SISTEM PREDIKSI DIAGNOSIS KESEHATAN MENTAL

MENGGUNAKAN RANDOM FOREST DAN AHP

Diajukan Untuk Memenuhi Tugas Sistem Penunjang Keputusan

Kelompok

19220737 - Rafif Setyo Nugroho

19220918 - Matius Dimas Prasetia

19221464 - Fahmi Akmal Aziz Pane

19220827 - Anugrah Akbar Riyadi

19220021 - Muhammad Reza Pahlevy

PROGRAM STUDI SISTEM INFORMASI

FAKULTAS TEKNIK DAN INFORMATIKA

UNIVERSITAS BINA SARANA INFORMATIKA JATIWARINGIN

TAHUN 2025

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Pendahuluan

Kesehatan mental merupakan aspek krusial dari kesejahteraan individu dan masyarakat. Gangguan kesehatan mental, seperti depresi dan gangguan bipolar, dapat berdampak signifikan pada kualitas hidup. Diagnosis dini dan akurat memainkan peran penting dalam memfasilitasi perawatan yang tepat. Namun, diagnosis kesehatan mental seringkali bersifat kompleks, melibatkan evaluasi subjektif terhadap berbagai gejala dan faktor.

Dalam beberapa tahun terakhir, *machine learning* telah menunjukkan potensi besar dalam mendukung diagnosis medis, termasuk kesehatan mental, dengan mengidentifikasi pola-pola dalam data yang mungkin tidak terlihat oleh analisis manual. Random Forest, khususnya, dikenal karena kemampuannya menangani data kompleks, ketahanan terhadap *overfitting*, dan kemampuannya untuk mengukur *feature importance*.

Meskipun *machine learning* dapat memberikan prediksi, proses pengambilan keputusan dalam ranah medis seringkali membutuhkan lebih dari sekadar keluaran prediktif. Dibutuhkan pemahaman tentang faktor-faktor pendorong prediksi dan cara berbagai kriteria berkontribusi pada keputusan akhir. Di sinilah Analisis Hierarki Proses (AHP) dapat berperan. AHP adalah teknik pengambilan keputusan multi-kriteria terstruktur yang membantu dalam menetapkan bobot relatif untuk berbagai kriteria berdasarkan perbandingan berpasangan.

Makalah ini bertujuan untuk menjembatani kesenjangan antara prediksi *machine learning* dan pengambilan keputusan terstruktur dengan mengintegrasikan Random Forest dan AHP. Kami akan menggunakan *feature importance* yang diperoleh dari Random Forest sebagai dasar untuk membangun matriks perbandingan berpasangan AHP, yang kemudian digunakan untuk menghitung bobot kriteria. Pendekatan hibrida ini diharapkan dapat memberikan sistem diagnosis kesehatan mental yang lebih transparan dan dapat diinterpretasikan.

1.2 Rumusan Masalah

1. Bagaimana penerapan algoritma Random Forest dalam mendiagnosis gangguan kesehatan mental berdasarkan data pasien?
2. Bagaimana metode AHP dapat digunakan untuk menginterpretasikan feature importance dari model Random Forest dalam pengambilan keputusan?

1.3 Tujuan

1. Menerapkan algoritma Random Forest untuk menganalisis dan memprediksi gangguan kesehatan mental secara otomatis.
2. Menggunakan metode AHP untuk menyusun bobot kriteria berdasarkan feature importance guna mendukung keputusan diagnosis yang lebih terstruktur dan dapat dijelaskan.

1.4 Manfaat

1. Membantu tenaga medis atau konselor dalam melakukan diagnosis awal gangguan kesehatan mental secara lebih objektif.
2. Menyediakan sistem pendukung keputusan yang menggabungkan machine learning dan AHP untuk hasil diagnosis yang akurat dan dapat dijelaskan.
3. Meningkatkan efisiensi proses identifikasi gangguan mental berdasarkan data gejala pasien.

BAB II

TINJAUAN PUSTAKA

2.1 Random Forest dalam Diagnosis Kesehatan Mental

Beberapa penelitian telah mengeksplorasi penggunaan Random Forest untuk diagnosis gangguan mental (Nurdiansyah et al., 2025). Random Forest efektif dalam mengklasifikasikan berbagai kondisi seperti depresi, kecemasan, dan gangguan bipolar berdasarkan fitur-fitur seperti gejala, riwayat pasien, dan data demografi. Keunggulan utamanya adalah kemampuannya menangani fitur kategorikal dan numerik serta memberikan metrik *feature importance* yang dapat membantu mengidentifikasi prediktor utama.

2.2 Analisis Hierarki Proses (AHP) dalam Pengambilan Keputusan Medis

AHP, yang dikembangkan oleh Thomas Saaty, adalah metode matematis untuk mengatur dan menganalisis keputusan kompleks (Rozali et al., 2023). Ini melibatkan dekomposisi masalah menjadi hierarki, perbandingan berpasangan elemen-elemen pada setiap tingkat hierarki, dan sintesis untuk mendapatkan bobot prioritas. AHP telah diterapkan secara luas dalam pengambilan keputusan medis, misalnya untuk pemilihan terapi, evaluasi risiko, dan alokasi sumber daya, karena kemampuannya untuk mengintegrasikan penilaian subjektif para ahli dengan data objektif.

2.3 Integrasi Machine Learning dan AHP

Meskipun Random Forest dan AHP sering digunakan secara terpisah, ada potensi sinergi ketika digabungkan (Haditama Rifan, 2023). *Feature importance* dari model *machine learning* dapat memberikan masukan objektif untuk perbandingan berpasangan AHP, mengurangi subjektivitas awal dalam pembentukan matriks perbandingan. Sebaliknya, AHP dapat memberikan kerangka kerja yang lebih formal untuk memahami kontribusi relatif dari fitur-fitur yang diidentifikasi oleh model *machine learning*.

BAB III

METODOLOGI

3.1 Pengumpulan dan Pra-pemrosesan Data

Dataset yang digunakan adalah dataset diagnosa kesehatan mental Yang berisi 121 baris dan 19 kolom.

Tahapan pra-pemrosesan meliputi:

1. Membaca Data

   Langkah pertama adalah membuka dan membaca data dari file yang sudah disiapkan dalam format CSV.

1. Menangani Data yang Kosong

   Data diperiksa untuk memastikan tidak ada bagian yang kosong. Jika ditemukan, data yang kosong diisi secara otomatis menggunakan cara mengisi dari atas ke bawah dan sebaliknya. Namun, pada data ini tidak ditemukan kekosongan yang berarti.

2. Menghapus Kolom yang Tidak Diperlukan

   Kolom "Patient Number" dihapus karena tidak memiliki pengaruh terhadap proses prediksi.

4. Menentukan Hasil yang Akan Diprediksi

   Kolom "Expert Diagnose" dipilih sebagai bagian data yang ingin diprediksi, yaitu hasil diagnosis dari ahli.

5. Menyaring Data yang Jumlahnya Terlalu Sedikit

   Beberapa jenis diagnosis yang hanya muncul satu kali dihapus agar hasil pelatihan model menjadi lebih seimbang dan stabil. Setelah proses ini, tersisa 120 data dengan penyebaran diagnosis yang cukup merata, seperti Bipolar Tipe-2, Depresi, Normal, dan Bipolar Tipe-1.

6. Mengubah Data Teks Menjadi Angka

   Data yang berbentuk teks diubah menjadi angka agar bisa diproses oleh sistem.

7. Menyamakan Skala Angka dalam Data

   Nilai-nilai angka dalam data disesuaikan skalanya agar semua fitur memiliki pengaruh yang seimbang dalam proses pembelajaran model.

3.2 Pelatihan dan Evaluasi Model Random Forest

1. Membagi Data

   Setelah data dipersiapkan, data tersebut dibagi menjadi dua bagian: 80% untuk melatih sistem dan 20% untuk menguji hasilnya. Pembagian ini dilakukan secara acak namun tetap menjaga keseimbangan jumlah masing-masing jenis diagnosis di kedua bagian. Hasilnya, terdapat 96 data untuk pelatihan dan 24 data untuk pengujian.

2. Melatih Model

   Model Random Forest dilatih menggunakan data pelatihan. Model ini diatur untuk membangun 100 pohon keputusan dan membatasi kedalaman setiap pohon hingga 10 tingkat, agar hasil prediksinya tetap akurat namun tidak berlebihan.

3. Mengevaluasi Hasil Model

   Setelah model selesai dilatih, kinerjanya diuji menggunakan beberapa ukuran penilaian:

   1. Akurasi: Mengukur seberapa sering model memberikan prediksi yang benar. Dalam pengujian ini, akurasi model mencapai 83,33%.
   2. Laporan Klasifikasi: Menyediakan informasi lebih rinci tentang seberapa baik model mengenali setiap jenis diagnosis, termasuk ketepatan dan konsistensi prediksinya.
   3. Confusion Matrix: Menampilkan jumlah prediksi yang benar dan salah dalam bentuk tabel, sehingga bisa dilihat secara jelas di mana model berhasil dan di mana masih melakukan kesalahan.

3.3 Integrasi AHP Berdasarkan Feature Importance Random Forest

1. Menentukan Fitur Paling Berpengaruh

   Setelah model Random Forest selesai dilatih, bobot penting tiap fitur diambil sebagai ukuran pengaruh fitur terhadap hasil prediksi.

2. Memilih 5 Fitur Utama untuk Analisis AHP

   Dari semua fitur, lima dengan bobot terbesar dipilih untuk dianalisis lebih lanjut menggunakan metode Analytic Hierarchy Process (AHP).

3. Membuat Matriks Perbandingan Berpasangan

   Matriks perbandingan dibuat berdasarkan rasio bobot fitur-fitur tersebut. Nilai perbandingan antar fitur diterjemahkan ke dalam skala 1 sampai 9 sesuai skala Saaty pada AHP. Skala Saaty adalah sebagai berikut:

   a. 1 = Sama penting
   b. 3 = Sedikit lebih penting

c. 5 = Lebih penting
d. 7 = Sangat penting
e. 9 = Mutlak lebih penting

3.4 Perhitungan Bobot Prioritas AHP

Bobot prioritas tiap fitur dihitung menggunakan metode eigenvalue pada matriks perbandingan berpasangan. Secara matematis, bobot prioritas w adalah solusi dari persamaan:

$$A\mathbf{w} = \lambda_{\max}\mathbf{w}$$

dimana:

a. A adalah matriks perbandingan berpasangan,
b. w adalah vektor bobot prioritas,
c. λmax adalah eigenvalue terbesar dari matriks A.

Vektor eigen w tersebut kemudian dinormalisasi sehingga jumlah semua bobot sama dengan 1.

3.5 Pemeriksaan Konsistensi AHP

Untuk memastikan bahwa penilaian dalam matriks perbandingan berpasangan cukup konsisten, dihitung dua nilai berikut:

1. Indeks Konsistensi (CI), dengan rumus:

$$CI = \frac{\lambda_{\max} - n}{n - 1}$$

dimana n adalah jumlah fitur (ukuran matriks).

2. Rasio Konsistensi (CR), dengan rumus:

$$CR = \frac{CI}{RI}$$

dimana RI adalah nilai indeks acak (Random Index) yang bergantung pada ukuran matriks, misalnya:

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
RI	0	0	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49

Jika nilai CR<0.10, maka tingkat konsistensi penilaian dianggap dapat diterima.

BAB IV

HASIL DAN PEMBAHASAN

4.1 Alur Kerja Prototipe

1. Membuat Alat Prediksi Kesehatan Mental

   Langkah pertama dimulai dengan membuat sebuah alat bernama MentalHealthPredictor. Alat ini akan digunakan untuk mengatur dan menjalankan seluruh proses analisis, mulai dari memproses data hingga menghasilkan prediksi.

2. Memuat dan Menyiapkan Data

   Data yang akan digunakan diambil dari file dan dipersiapkan agar siap dianalisis. Proses ini mencakup membersihkan data, mengatur format, dan menyesuaikan nilai-nilai agar bisa digunakan oleh sistem.

3. Melatih Model Prediksi

   Setelah data siap, sistem mulai belajar dari data tersebut menggunakan metode pembelajaran mesin. Tujuannya adalah agar model bisa mengenali pola-pola yang berkaitan dengan kondisi kesehatan mental.

4. Menguji dan Mengevaluasi Model

   Model yang sudah dilatih kemudian diuji untuk mengetahui seberapa baik kemampuannya dalam memprediksi. Hasil uji ini mencakup akurasi, rincian kinerja, dan gambaran kesalahan model dalam membuat prediksi.

5. Menganalisis Faktor yang Mempengaruhi

   Sistem juga melihat faktor-faktor apa saja yang paling berpengaruh dalam hasil prediksi. Hasil analisis ini kemudian digunakan untuk membantu pengambilan keputusan dengan metode pembobotan (AHP).

6. Memprediksi Data Baru

   Terakhir, sistem bisa digunakan untuk memprediksi kondisi kesehatan mental seseorang berdasarkan data baru yang diberikan. Hasilnya berupa diagnosis yang diperkirakan beserta tingkat kepercayaannya.

4.2. Hasil Eksperimen

Berdasarkan *output* yang disediakan:

1. Performa Random Forest:
    a. Akurasi: 0.8333.

```
Evaluating models...
Random Forest: Accuracy = 0.8333
```

    b. Laporan Klasifikasi: Menunjukkan kinerja yang baik untuk kelas 'Bipolar Type-1', 'Bipolar Type-2', dan 'Depression' (dengan *precision* dan *recall* tinggi), sementara 'Normal' memiliki *recall* yang lebih rendah, menunjukkan beberapa kebingungan dengan kelas lain.

```
Classification Report:
              precision    recall  f1-score   support

Bipolar Type-1       1.00      0.83      0.91         6
Bipolar Type-2       0.75      1.00      0.86         6
    Depression       0.75      1.00      0.86         6
        Normal       1.00      0.50      0.67         6

      accuracy                           0.83        24
     macro avg       0.88      0.83      0.82        24
  weighted avg       0.88      0.83      0.82        24
```

    c. Confusion Matrix: Mengkonfirmasi bahwa model sangat baik dalam mengidentifikasi 'Bipolar Type-2' dan 'Depression' (6 dari 6 prediksi benar), tetapi ada beberapa kesalahan klasifikasi untuk 'Bipolar Type-1' dan 'Normal'.

```
Confusion Matrix:
              Bipolar Type-1  Bipolar Type-2  Depression  Normal
Bipolar Type-1              5               1           0       0
Bipolar Type-2              0               6           0       0
Depression                  0               0           6       0
Normal                      0               1           2       3
```

2. Feature Importance (Random Forest):
    a. 'Mood Swing' adalah fitur paling penting (0.249118), diikuti oleh 'Optimisim' (0.126139) dan 'Sexual Activity' (0.113825). Ini menunjukkan bahwa fluktuasi suasana hati, tingkat optimisme, dan aktivitas seksual adalah prediktor kunci diagnosis kesehatan mental dalam dataset ini.

Feature Importance (Random Forest):

	feature	rf_importance
4	Mood Swing	0.249118
16	Optimisim	0.126139
14	Sexual Activity	0.113825
5	Suicidal thoughts	0.077988
15	Concentration	0.061024
0	Sadness	0.050917
1	Euphoric	0.044147
2	Exhausted	0.039963
3	Sleep dissorder	0.038766
9	Aggressive Response	0.037282
7	Authority Respect	0.035373
11	Nervous Break-down	0.027212
13	Overthinking	0.020706
6	Anorxia	0.020648
12	Admit Mistakes	0.019911
10	Ignore & Move-On	0.019731
8	Try-Explanation	0.017250

    b. Bobot AHP (Berdasarkan Top 5 Fitur RF):

Bobot AHP (PCM Contoh Otomatis):

	feature	ahp_weight
0	Mood Swing	0.3798
1	Optimisim	0.2309
2	Sexual Activity	0.1428
3	Suicidal thoughts	0.1428
4	Concentration	0.1037

Bobot AHP ini secara kuantitatif mendukung temuan *feature importance* dari Random Forest, menggarisbawahi 'Mood Swing' sebagai faktor paling dominan dalam konteks pengambilan keputusan yang terstruktur.

a. Konsistensi AHP: CR sebesar 0.0512 menunjukkan bahwa matriks perbandingan berpasangan yang dihasilkan secara otomatis cukup konsisten, meskipun ini adalah prototipe dan idealnya perlu divalidasi oleh pakar domain.

```
Konsistensi AHP (N=5):
  Lambda Max: 5.2293, CI: 0.0573, RI: 1.12, CR: 0.0512
  CR < 0.10 umumnya dianggap konsisten.
```

b. Contoh Prediksi: Untuk contoh input yang diberikan, model memprediksi 'Bipolar Type-2' dengan kepercayaan 82.00%, yang merupakan hasil yang cukup tinggi.

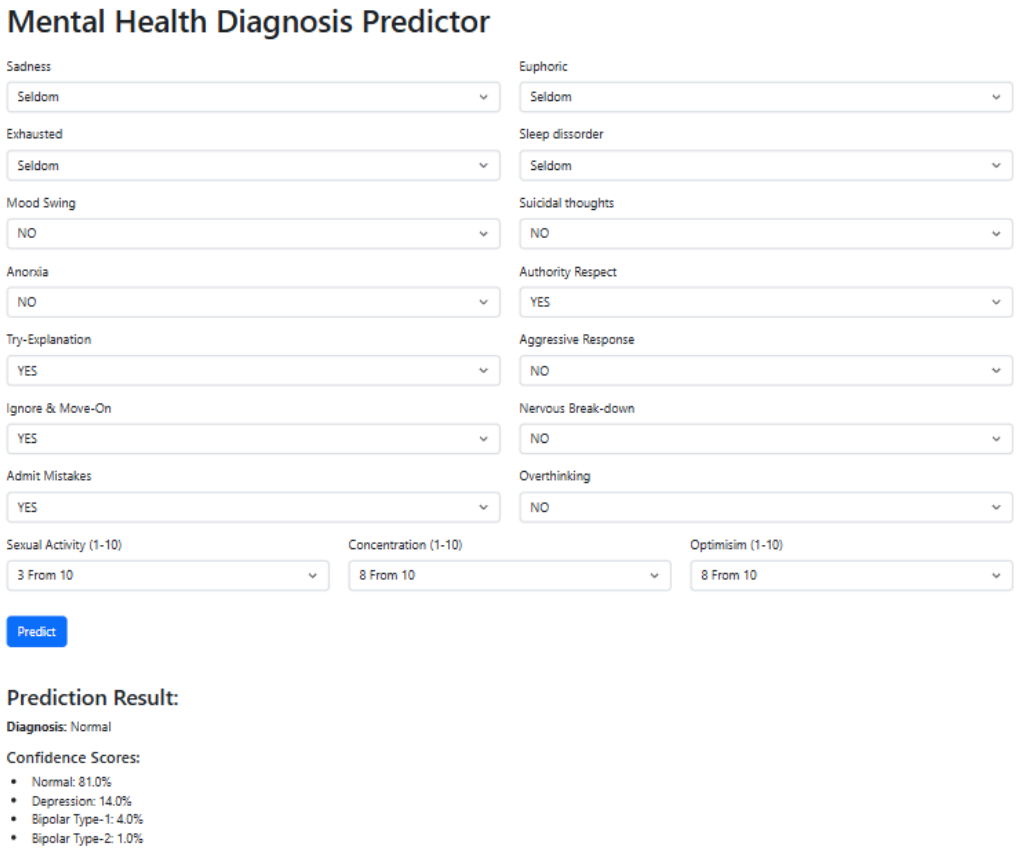
```
==================================================
CONTOH PREDIKSI UNTUK DATA BARU
==================================================
Input data untuk prediksi contoh:
  Sadness: Usually
  Euphoric: Seldom
  Exhausted: Most-Often
  Sleep dissorder: Usually
  Mood Swing: YES
  Suicidal thoughts: YES
  Anorxia: NO
  Authority Respect: NO
  Try-Explanation: YES
  Aggressive Response: NO
  Ignore & Move-On: NO
  Nervous Break-down: YES
  Admit Mistakes: NO
  Overthinking: YES
  Sexual Activity: 3 From 10
  Concentration: 2 From 10
  Optimisim: 2 From 10

Predicted Diagnosis: Bipolar Type-2
Confidence Scores (sorted):
  Bipolar Type-2: 82.00%
  Depression: 10.00%
  Bipolar Type-1: 8.00%
  Normal: 0.00%
```

4.3 Tampilan Aplikasi Web Interaktif

Untuk mendemonstrasikan kapabilitas sistem prediksi secara lebih interaktif, sebuah antarmuka pengguna berbasis web sederhana telah dikembangkan. Antarmuka ini memungkinkan pengguna (misalnya, tenaga medis atau peneliti) untuk memasukkan data gejala pasien secara manual melalui formulir web, dan kemudian menerima diagnosis yang diprediksi serta tingkat kepercayaan model secara *real-time*.

Pengembangan antarmuka web ini bertujuan untuk mensimulasikan skenario penggunaan praktis di mana data pasien baru dapat diinput dan dianalisis



4.4 Kode Program

```
class MentalHealthPredictor:
    def __init__(self):
        self.models = {}
        self.label_encoders = {}
        self.scaler = StandardScaler()
        self.feature_columns = []
        self.target_encoder = LabelEncoder()
```

```
        self.best_model = None
        self.best_model_name = None
        self.ahp_weights = None

    def load_and_preprocess_data(self, file_path):
        try:
            file_extension = os.path.splitext(file_path)[1].lower()
            if file_extension in ['.xlsx', '.xls']:
                print(f"Membaca file Excel: {file_path}")
                df = pd.read_excel(file_path)
            elif file_extension == '.csv':
                print(f"Membaca file CSV: {file_path}")
                df = pd.read_csv(file_path)
            else:
                raise ValueError(f"Format file tidak didukung: {file_extension}. Gunakan .xlsx, .xls, atau .csv")

            print(f"Data berhasil dimuat: {df.shape[0]} baris, {df.shape[1]} kolom")
            print("\nKolom yang ditemukan:", list(df.columns))

            missing_data = df.isnull().sum()
            if missing_data.any() and missing_data.sum() > 0 : # Check if there are any missing values
                print("\nMissing values ditemukan:")
                for col, count in missing_data[missing_data > 0].items():
                    print(f"  {col}: {count} missing values")
                df = df.fillna(method='ffill').fillna(method='bfill')
                print("Missing values telah diisi.")
            else:
```

```
        print("\nTidak ada missing values yang signifikan ditemukan.")

    df.columns = df.columns.str.strip()
    if 'Patient Number' in df.columns:
        df = df.drop('Patient Number', axis=1)
        print("Kolom 'Patient Number' telah dihapus.")

    target_col = "Expert Diagnose"
    if target_col not in df.columns:
        raise ValueError(f"Kolom target '{target_col}' tidak ditemukan di dataset.")

    X = df.drop(target_col, axis=1)
    y = df[target_col]

    print(f"\nDistribusi kelas dalam '{target_col}':")
    class_distribution = y.value_counts()
    for class_name, count in class_distribution.items():
        percentage = (count / len(y)) * 100
        print(f"  {class_name}: {count} ({percentage:.1f}%)")

    min_samples_per_class = 2
    classes_to_remove = class_distribution[class_distribution < min_samples_per_class].index.tolist()

    if classes_to_remove:
        print(f"\nPeringatan: Kelas dengan sampel < {min_samples_per_class} akan dihapus: {classes_to_remove}")
        mask = ~y.isin(classes_to_remove)
        X = X[mask]
```

```
            y = y[mask]
            y = y.reset_index(drop=True) # Reset index for y
            X = X.reset_index(drop=True) # Reset index for X
            print(f"Data setelah pembersihan: {len(y)} sampel")
            new_class_distribution = y.value_counts()
            print("Distribusi kelas baru:")
            for class_name, count in new_class_distribution.items():
                percentage = (count / len(y)) * 100
                print(f"  {class_name}: {count} ({percentage:.1f}%)")

        if len(y.unique()) < 2:
            print("Peringatan: Setelah pembersihan, hanya ada satu atau tidak ada kelas tersisa. Model tidak dapat dilatih.")
            return None, None, None, None

        X_encoded = X.copy()
        for col in X.columns:
            if X[col].dtype == 'object':
                le = LabelEncoder()
                X_encoded[col] = le.fit_transform(X[col].astype(str))
                self.label_encoders[col] = le

        self.target_encoder.fit(y)
        y_encoded = self.target_encoder.transform(y)

        self.feature_columns = X_encoded.columns.tolist()
        X_scaled = self.scaler.fit_transform(X_encoded)
```

```
            return X_scaled, y_encoded, X_encoded, y

        except FileNotFoundError:
            print(f"Error: File dataset tidak ditemukan di path: {file_path}")
            return None, None, None, None
        except Exception as e:
            print(f"Error dalam memuat atau memproses data: {str(e)}")
            import traceback
            traceback.print_exc()
            return None, None, None, None

    def train_models(self, X_train, y_train):
        print("\nTraining model...")
        rf = RandomForestClassifier(n_estimators=100, random_state=42, max_depth=10)
        self.models['Random Forest'] = rf
        for name, model in self.models.items():
            print(f"Training {name}...")
            model.fit(X_train, y_train)

    def evaluate_models(self, X_test, y_test):
        print("\nEvaluating models...")
        results = {}
        if not self.models:
            print("Tidak ada model yang tersedia untuk dievaluasi.")
            return results

        for name, model in self.models.items():
            y_pred = model.predict(X_test)
```

```
            accuracy = accuracy_score(y_test, y_pred)
            results[name] = accuracy
            print(f"{name}: Accuracy = {accuracy:.4f}")

        if not results:
            self.best_model_name = None
            self.best_model = None
            return results

        self.best_model_name = max(results, key=results.get)
        self.best_model = self.models[self.best_model_name]
        print(f"\nModel terbaik: {self.best_model_name} dengan akurasi
{results[self.best_model_name]:.4f}")
        return results

    def detailed_evaluation(self, X_test, y_test):
        if self.best_model is None:
            print("Tidak ada model terbaik untuk evaluasi detail.")
            return
        print(f"\nDetailed evaluation for {self.best_model_name}:")
        y_pred = self.best_model.predict(X_test)
        target_names = self.target_encoder.classes_
        print("\nClassification Report:")
        print(classification_report(y_test, y_pred, target_names=target_names,
zero_division=0))
        print("\nConfusion Matrix:")
        cm = confusion_matrix(y_test, y_pred)
        print(pd.DataFrame(cm, index=target_names, columns=target_names))
```

```
    def predict_single(self, input_data):
        if self.best_model is None:
            print("Model belum dilatih!")
            return None, None
        try:
            input_df = pd.DataFrame([input_data])
            input_df_processed = input_df.copy()
            for col in self.feature_columns:
                if col not in input_df_processed.columns:
                    print(f"Peringatan: Kolom input '{col}' tidak ada, menggunakan default (0).")
                    input_df_processed[col] = "0" # Default as string to be encoded if needed

            input_df_encoded = input_df_processed[self.feature_columns].copy()
            for col in input_df_encoded.columns:
                if col in self.label_encoders:
                    val_to_transform = input_df_encoded[col].astype(str).iloc[0]
                    try:
                        input_df_encoded[col] =
self.label_encoders[col].transform([val_to_transform])[0]
                    except ValueError:
                        print(f"Peringatan: Nilai '{val_to_transform}' untuk '{col}' tidak dikenal.
Menggunakan encoding 0.")
                        input_df_encoded[col] = 0
                else: # Assume numeric if no encoder, try to convert
                    try:
                        input_df_encoded[col] = pd.to_numeric(input_df_encoded[col])
                    except ValueError:
                        print(f"Peringatan: Gagal konversi '{col}' ke numerik. Menggunakan 0.")
                        input_df_encoded[col] = 0
```

```
            input_df_encoded[col] = pd.to_numeric(input_df_encoded[col],
errors='coerce').fillna(0)


            input_scaled = self.scaler.transform(input_df_encoded)
            prediction_encoded = self.best_model.predict(input_scaled)[0]
            prediction_proba = self.best_model.predict_proba(input_scaled)[0]
            diagnosis = self.target_encoder.inverse_transform([prediction_encoded])[0]
            confidence_scores = {self.target_encoder.classes_[i]: proba for i, proba in
enumerate(prediction_proba)}
            return diagnosis, confidence_scores
        except Exception as e:
            print(f"Error dalam prediksi tunggal: {str(e)}")
            import traceback
            traceback.print_exc()
            return None, None


    def _create_pairwise_comparison_matrix_from_rf(self, feature_names, rf_importances,
n_top_features=5):
        if not feature_names or rf_importances is None or len(rf_importances) == 0: return
None, []
        n_top_features = min(n_top_features, len(rf_importances))
        if n_top_features == 0: return None, []


        sorted_indices = np.argsort(rf_importances)[::-1][:n_top_features]
        top_feature_names = [feature_names[i] for i in sorted_indices]
        top_importances = np.maximum(rf_importances[sorted_indices], 1e-9)


        n = n_top_features
        pcm = np.ones((n, n))
```

```
        def map_ratio_to_saaty(ratio):
            if ratio >= 8: return 9
            if ratio >= 6: return 7
            if ratio >= 4: return 5
            if ratio >= 2: return 3
            return 1
        for i in range(n):
            for j in range(i, n):
                if i == j: continue
                ratio = top_importances[i] / top_importances[j]
                if ratio >= 1:
                    pcm[i, j] = float(map_ratio_to_saaty(ratio))
                    pcm[j, i] = 1.0 / pcm[i, j]
                else:
                    pcm[j, i] = float(map_ratio_to_saaty(1.0 / ratio))
                    pcm[i, j] = 1.0 / pcm[j, i]
        return pcm, top_feature_names


    def calculate_ahp_weights(self, rf_feature_names_full, rf_importances_full,
n_top_features=5):
        print("\n" + "="*60 + "\nANALISIS AHP (BERDASARKAN TOP FITUR RF - UNTUK
DEMONSTRASI)")
        print("PERHATIAN: Matriks perbandingan AHP idealnya diisi oleh pakar domain.")
        print("Matriks ini dihasilkan otomatis dari bobot RF untuk ilustrasi.\n" + "="*60)

        if rf_importances_full is None or len(rf_importances_full) == 0:
            print("Tidak ada bobot fitur RF untuk AHP.")
            return
```

```
        pcm, ahp_feature_names = self._create_pairwise_comparison_matrix_from_rf(
            rf_feature_names_full, rf_importances_full, n_top_features)
        if pcm is None or not ahp_feature_names:
            print("Tidak dapat membuat matriks perbandingan AHP.")
            return

        print("\nMatriks Perbandingan Berpasangan (PCM) AHP (Contoh Otomatis):")
        print(pd.DataFrame(pcm, index=ahp_feature_names,
columns=ahp_feature_names).to_string(float_format="%.3f"))

        try:
            eigenvalues, eigenvectors = np.linalg.eig(pcm)
            principal_eigenvalue_idx = np.argmax(np.real(eigenvalues))
            weights = np.real(eigenvectors[:, principal_eigenvalue_idx])
            weights /= np.sum(weights)
        except np.linalg.LinAlgError:
            print("Error kalkulasi Eigen. AHP tidak dapat dilanjutkan.")
            return

        self.ahp_weights = pd.DataFrame({'feature': ahp_feature_names, 'ahp_weight':
weights}
                                        ).sort_values('ahp_weight', ascending=False)
        print("\nBobot AHP (PCM Contoh Otomatis):")
        print(self.ahp_weights.to_string(float_format="%.4f"))

        n = len(ahp_feature_names)
        lambda_max = np.real(eigenvalues[principal_eigenvalue_idx])
        if n <= 1: ci, cr, ri = 0, 0, 0
        else:
```

```
        ci = (lambda_max - n) / (n - 1) if n > 1 else 0
        ri_table = {1:0, 2:0, 3:0.58, 4:0.90, 5:1.12, 6:1.24, 7:1.32, 8:1.41, 9:1.45, 10:1.49,
                    11:1.51, 12:1.48, 13:1.56, 14:1.57, 15:1.59}
        ri = ri_table.get(n, 1.59)
        cr = (ci / ri) if ri > 0 else (0 if ci == 0 else float('inf'))

    print(f"\nKonsistensi AHP (N={n}):")
    print(f"  Lambda Max: {lambda_max:.4f}, CI: {ci:.4f}, RI: {ri:.2f}, CR: {cr:.4f}")
    print("  CR < 0.10 umumnya dianggap konsisten." if cr < 0.10 else
          "  CR >= 0.10 menunjukkan potensi inkonsistensi (PCM otomatis).")

def get_feature_importance_and_ahp(self, n_top_features_for_ahp=5):
    if not hasattr(self.best_model, 'feature_importances_'):
        print("Model terbaik tidak mendukung feature importance.")
        return

    importances = self.best_model.feature_importances_
    if not self.feature_columns: # Should be set during preprocessing
        feature_names = [f"Feature_{i}" for i in range(len(importances))]
        print("Peringatan: self.feature_columns kosong, menggunakan nama generik.")
    else:
        feature_names = self.feature_columns

    rf_importance_df = pd.DataFrame({'feature': feature_names, 'rf_importance': importances}
                                    ).sort_values('rf_importance', ascending=False)
    print("\nFeature Importance (Random Forest):")
    print(rf_importance_df.to_string(float_format="%.6f"))
```

```
        self.calculate_ahp_weights(feature_names, importances, n_top_features_for_ahp)

def find_dataset_file(explicit_file_path=None):
    if explicit_file_path and os.path.exists(explicit_file_path):
        print(f"Menggunakan file yang disediakan secara eksplisit: {explicit_file_path}")
        return explicit_file_path

    print(f"File eksplisit '{explicit_file_path}' tidak ditemukan. Mencari di direktori saat ini...")
    current_dir = os.getcwd()
    possible_files = []
    for fname in os.listdir(current_dir):
        if any(kw in fname.lower() for kw in ['mental', 'dataset', 'data']) and \
           any(fname.lower().endswith(ext) for ext in ['.xlsx', '.xls', '.csv']):
            if not fname.lower().startswith('~$'): # Exclude temp files
                possible_files.append(fname)

    if not possible_files:
        print(f"Tidak ada file dataset yang cocok ditemukan di {current_dir}.")
        if not sys.stdin.isatty(): return None # Non-interactive
        user_path = input("Masukkan path lengkap file dataset atau biarkan kosong: ").strip()
        return user_path if user_path and os.path.exists(user_path) else None

    if len(possible_files) == 1:
        print(f"Menggunakan file: {possible_files[0]} secara otomatis.")
        return possible_files[0]

    print("File dataset yang ditemukan:")
```

```
    for i, f in enumerate(possible_files, 1): print(f"  {i}. {f}")
    if not sys.stdin.isatty(): return possible_files[0] # Non-interactive default

    try:
        choice = int(input(f"Pilih nomor file (1-{len(possible_files)}): ")) - 1
        return possible_files[choice] if 0 <= choice < len(possible_files) else possible_files[0]
    except (ValueError, IndexError, EOFError):
        print("Pilihan tidak valid atau tidak ada input, menggunakan file pertama.")
        return possible_files[0]

def main():
    print("="*60 + "\nSISTEM PREDIKSI DIAGNOSIS KESEHATAN MENTAL\n" + "="*60)
    predictor = MentalHealthPredictor()

    # The filename from user context, ensure it points to the CSV version if it was originally .xlsx
    # The original filename was "Dataset-Mental-Disorders excel.xlsx - Dataset-Mental-Disorders.csv"
    # and mimetype "text/csv"
    uploaded_filename = "Dataset-Mental-Disorders excel.xlsx - Dataset-Mental-Disorders.csv"

    dataset_file = find_dataset_file(explicit_file_path=uploaded_filename)
    if not dataset_file:
        print("Gagal menemukan file dataset. Program dihentikan.")
        return

    print(f"\nMenggunakan file: {dataset_file}")
```

```
    X_scaled, y_encoded, X_original, y_original = predictor.load_and_preprocess_data(dataset_file)

    if X_scaled is None or y_encoded is None or X_scaled.shape[0] == 0 or y_encoded.shape[0] == 0:
        print("Gagal memuat/memproses data atau data kosong. Program dihentikan.")
        return

    if X_scaled.shape[0] < 2 or len(np.unique(y_encoded)) < 1 : # Need at least 2 samples and 1 class
        print("Data tidak cukup untuk split/training. Program dihentikan.")
        return

    stratify_opt = y_encoded if len(np.unique(y_encoded)) > 1 and all(c >= 2 for c in pd.Series(y_encoded).value_counts()) else None
    if stratify_opt is None and len(np.unique(y_encoded)) > 1:
        print("Peringatan: Stratifikasi tidak dapat dilakukan, beberapa kelas < 2 sampel. Split tanpa stratifikasi.")

    X_train, X_test, y_train, y_test = train_test_split(
        X_scaled, y_encoded, test_size=0.2, random_state=42, stratify=stratify_opt)

    if X_train.shape[0] == 0 or X_test.shape[0] == 0:
        print("Training atau Test set kosong. Program dihentikan.")
        return

    print(f"Training set: {X_train.shape[0]} sampel, Test set: {X_test.shape[0]} sampel")

    predictor.train_models(X_train, y_train)
```

```
    predictor.evaluate_models(X_test, y_test)

    if predictor.best_model:
        predictor.detailed_evaluation(X_test, y_test)
        predictor.get_feature_importance_and_ahp(n_top_features_for_ahp=5)

        print("\n" + "="*50 + "\nCONTOH PREDIKSI UNTUK DATA BARU\n" + "="*50)
        sample_input = {'Sadness': 'Usually', 'Euphoric': 'Seldom', 'Exhausted': 'Most-Often',
                        'Sleep dissorder': 'Usually', 'Mood Swing': 'YES', 'Suicidal thoughts': 'YES',
                        'Anorxia': 'NO', 'Authority Respect': 'NO', 'Try-Explanation': 'YES',
                        'Aggressive Response': 'NO', 'Ignore & Move-On': 'NO',
                        'Nervous Break-down': 'YES', 'Admit Mistakes': 'NO', 'Overthinking': 'YES',
                        'Sexual Activity': '3 From 10', 'Concentration': '2 From 10',
                        'Optimisim': '2 From 10'}

        # Fill missing keys in sample_input from feature_columns with a default
        if X_original is not None:
            for col_name in predictor.feature_columns:
                if col_name not in sample_input:
                    default_val = X_original[col_name].mode()
                    sample_input[col_name] = default_val[0] if not default_val.empty else
"Unknown"
                    print(f"Info: Kolom '{col_name}' diisi default '{sample_input[col_name]}' untuk
prediksi contoh.")

        print("Input data untuk prediksi contoh:")
        for key, value in {k: sample_input.get(k) for k in predictor.feature_columns}.items():
            print(f"  {key}: {value}")
```

```
        diagnosis, confidence = predictor.predict_single(sample_input)
        if diagnosis and confidence:
            print(f"\nPredicted Diagnosis: {diagnosis}")
            print("Confidence Scores (sorted):")
            for cond, score in sorted(confidence.items(), key=lambda item: item[1],
reverse=True):
                print(f"  {cond}: {score*100:.2f}%")
    else:
        print("Model tidak berhasil dilatih/dipilih. Contoh prediksi dan evaluasi detail dilewati.")


    return predictor


def interactive_prediction(predictor):
    if not predictor or not predictor.best_model:
        print("\nModel tidak siap. Mode interaktif tidak dapat dimulai.")
        return
    print("\n" + "="*60 + "\nSISTEM PREDIKSI INTERAKTIF\n" + "="*60)


    maps = {
        'freq': {'1':'Seldom', '2':'Sometimes', '3':'Usually', '4':'Most-Often'},
        'yes_no': {'1':'YES', '0':'NO'},
        'rating': {str(i):f"{i} From 10" for i in range(1,11)}
    }
    q_types = { # Simplified mapping based on expected feature names
        'Sadness': 'freq', 'Euphoric': 'freq', 'Exhausted': 'freq', 'Sleep dissorder': 'freq',
        'Sexual Activity': 'rating', 'Concentration': 'rating', 'Optimisim': 'rating'
    } # Default to 'yes_no'
```

```
while True:
    print("\nMasukkan data pasien (ketik 'quit' untuk keluar):")
    input_data = {}
    quit_flag = False
    for feature in predictor.feature_columns:
        q_type_key = q_types.get(feature, 'yes_no') # Default to yes_no
        current_map = maps[q_type_key]

        options_str = ", ".join([f"{k}={v}" for k,v in current_map.items()])
        prompt = f"{feature} ({options_str}, atau 'quit'): "

        while True:
            if not sys.stdin.isatty(): print("Mode non-interaktif."); quit_flag=True; break
            response = input(prompt).strip().lower()
            if response == 'quit': quit_flag=True; break
            if response in current_map: input_data[feature] = current_map[response]; break
            # Allow direct value input if it matches a map value
            found_direct = False
            for k_map, v_map in current_map.items():
                if response == v_map.lower():
                    input_data[feature] = v_map
                    found_direct = True
                    break
            if found_direct: break
            print(f"Input tidak valid. Coba lagi.")
        if quit_flag: break
    if quit_flag: break
```

```
        print("\nMemproses...")
        diag, conf = predictor.predict_single(input_data)
        if diag and conf:
            print(f"\n{'='*30}\nHASIL PREDIKSI: {diag}\n{'='*30}")
            print("Tingkat Kepercayaan (sorted):")
            for cond, score in sorted(conf.items(), key=lambda item: item[1], reverse=True):
                print(f"  {cond}: {score*100:.1f}%")
            print("\nCatatan: Ini hanya referensi. Konsultasikan dengan profesional.")
        else: print("Gagal mendapatkan prediksi.")

        if not sys.stdin.isatty(): break # No "predict again?" in non-interactive
        if input("\nPrediksi lagi? (y/n): ").lower() != 'y': break
    print("Terima kasih!")

if __name__ == "__main__":
    try: import openpyxl
    except ImportError: print("Info: openpyxl tidak ditemukan (opsional untuk .xlsx).")

    trained_predictor = main()
    if trained_predictor and trained_predictor.best_model and sys.stdin.isatty():
        interactive_prediction(trained_predictor)
    elif trained_predictor and trained_predictor.best_model: # Non-interactive but model trained
        print("\nModel dilatih. Mode interaktif dilewati (non-interaktif).")
    else: # Model not trained
        print("\nPelatihan model gagal/dibatalkan. Mode interaktif tidak dijalankan.")

print("--- Script execution has reached the very end ---")
```

BAB V

PENUTUP

5.1 Kesimpulan

Integrasi Random Forest dan AHP dalam sistem ini menawarkan beberapa keuntungan. Pertama, Random Forest menyediakan model prediktif yang kuat dan akurat untuk diagnosis kesehatan mental. Kedua, *feature importance* dari Random Forest memberikan dasar objektif untuk memahami fitur-fitur mana yang paling relevan. Ketiga, AHP memungkinkan strukturisasi pengambilan keputusan lebih lanjut, mengkuantifikasi bobot relatif dari fitur-fitur kunci yang diidentifikasi oleh Random Forest.

Meskipun matriks perbandingan AHP dalam prototipe ini dibuat secara otomatis berdasarkan *feature importance*, tujuan utamanya adalah untuk mendemonstrasikan bagaimana bobot *machine learning* dapat diumpankan ke kerangka kerja AHP. Dalam aplikasi dunia nyata, matriks ini idealnya akan diisi oleh pakar domain (psikolog, psikiater) untuk menggabungkan pengetahuan klinis dan memastikan validitas pengambilan keputusan.

CR yang rendah menunjukkan bahwa metode otomatis untuk menghasilkan PCM dari *feature importance* dapat menghasilkan matriks yang konsisten, yang merupakan langkah menjanjikan menuju otomatisasi proses AHP. Namun, penting untuk dicatat bahwa *feature importance* Random Forest mencerminkan kontribusi fitur terhadap akurasi model, yang mungkin berbeda dari persepsi klinis atau preferensi dalam skenario AHP yang lebih kompleks.

5.2 Saran

Berdasarkan hasil penelitian, terdapat beberapa rekomendasi yang dapat diterapkan untuk pengembangan sistem prediksi kesehatan mental ke depannya. Pertama, penting untuk mengembangkan dataset yang lebih luas dengan mengumpulkan data dari berbagai sumber dan populasi. Hal ini bertujuan untuk meningkatkan akurasi serta kemampuan generalisasi model prediksi terhadap berbagai kondisi dan latar belakang individu. Kedua, sistem sebaiknya diintegrasikan dengan faktor-faktor klinis tambahan, seperti riwayat keluarga, kondisi psikososial, dan pengaruh lingkungan, agar diagnosis yang dihasilkan menjadi lebih menyeluruh dan realistis.

Selain itu, penggunaan metode hybrid juga direkomendasikan, misalnya dengan menggabungkan algoritma Random Forest dengan pendekatan seperti fuzzy AHP atau teknik deep learning. Pendekatan ini dapat membantu menangani kompleksitas serta ketidakpastian

yang sering muncul dalam diagnosis kesehatan mental. Keempat, validasi sistem oleh tenaga ahli seperti psikolog dan psikiater dalam lingkungan klinis nyata sangat diperlukan untuk memastikan bahwa hasil diagnosis yang dihasilkan benar-benar akurat dan relevan dalam praktik profesional. Terakhir, pengembangan antarmuka pengguna yang lebih interaktif dan ramah pengguna juga penting, agar sistem ini dapat diakses dan dimanfaatkan dengan mudah oleh tenaga medis atau konselor sebagai alat bantu dalam proses diagnosis awal.

DAFTAR PUSTAKA

1. Haditama Rifan, M. (2023). Analisis Dan Pembuatan Dashboard Prediksi KelulusanMahasiswa Menggunakan Metode Random Forest, NaïveBayes Dan Support Vector Machine. *Analisis Dan Pembuatan Dashboard Prediksi Kelulusan Mahasiswa Menggunakan Metode Random Forest, Naïve Bayes Dan Support Vector Machine*, 1159.
2. Nurdiansyah, N., Febriyan, F. S., Gesit, Z., & Amanta, D. (2025). Mental Health Analysis to Prevent Mental Disorders in Students Using The K-Nearest Neighbor (K-NN) Algorithm and Random Forest Algorithm Analisis Kesehatan Mental untuk Mencegah Gangguan Mental pada Mahasiswa Menggunakan Algoritma K-Nearest Neighbor (K. *MALCOM: Indonesian Journal of Machine Learning and Computer Science*, *5*(January), 1–9. https://doi.org/https://doi.org/10.57152/malcom.v5i1.1537
3. Rozali, C., Zein, A., & Farizy, S. (2023). Penerapan Analytic Hierarchy Process (Ahp) Untuk Pemilihan Penerimaan Karyawan Baru. *JITU: Jurnal Informatika Utama*, *1*, 32–36.