CGPA',
                               'Study Satisfaction', 'Job Satisfaction',
                               'Work/Study Hours', 'Financial Stress'] # Termasuk Financial Stress
    if 'Depression' in original_numerical_cols:
         original_numerical_cols.remove('Depression')
    numerical_features_to_scale = [col for col in original_numerical_cols if col in df_encoded.columns]

    if numerical_features_to_scale:
```

```
        print(f"Fitur numerik yang akan di-scale: {numerical_features_to_scale}")
        scaler = MinMaxScaler()
        df_scaled = df_encoded.copy()
        # Terapkan scaler (Idealnya fit_transform di train, transform di test)
        df_scaled[numerical_features_to_scale] = scaler.fit_transform(df_scaled[numerical_features_to_scale])

        print("\nContoh data setelah normalisasi (fitur numerik yang di-scale):")
        print(df_scaled[numerical_features_to_scale].head())
        print("\nStatistik deskriptif setelah scaling (cek min/max):")
        print(df_scaled[numerical_features_to_scale].describe().T)
    else:
        print("Tidak ada fitur numerik asli yang perlu di-scale.")
        df_scaled = df_encoded.copy()
else:
    print("Dataframe 'df_encoded' tidak tersedia.")
    df_scaled = None
```

Listing 10: Min-Max Scaling Fitur Numerik

Hasil Scaling: DataFrame `df_scaled` berisi data dengan fitur numerik asli yang diskalakan ke [0, 1].

3.4 Pembagian Dataset (Train-Test Split)

Membagi data menjadi 80% training set dan 20% testing set, menjaga proporsi kelas target.

```
from sklearn.model_selection import train_test_split

if 'df_scaled' in locals() and df_scaled is not None and 'Depression' in df_scaled.columns:
    print("\n--- Memulai Pembagian Dataset (Train-Test Split) ---")
    target_column = 'Depression'
    X = df_scaled.drop(target_column, axis=1)
    y = df_scaled[target_column]

    print(f"Ukuran Fitur (X) sebelum split: {X.shape}")
    print(f"Ukuran Target (y) sebelum split: {y.shape}")

    try:
        X_train, X_test, y_train, y_test = train_test_split(
            X, y,
            test_size=0.2,      # 20% untuk test set
            random_state=42,    # Seed untuk reproduktibilitas
            stratify=y          # Menjaga proporsi kelas target
        )
        print("\nUkuran dataset setelah dibagi:")
        print(f"X_train shape: {X_train.shape}")
        print(f"X_test shape:  {X_test.shape}")
        print(f"y_train shape: {y_train.shape}")
        print(f"y_test shape:  {y_test.shape}")
        print("\nProporsi target di y_train:")
        print(y_train.value_counts(normalize=True).map('{:.2%}'.format))
        print("\nProporsi target di y_test:")
        print(y_test.value_counts(normalize=True).map('{:.2%}'.format))
        print("\nDataset siap untuk tahap pemodelan!")
    except Exception as e:
        print(f"\nError saat membagi dataset: {e}")
else:
```

```
    print("Dataframe 'df_scaled' atau kolom target 'Depression' tidak tersedia."
    )
```

Listing 11: Membagi Dataset menjadi Train dan Test Set

Hasil Akhir: Kita sekarang memiliki empat set data: `X_train`, `y_train`, `X_test`, dan `y_test`, siap untuk pemodelan.

4 Kesimpulan

Tahap **Data Understanding** dan **Data Preparation** merupakan fondasi yang tidak terpisahkan dalam membangun Sistem Informasi Cerdas yang efektif dan berbasis data.

Melalui **Data Understanding**, kita memperoleh wawasan mendalam mengenai karakteristik data, distribusi nilai, potensi masalah kualitas, serta hubungan awal antar variabel. Contoh dari dataset `student_depression_dataset.csv` menunjukkan bagaimana kita dapat mengidentifikasi tipe data, distribusi usia, dominasi profesi 'Student', potensi hubungan antara kebiasaan makan dan depresi, serta memastikan tidak adanya nilai hilang.

Selanjutnya, **Data Preparation** mengambil temuan dari tahap understanding untuk membersihkan dan mentransformasi data ke dalam format yang optimal untuk pemodelan. Ini mencakup langkah-langkah krusial seperti penanganan nilai hilang (jika ada), konversi tipe data, encoding variabel kategorikal, scaling fitur numerik, dan membagi data menjadi set pelatihan dan pengujian.

Kedua fase ini seringkali bersifat iteratif dan memakan porsi waktu yang signifikan dalam siklus proyek data. Namun, investasi waktu dan perhatian pada pemahaman dan persiapan data akan sangat menentukan kualitas, keandalan, dan akurasi model atau sistem cerdas yang dihasilkan. Data yang dipersiapkan dengan baik adalah kunci menuju hasil analisis yang bermakna dan solusi cerdas yang efektif.