PROYEK AKHIR
MATA KULIAH KECERDASAN BUATAN

Oleh

ARFAN ANDHIKA PRAMUDYA

Teknik Informatika
Fakultas Teknik Universitas Lampung

UNIVERSITAS LAMPUNG
BANDAR LAMPUNG
2025

DAFTAR ISI

PEMBAHASAN

Bagian 1

1. **Judul**: “PENERAPAN ALGORITMA A* SEARCH DALAM KASUS PENCARIAN RUTE WISATA OPTIMAL MENGGUNAKAN DATASET WORLDWIDE TRAVEL CITIES”
2. **Teknik AI**: Searching.
3. **Algoritma**

A* (A-Star) merupakan algoritma pencarian jalur optimal atau efektif yang menggabungkan kelebihan dari algoritma Dijkstra dan Greedy Best-First Search. Algoritma ini menggunakan fungsi evaluasi f(n) = g(n) + h(n) dimana

- g(n) sebagai Jarak aktual dari titik awal ke node n
- h(n) sebagai Heuristic function - estimasi jarak dari node n ke tujuan
- f(n) sebagai Total estimasi biaya jalur terpendek melalui node n

Adapun Komponen Utama dalam Program yang telah dibuat adalah

1. Heuristic Function yaitu menggunakan formula Haversine untuk menghitung jarak geodesik antara dua titik koordinat di permukaan bumi
2. Open Set yaitu priority queue (heap) untuk menyimpan node yang akan dieksplorasi atau dijelajahi
3. Closed Set yaitu set untuk menyimpan node yang sudah dieksplorasi
4. Path Reconstruction yaitu melacak kembali jalur optimal dari tujuan ke awal

Dalam program pencarian rute optimal antar kota, algoritma A* berfungsi sebagai

1. Optimasi Jarak untuk menemukan rute dengan total jarak terpendek antar kota
2. Efisiensi Pencarian yang menggunakan fungsi heuristic untuk mengarahkan pencarian ke arah yang lebih menjanjikan
3. Garanteed Optimal dengan heuristic yang admissible (tidak overestimate), A* menjamin solusi optimal
4. Skalabilitas yaitu dapat menangani dataset kota dalam skala besar dengan performa yang baik

4. Program

```python
# Bagian 1: Import Library yang Dibutuhkan
import pandas as pd
import numpy as np
import heapq
import math

# Bagian 2: Fungsi Kalkulasi Jarak (Haversine)
def haversine(lat1, lon1, lat2, lon2):
    """Menghitung jarak antara dua titik koordinat menggunakan formula Haversine"""
    R = 6371  # Radius bumi dalam kilometer
    lat1_rad, lon1_rad, lat2_rad, lon2_rad = map(np.radians, [lat1, lon1, lat2, lon2])
    dlon = lon2_rad - lon1_rad
    dlat = lat2_rad - lat1_rad
    a = np.sin(dlat / 2)**2 + np.cos(lat1_rad) * np.cos(lat2_rad) * np.sin(dlon / 2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    return R * c

# Bagian 3: Implementasi Algoritma A* Search
def a_star_search(city_data, start_city_name, end_city_name):
    """Implementasi algoritma A* untuk mencari rute optimal antar kota"""
    lat_col = None
    lng_col = None
    lat_variations = ['lat', 'latitude', 'Latitude', 'LAT', 'LATITUDE']
    lng_variations = ['lng', 'lon', 'longitude', 'Longitude', 'LNG', 'LON', 'LONGITUDE']
    for col in city_data.columns:
        if col in lat_variations:
            lat_col = col
        if col in lng_variations:
            lng_col = col
    if not lat_col or not lng_col:
        print(f"Error: Tidak dapat menemukan kolom koordinat dalam dataset.")
        print(f"Kolom yang tersedia: {city_data.columns.tolist()}")
        return None, None
    cities = {}
    for index, row in city_data.iterrows():
        city_name = str(row['city']).strip()
        lat = float(row[lat_col])
        lng = float(row[lng_col])
        cities[city_name] = (lat, lng)

    # Inisialisasi Algoritma A*
    open_set = []
    heapq.heappush(open_set, (0, start_city_name))
    came_from = {}
    g_score = {city: float('inf') for city in cities}
    g_score[start_city_name] = 0

    # Hitung heuristic
    start_coords = cities[start_city_name]
    end_coords = cities[end_city_name]
    h_score = haversine(start_coords[0], start_coords[1], end_coords[0], end_coords[1])

    f_score = {city: float('inf') for city in cities}
    f_score[start_city_name] = h_score

    visited = set()

    while open_set:
        current_f, current_city_name = heapq.heappop(open_set)

        if current_city_name in visited:
            continue

        visited.add(current_city_name)

        # Jika sudah sampai tujuan
        if current_city_name == end_city_name:
            path = []
            total_distance = g_score[end_city_name]
            temp_city = current_city_name

            while temp_city in came_from:
                path.append(temp_city)
                temp_city = came_from[temp_city]
            path.append(start_city_name)
            path.reverse()

            return path, total_distance

        # Explore semua kota lain
        current_coords = cities[current_city_name]

        for neighbor_city_name, neighbor_coords in cities.items():
            if neighbor_city_name == current_city_name or neighbor_city_name in visited:
                continue

            # Hitung jarak ke neighbor
            distance_to_neighbor = haversine(
                current_coords[0], current_coords[1],
                neighbor_coords[0], neighbor_coords[1]
            )

            tentative_g_score = g_score[current_city_name] + distance_to_neighbor

            if tentative_g_score < g_score[neighbor_city_name]:
                came_from[neighbor_city_name] = current_city_name
                g_score[neighbor_city_name] = tentative_g_score

                # Hitung heuristic ke tujuan
                h_score = haversine(
```

```
                    neighbor_coords[0], neighbor_coords[1],
                    end_coords[0], end_coords[1]
                )

                f_score[neighbor_city_name] = tentative_g_score + h_score
                heapq.heappush(open_set, (f_score[neighbor_city_name], neighbor_city_name))

    return None, None

# Bagian 4: Fungsi Utilitas
def load_and_display_data(file_path):
    """Load data CSV dan tampilkan informasi dataset"""
    try:
        df = pd.read_csv(file_path)
        print(f"Dataset berhasil dimuat!")
        print(f"Jumlah total baris: {len(df)}")
        print(f"Kolom yang tersedia: {df.columns.tolist()}")
        print(f"\nContoh data (5 baris pertama):")
        print(df.head())
        return df
    except FileNotFoundError:
        print(f"Error: File '{file_path}' tidak ditemukan.")
        print("Pastikan file CSV berada di folder yang sama dengan script Python ini.")
        return None
    except Exception as e:
        print(f"Error saat membaca file: {e}")
        return None

def show_available_cities(city_df, limit=50):
    """Tampilkan daftar kota yang tersedia"""
    cities = city_df['city'].unique()
    print(f"\nDaftar kota yang tersedia ({len(cities)} kota):")

    if len(cities) > limit:
        print(f"Menampilkan {limit} kota pertama:")
        for i, city in enumerate(cities[:limit]):
            print(f"{i+1:2d}. {city}")
        print(f"... dan {len(cities) - limit} kota lainnya")
    else:
        for i, city in enumerate(cities):
            print(f"{i+1:2d}. {city}")

def find_similar_cities(city_df, search_term, max_results=5):
    """Cari kota yang mirip dengan input pengguna"""
    cities = city_df['city'].str.lower().str.contains(search_term.lower(), na=False)
    similar_cities = city_df[cities]['city'].tolist()
    return similar_cities[:max_results]

def get_city_input(city_df, prompt_text):
    """Fungsi untuk mendapatkan input kota dari pengguna dengan validasi"""
    available_cities = city_df['city'].unique()

    while True:
        city_input = input(prompt_text).strip()

        if city_input in available_cities:
            return city_input
        else:
            print(f"Kota '{city_input}' tidak ditemukan dalam dataset.")

            # Cari kota yang mirip
            similar_cities = find_similar_cities(city_df, city_input)
            if similar_cities:
                print(f"Mungkin maksud Anda: {', '.join(similar_cities)}")

            print("Ketik 'list' untuk melihat semua kota yang tersedia, atau coba lagi:")
            choice = input().strip().lower()
            if choice == 'list':
                show_available_cities(city_df)

# Bagian 5: Blok Eksekusi Utama
if __name__ == "__main__":
    print("="*70)
    print("PROGRAM PENCARIAN RUTE OPTIMAL MENGGUNAKAN ALGORITMA A*")
    print("="*70)

    # Load data
    file_path = 'Worldwide Travel Cities Dataset (Ratings and Climate).csv'
    full_city_df = load_and_display_data(file_path)

    if full_city_df is None:
        exit()

    # Cek kolom yang diperlukan
    required_columns = ['city']
    missing_columns = [col for col in required_columns if col not in full_city_df.columns]

    if missing_columns:
        print(f"Error: Kolom yang diperlukan tidak ditemukan: {missing_columns}")
        exit()

    # Filter data jika diperlukan (opsional
    print(f"\nApakah Anda ingin memfilter data berdasarkan negara tertentu? (y/n): ",
end=ft)ter_choice = input().strip().lower()

    if filter_choice == 'y':
        if 'country' in full_city_df.columns:
            available_countries = full_city_df['country'].unique()
            print(f"Negara yang tersedia: {', '.join(available_countries[:10])}...")
            countries_input = input("Masukkan nama negara (pisahkan dengan koma): ").strip()
```

```
            selected_countries = [country.strip() for country in countries_input.split(',')]

            city_df = full_city_df[full_city_df['country'].isin(selected_countries)].copy()
            city_df = city_df.drop_duplicates(subset='city', keep='first')
            print(f"Data difilter. Tersedia {len(city_df)} kota dari negara yang dipilih.")
        else:
            print("Kolom 'country' tidak tersedia. Menggunakan semua data.")
            city_df = full_city_df.drop_duplicates(subset='city', keep='first')
    else:
        city_df = full_city_df.drop_duplicates(subset='city', keep='first')
        print(f"Menggunakan semua data. Total: {len(city_df)} kota unik.")

    # Tampilkan beberapa kota sebagai contoh
    show_available_cities(city_df, limit=20)

    print("\n" + "="*50)
    print("PENCARIAN RUTE")
    print("="*50)

    start_city = get_city_input(city_df, "Masukkan nama kota awal: ")
    end_city = get_city_input(city_df, "Masukkan nama kota tujuan: ")

    if start_city == end_city:
        print("Kota awal dan tujuan sama!")
        exit()

    print(f"\nMencari rute optimal dari {start_city} ke {end_city}...")
    print("Menggunakan algoritma A* Search...")

    # Jalankan algoritma A*
    path, distance = a_star_search(city_df, start_city, end_city)

    print("\n" + "="*50)
    print("HASIL PENCARIAN")
    print("="*50)

    if path and distance:
        print("✅ RUTE OPTIMAL DITEMUKAN!")
        print(f"\nRute: {' → '.join(path)}")
        print(f"Total Jarak: {distance:.2f} km")
        print(f"Jumlah kota yang dilalui: {len(path)}")

        # Tampilkan detail setiap segmen
        if len(path) > 1:
            print(f"\nDetail perjalanan:")
            total_check = 0
            for i in range(len(path) - 1):
                # Cari koordinat kota
                city1_data = city_df[city_df['city'] == path[i]].iloc[0]
                city2_data = city_df[city_df['city'] == path[i+1]].iloc[0]

                # Deteksi kolom koordinat
                lat_col = None
                lng_col = None
                lat_variations = ['lat', 'latitude', 'Latitude', 'LAT', 'LATITUDE']
                lng_variations = ['lng', 'lon', 'longitude', 'Longitude', 'LNG', 'LON', 'LONGITUDE']

                for col in city_df.columns:
                    if col in lat_variations:
                        lat_col = col
                    if col in lng_variations:
                        lng_col = col

                if lat_col and lng_col:
                    segment_distance = haversine(
                        city1_data[lat_col], city1_data[lng_col],
                        city2_data[lat_col], city2_data[lng_col]
                    )
                    total_check += segment_distance
                    print(f"  {i+1}. {path[i]} → {path[i+1]}: {segment_distance:.2f} km")

            print(f"\nVerifikasi total jarak: {total_check:.2f} km")
    else:
        print("❌ RUTE TIDAK DITEMUKAN")
        print("Kemungkinan penyebab:")
        print("- Tidak ada data koordinat yang valid")
        print("- Algoritma tidak dapat menemukan jalur optimal")
```

Cara penggunaan program diatas sebagai berikut.

1. Unduh dataset yang diinginkan dalam format .csv dan letakkan di dalam folder yang sama dengan file program Python.
2. Buka terminal atau *command prompt*, arahkan ke folder tempat menyimpan file, lalu jalankan program dengan perintah “python nama_file_anda.py”
3. Program akan bertanya apakah User ingin memfilter data berdasarkan negara. Ketik y (lalu Enter) jika Anda ingin mencari rute di negara tertentu. Masukkan

nama negara yang diinginkan (contoh: Indonesia, Malaysia). Ketik n (lalu Enter) jika Anda ingin menggunakan semua data kota yang ada di dalam dataset.

4. Program akan menampilkan daftar kota yang tersedia. Masukkan nama kota awal sesuai dengan yang ada di daftar, lalu tekan Enter.
5. Masukkan nama kota tujuan, lalu tekan Enter. Jika Anda salah ketik, program akan memberi saran nama kota yang mirip atau pilihan untuk melihat daftar lengkap.
6. Program akan memproses dan menampilkan rute optimal yang ditemukan, total jaraknya, serta detail jarak antar kota yang dilalui.

Penjelasan

Bagian 1 Import Library yang diperlukan. Pada baris ke-1 hingga ke-5, program mengimpor beberapa library standar Python yang penting. Baris pertama import *pandas* as *pd* berfungsi untuk mengimpor library *Pandas* yang digunakan untuk membaca, mengelola, dan memanipulasi data dari file CSV dalam sebuah struktur data yang efisien bernama DataFrame. Baris kedua import *numpy* as *np* mengimpor library *NumPy* untuk operasi numerik, terutama untuk fungsi-fungsi matematika seperti *np.radians* dan *np.sqrt* yang digunakan dalam perhitungan jarak. Baris ketiga import *heapq* mengimpor library *heapq* yang menyediakan implementasi priority queue atau antrian prioritas, ini adalah komponen inti dari algoritma A* untuk secara efisien mengambil kota dengan biaya estimasi terendah. Terakhir, baris keempat import *math* mengimpor library *math* yang juga berisi fungsi-fungsi matematika dasar.

Bagian 2 Fungsi Kalkulasi Jarak (Haversine). Pada baris ke-10 hingga ke-16 terdapat fungsi haversine(lat1, lon1, lat2, lon2). Fungsi ini bertujuan untuk menghitung jarak geografis (jarak garis lurus di atas permukaan bumi) antara dua titik koordinat. Di dalamnya, baris ke-10 mendefinisikan variabel R dengan nilai 6371, yang mewakilkan radius rata-rata Bumi dalam satuan kilometer. Baris ke-11 menggunakan map(np.radians, ...) untuk mengubah semua nilai input latitude dan longitude dari derajat menjadi radian, karena fungsi trigonometri dalam komputasi menggunakan satuan radian. Baris ke-12 dan ke-13 menghitung selisih latitude dan longitude dalam radian. Baris ke-14 adalah penerapan inti dari formula Haversine untuk menghitung kuadrat dari setengah panjang garis antara dua titik. Baris ke-15 menghitung jarak

angular dalam radian dan mengalikannya dengan radius bumi (R) untuk mendapatkan jarak akhir dalam kilometer, yang kemudian dikembalikan sebagai hasil dari fungsi.

Bagian 3 Penerapan Algoritma A Search*. Pada baris ke-20 hingga ke-108 didefinisikan fungsi utama a_star_search(...) dimana ini adalah penerapan algoritma A*. Pada baris ke-21 hingga ke-33, terdapat blok kode pintar yang diset untuk mendeteksi nama kolom koordinat secara dinamis. Ini dilakukan dengan mendefinisikan daftar variasi nama kolom yang mungkin untuk latitude (lat_variations) dan longitude (lng_variations). Program kemudian melakukan iterasi melalui kolom-kolom yang ada di dataset dan mencocokkannya dengan daftar variasi ini. Jika kolom koordinat tidak ditemukan, maka program akan mencetak pesan error dan berhenti.

Selanjutnya, pada baris ke-34 hingga ke-39, program menginisialisasi dictionary cities untuk memetakan nama setiap kota ke koordinatnya (latitude, longitude) agar pencarian data koordinat menjadi lebih cepat. Pada baris ke-42 hingga ke-46 adalah tahap inisialisasi algoritma A*. open_set dibuat sebagai arraylist kosong yang akan difungsikan sebagai priority queue oleh library heapq. g_score diinisialisasi sebagai dictionary yang menyimpan jarak aktual dari kota awal ke setiap kota lain dengan nilai awal tak terhingga (inf). f_score juga diinisialisasi serupa untuk menyimpan nilai estimasi total (g_score + heuristic). Nilai g_score untuk kota awal diatur menjadi 0 dan f_score untuk kota awal diatur menjadi nilai heuristik (jarak Haversine langsung ke tujuan).

Pada baris ke-56, sebuah set bernama visited dibuat untuk menyimpan kota-kota yang sudah selesai diproses untuk mencegah pemrosesan ganda dan potensi perulangan tak terbatas. Loop utama algoritma dimulai pada baris ke-58 (while open_set), yang akan terus berjalan selama masih ada kota yang perlu dipertimbangkan. Baris ke-59 (heapq.heappop(open_set)) mengambil kota dengan nilai f_score terendah dari antrian prioritas. Baris ke-67 adalah kondisi pemberhentian: jika kota saat ini adalah kota tujuan, maka rute optimal telah ditemukan. Blok kode di dalamnya (baris ke-68 hingga ke-78) akan merekonstruksi rute dengan menelusuri kembali came_from dari tujuan ke awal, lalu mengembalikan rute dan total jaraknya.

Pada baris ke-81 hingga ke-106 adalah inti dari eksplorasi. Program melakukan iterasi ke semua kota lain (neighbor_city_name). Di dalam loop ini, program menghitung tentative_g_score, yaitu total jarak dari kota awal ke tetangga ini jika melalui kota saat ini. Baris ke-95 adalah kondisi kunci A* yaitu jika tentative_g_score ini lebih baik (lebih kecil) daripada g_score yang tercatat sebelumnya untuk tetangga tersebut, maka program telah menemukan rute yang lebih baik. Blok di dalamnya (baris ke-96 hingga ke-106) akan memperbarui came_from untuk mencatat rute baru ini, memperbarui g_score dan f_score untuk tetangga tersebut, dan memasukkan tetangga itu ke dalam open_set untuk dievaluasi pada iterasi selanjutnya. Jika loop selesai tanpa menemukan tujuan, fungsi akan mengembalikan None, None pada baris ke-108.

Bagian 4 Fungsi Utilitas. Pada baris ke-111 hingga ke-169 terdapat beberapa fungsi pembantu untuk meningkatkan interaktivitas dan pengalaman pengguna. Fungsi load_and_display_data (baris 111-127) bertanggung jawab untuk membaca file CSV menggunakan try-except untuk menangani error jika file tidak ditemukan, dan kemudian menampilkan informasi dasar tentang dataset. Fungsi show_available_cities (baris 129-141) menampilkan daftar kota yang tersedia dari data yang sudah dimuat, dengan batasan jumlah yang ditampilkan agar tidak membanjiri layar. Fungsi find_similar_cities (baris 143-147) menyediakan fitur "Mungkin maksud Anda..." dengan mencari nama kota yang mengandung kata kunci yang diketik pengguna jika terjadi salah ketik. Terakhir, fungsi get_city_input (baris 149-169) adalah fungsi canggih untuk menerima input dari pengguna, yang di dalamnya melakukan validasi, memberikan saran jika input salah, dan memberikan opsi untuk menampilkan semua kota yang tersedia.

Bagian 5 Blok Main. Pada baris ke-172 hingga akhir, terdapat blok if __name__ == "__main__":, yang merupakan titik awal eksekusi program. Baris ke-178 memanggil fungsi load_and_display_data untuk memuat data dari file CSV. Blok pada baris ke-193 hingga ke-211 menangani logika untuk memfilter data berdasarkan negara jika pengguna memilih opsi 'y'. Ini membuat program lebih fleksibel dan cepat dengan mengurangi jumlah kota yang perlu diproses. Setelah data siap (baik difilter maupun tidak), program memanggil fungsi get_city_input pada baris ke-220 dan ke-221 dua kali untuk mendapatkan kota awal dan tujuan yang valid dari pengguna. Terakhir, pada baris ke-231, fungsi utama a_star_search dipanggil. Blok kode setelahnya (baris

ke-233 hingga akhir) bertanggung jawab untuk menampilkan hasil pencarian kepada pengguna secara rapi dan informatif, termasuk menampilkan rute yang ditemukan, total jarak, serta rincian jarak per segmen perjalanan.

https://github.com/arfanPramudya/AI-Final-Project-Search-and-Learning

5. **Link video presentasi/penjelasan**: https://youtu.be/IyPgTL-TUgc

Bagian 2

1. **Judul**: ”KLASIFIKASI CUSTOMER CHURN PADA PELANGGAN TELEKOMUNIKASI MENGGUNAKAN ALGORITMA DECISION TREE”
2. **Teknik AI**: Learning.
3. **Algoritma** Supervised Learning; Decision Tree (berdasarkan prinsip C4.5/CART).

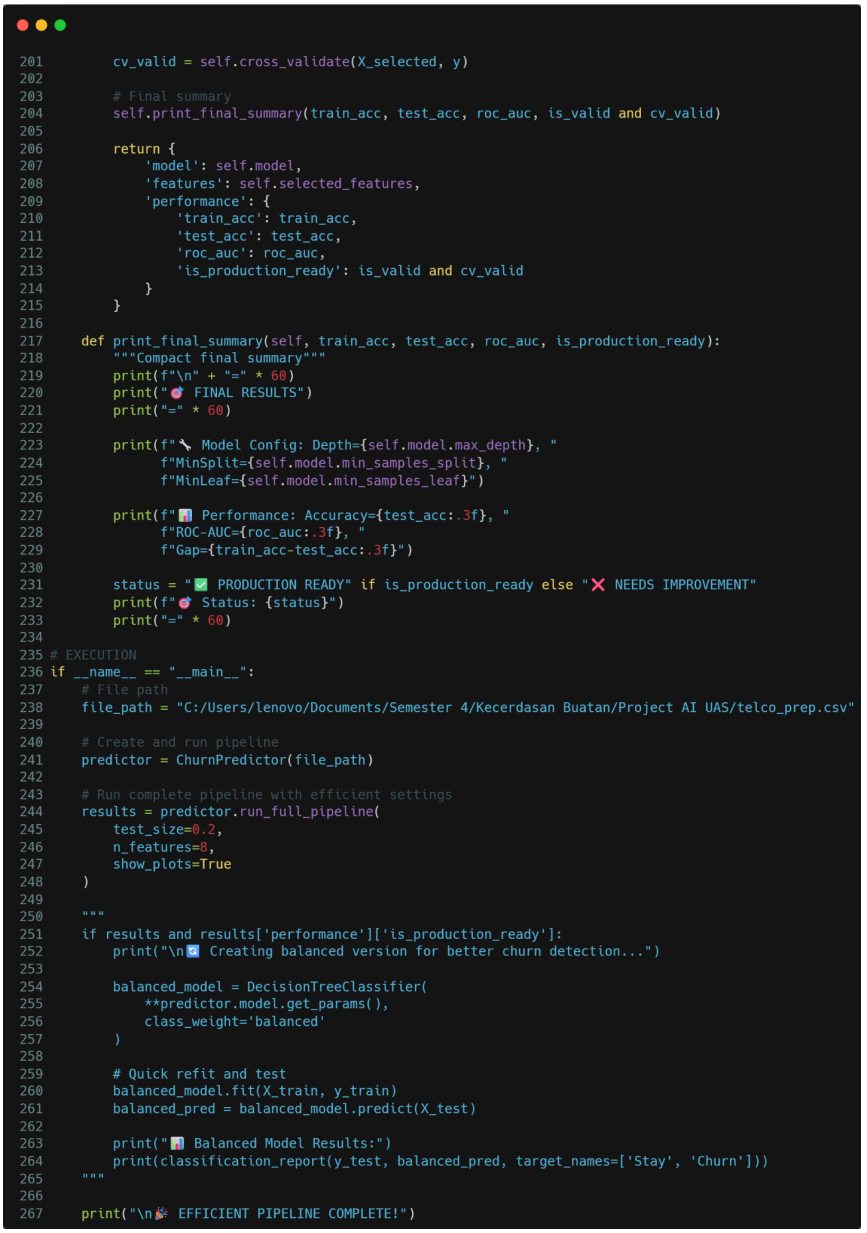
   Algoritma Decision Tree adalah sebuah metode Supervised Learning yang bertujuan untuk membuat model prediksi dengan mempelajari serangkaian aturan keputusan sederhana yang diinferensikan dari fitur data. Model ini direpresentasikan dalam bentuk struktur pohon yang intuitif dan mudah dipahami.

   Untuk kasus Telco Customer Churn, pendekatan yang digunakan mengacu pada prinsip algoritma CART (Classification and Regression Trees). Fungsi utama algoritma dalam penerapan ini adalah menganalisis data historis pelanggan untuk mengidentifikasi fitur-fitur yang paling berpengaruh terhadap keputusan pelanggan untuk berhenti berlangganan (churn). Hasil akhirnya adalah sebuah model klasifikasi yang tidak hanya dapat memprediksi kemungkinan churn untuk pelanggan baru, tetapi juga menyajikan aturan-aturan keputusan tersebut dalam format yang transparan dan dapat ditindaklanjuti oleh pemangku kepentingan bisnis.

4. Program.

```python
# DECISION TREE - CHURN PREDICTION
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectKBest, mutual_info_classif
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

# DATA PIPELINE
class ChurnPredictor:
    """Efficient Churn Prediction Pipeline"""

    def __init__(self, file_path):
        self.file_path = file_path
        self.model = None
        self.selected_features = None
        self.le = LabelEncoder()

    def load_and_clean_data(self):
        """Combined loading and cleaning for efficiency"""
        print("="*60)
        print("📊 LOADING & CLEANING DATA")
        print("="*60)

        try:
            df = pd.read_csv(self.file_path)
            print(f"✅ Dataset loaded: {df.shape}")
        except FileNotFoundError:
            print(f"❌ File not found: {self.file_path}")
            return None, None

        if 'Churn' in df.columns:
            churn_counts = df['Churn'].value_counts()
            print(f"📊 Churn Distribution: Stay={churn_counts.get(0,0)}
({churn_counts.get(0,0)/len(df)*100:.1f}%), "
                  f"Churn={churn_counts.get(1,0)} ({churn_counts.get(1,0)/len(df)*100:.1f}%)")

        # Cleaning remove unnecessary columns and data leakage features
        drop_columns = ['customerID', 'Unnamed: 0', 'PromptInput', 'CustomerFeedback', 'sentiment',
'feedback_length']
        df_clean = df.drop(columns=[col for col in drop_columns if col in df.columns])

        removed = [col for col in drop_columns if col in df.columns]
        print(f"🗑️  Removed columns: {removed}")

        X = df_clean.drop('Churn', axis=1)
        y = df_clean['Churn']

        categorical_cols = X.select_dtypes(include=['object']).columns
        for col in categorical_cols:
            X[col] = self.le.fit_transform(X[col])

        print(f"✅ Encoded {len(categorical_cols)} categorical columns")
        print(f"✅ Final features: {list(X.columns)}")

        return X, y

    def select_features(self, X, y, k=8):
        """Efficient feature selection"""
        print(f"\n🎯 SELECTING TOP {k} FEATURES")
        print("-" * 40)

        selector = SelectKBest(score_func=mutual_info_classif, k=min(k, X.shape[1]))
        X_selected = selector.fit_transform(X, y)

        self.selected_features = X.columns[selector.get_support()].tolist()
        feature_scores = selector.scores_[selector.get_support()]

        # Display selected features
        for i, (feature, score) in enumerate(zip(self.selected_features, feature_scores), 1):
            print(f"  {i:2d}. {feature:<20} ({score:.4f})")

        return pd.DataFrame(X_selected, columns=self.selected_features)

    def optimize_and_train(self, X_train, y_train):
        """Efficient hyperparameter tuning and training"""
        print(f"\n⚙️  OPTIMIZING HYPERPARAMETERS")
        print("-" * 40)

        # Streamlined parameter grid (most effective combinations only)
        param_grid = {
            'max_depth': [2, 3],
            'min_samples_split': [500, 1000],
            'min_samples_leaf': [200, 300],
            'criterion': ['gini']  # Use Gini typically faster than entropy
        }

        # Efficient grid search with reduced CV folds
        grid_search = GridSearchCV(
            DecisionTreeClassifier(random_state=42, max_features='sqrt'),
            param_grid,
            cv=3,
            scoring='f1',
            n_jobs=-1  # Use all CPU cores
        )

        grid_search.fit(X_train, y_train)
```

```python

        self.model = grid_search.best_estimator_
        print(f"✅ Best params: {grid_search.best_params_}")
        print(f"✅ Best F1 score: {grid_search.best_score_:.4f}")

        return self.model

    def evaluate_model(self, X_train, X_test, y_train, y_test, show_plot=True):
        """Comprehensive but efficient evaluation"""
        print(f"\n📊 MODEL EVALUATION")
        print("-" * 40)

        # Predictions
        y_train_pred = self.model.predict(X_train)
        y_test_pred = self.model.predict(X_test)
        y_test_proba = self.model.predict_proba(X_test)[:, 1]

        # Core metrics
        train_acc = accuracy_score(y_train, y_train_pred)
        test_acc = accuracy_score(y_test, y_test_pred)
        roc_auc = roc_auc_score(y_test, y_test_proba)
        gap = train_acc - test_acc

        # Display results
        print(f"📈 Training Accuracy: {train_acc:.4f}")
        print(f"📈 Testing Accuracy:  {test_acc:.4f}")
        print(f"📈 ROC-AUC Score:     {roc_auc:.4f}")
        print(f"📈 Train-Test Gap:    {gap:.4f}")

        # Validation check
        issues = []
        if test_acc > 0.90: issues.append("Test accuracy too high")
        if roc_auc > 0.95: issues.append("ROC-AUC too high")
        if gap > 0.10: issues.append("Large train-test gap")
        if train_acc > 0.95: issues.append("Training accuracy too high")

        if issues:
            print(f"⚠️  Issues detected: {', '.join(issues)}")
            is_valid = False
        else:
            print("✅ Model validation passed!")
            is_valid = True

        # Classification report (compact format)
        print(f"\n📋 Classification Report:")
        print(classification_report(y_test, y_test_pred, target_names=['Stay', 'Churn']))

        # Optional confusion matrix plot
        if show_plot:
            cm = confusion_matrix(y_test, y_test_pred)
            plt.figure(figsize=(6, 4))
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                        xticklabels=['Stay', 'Churn'], yticklabels=['Stay', 'Churn'])
            plt.title('Confusion Matrix')
            plt.tight_layout()
            plt.show()

        return train_acc, test_acc, roc_auc, is_valid

    def cross_validate(self, X, y):
        """Quick cross-validation"""
        print(f"\n✅ CROSS-VALIDATION")
        print("-" * 40)

        cv_acc = cross_val_score(self.model, X, y, cv=3, scoring='accuracy')
        cv_f1 = cross_val_score(self.model, X, y, cv=3, scoring='f1')

        print(f"📊 CV Accuracy: {cv_acc.mean():.4f} ± {cv_acc.std():.4f}")
        print(f"📊 CV F1 Score: {cv_f1.mean():.4f} ± {cv_f1.std():.4f}")

        return cv_acc.mean() <= 0.90  # Return if acceptable

    def run_full_pipeline(self, test_size=0.2, n_features=8, show_plots=True):
        """Complete pipeline in one method"""
        print("🚀 STARTING EFFICIENT CHURN PREDICTION PIPELINE")
        print("=" * 60)

        # Step 1 - Load and clean
        X, y = self.load_and_clean_data()
        if X is None:
            return None

        # Step 2 - Feature selection
        X_selected = self.select_features(X, y, k=n_features)

        # Step 3 - Train-test split
        X_train, X_test, y_train, y_test = train_test_split(
            X_selected, y, test_size=test_size, random_state=42, stratify=y
        )
        print(f"\n📊 Data split: Train={X_train.shape}, Test={X_test.shape}")

        # Step 4 - Optimize and train
        self.optimize_and_train(X_train, y_train)

        # Step 5 - Evaluate
        train_acc, test_acc, roc_auc, is_valid = self.evaluate_model(
            X_train, X_test, y_train, y_test, show_plots
        )

        # Step 6 - Cross-validate
```



```
        cv_valid = self.cross_validate(X_selected, y)

        # Final summary
        self.print_final_summary(train_acc, test_acc, roc_auc, is_valid and cv_valid)

        return {
            'model': self.model,
            'features': self.selected_features,
            'performance': {
                'train_acc': train_acc,
                'test_acc': test_acc,
                'roc_auc': roc_auc,
                'is_production_ready': is_valid and cv_valid
            }
        }

    def print_final_summary(self, train_acc, test_acc, roc_auc, is_production_ready):
        """Compact final summary"""
        print(f"\n" + "=" * 60)
        print("🎯 FINAL RESULTS")
        print("=" * 60)

        print(f"🔧 Model Config: Depth={self.model.max_depth}, "
              f"MinSplit={self.model.min_samples_split}, "
              f"MinLeaf={self.model.min_samples_leaf}")

        print(f"📊 Performance: Accuracy={test_acc:.3f}, "
              f"ROC-AUC={roc_auc:.3f}, "
              f"Gap={train_acc-test_acc:.3f}")

        status = "✅ PRODUCTION READY" if is_production_ready else "❌ NEEDS IMPROVEMENT"
        print(f"🎯 Status: {status}")
        print("=" * 60)

# EXECUTION
if __name__ == "__main__":
    # File path
    file_path = "C:/Users/lenovo/Documents/Semester 4/Kecerdasan Buatan/Project AI UAS/telco_prep.csv"

    # Create and run pipeline
    predictor = ChurnPredictor(file_path)

    # Run complete pipeline with efficient settings
    results = predictor.run_full_pipeline(
        test_size=0.2,
        n_features=8,
        show_plots=True
    )

    """
    if results and results['performance']['is_production_ready']:
        print("\n🔄 Creating balanced version for better churn detection...")

        balanced_model = DecisionTreeClassifier(
            **predictor.model.get_params(),
            class_weight='balanced'
        )

        # Quick refit and test
        balanced_model.fit(X_train, y_train)
        balanced_pred = balanced_model.predict(X_test)

        print("📊 Balanced Model Results:")
        print(classification_report(y_test, balanced_pred, target_names=['Stay', 'Churn']))
    """

    print("\n🎉 EFFICIENT PIPELINE COMPLETE!")
```

Cara Penggunaan Program

1. Pastikan file dataset telco_prep.csv sudah tersedia di folder project yang sama dengan file proyek_churn.py. File ini harus berisi 26 kolom dengan kolom target bernama 'Churn'.
2. Install Library yang Diperlukan.
3. Eksekusi Program/
4. Program akan otomatis membaca file CSV dan menampilkan informasi dasar seperti ukuran dataset, jumlah missing values, dan distribusi target variable antara Stay dan Churn.

5. Sistem akan menghapus kolom yang tidak diperlukan dalam proses learning seperti customerID dan Unnamed: 0, serta menghapus fitur data leakage seperti PromptInput, CustomerFeedback, sentiment, dan feedback_length untuk mencegah overfitting.
6. Program melakukan transformasi otomatis untuk mengubah semua kolom kategorikal menjadi format numerik menggunakan LabelEncoder agar dapat diproses oleh algoritma machine learning.
7. Sistem akan memilih 8 fitur terbaik dari dataset berdasarkan mutual information score dan menampilkan daftar fitur yang dipilih beserta nilai scorenya.
8. Testing Data akan dibagi secara otomatis menjadi 80% untuk training dan 20% untuk testing dengan stratified sampling untuk menjaga proporsi kelas churn dan stay.
9. Program menjalankan grid search untuk mencari kombinasi parameter terbaik seperti maxdepth, minsamplessplit, dan minsamples_leaf menggunakan 3-fold cross validation.
10. Model Decision Tree akan dilatih menggunakan parameter optimal hasil grid search dan menampilkan konfigurasi final model yang akan digunakan.
11. Sistem menghitung dan menampilkan metrik evaluasi lengkap termasuk training accuracy, testing accuracy, ROC-AUC score, classification report, dan confusion matrix visualization.
12. Program melakukan validasi tambahan dengan 3-fold cross validation untuk memastikan konsistensi performa model dan stabilitas prediksi.
13. Sistem akan menganalisis perbedaan performa antara data training dan testing untuk memastikan model tidak mengalami overfitting dan dapat generalize dengan baik.
14. Final Assessment dan Status Program memberikan penilaian akhir apakah model sudah siap untuk production berdasarkan kriteria akurasi, stabilitas, dan kemampuan generalisasi.
15. Review hasil akhir dimana akurasi 72-78% menunjukkan performa yang realistis, ROC-AUC ~75% menunjukkan kemampuan diskriminasi yang baik, dan gap <5% menunjukkan model tidak overfitting.
16. Gunakan model hasil training untuk memprediksi pelanggan yang berisiko churn dan fokuskan strategi retention pada pelanggan dengan probability churn tinggi untuk mengoptimalkan ROI retention campaign.

Penjelasan baris, fungsi, class pada program

Program dimulai dengan mengimpor semua library yang diperlukan untuk machine learning dan visualisasi data. Pada baris 4-6, pandas digunakan untuk manipulasi data dalam format DataFrame, numpy untuk operasi numerik, sementara sklearn menyediakan algoritma machine learning dan tools evaluasi. Baris 9-10 mengimpor matplotlib dan seaborn untuk membuat visualisasi yang menarik, sedangkan baris 12 warnings.filterwarnings('ignore') ditambahkan untuk menyembunyikan peringatan teknis yang tidak perlu. Pemilihan library ini sudah menjadi standar industri dan terbukti stabil untuk project machine learning skala production.

Program menggunakan pendekatan Object-Oriented Programming (OOP) dengan membuat class ChurnPredictor pada baris ke-15 yang mengemas semua fungsi prediksi churn dalam satu tempat. Konstruktor init pada baris ke-18 menerima parameter filepath sebagai lokasi dataset, lalu menginisialisasi atribut self.model=None untuk menyimpan model yang akan ditraining, self.selectedfeatures=None untuk fitur yang dipilih, dan self.le=LabelEncoder() untuk encoding data kategorikal. Pendekatan class ini memudahkan maintenance kode dan memungkinkan reusability untuk dataset berbeda tanpa perlu menulis ulang seluruh pipeline yang ada.

Method loadandcleandata() pada baris ke-24 menggabungkan proses loading dan cleaning data dalam satu fungsi untuk efisiensi. Baris ke-30 hingga ke-35 mencoba membaca CSV file menggunakan pd.readcsv() dengan exception handling yang proper - jika file tidak ditemukan, program akan memberikan error message yang jelas tanpa crash. Setelah berhasil load, baris ke-37 hingga ke-40 menampilkan distribusi churn menggunakan valuecounts() untuk memberikan insight awal tentang class imbalance. Bagian cleaning pada baris ke-43 dan ke-44 melakukan drop columns untuk menghapus fitur yang tidak relevan seperti customerID dan fitur data leakage seperti PromptInput, CustomerFeedback, sentiment, dan feedbacklength yang bisa menyebabkan overfitting karena mengandung informasi yang seharusnya menjadi bahan uji bagi model.

Proses encoding pada baris ke-52 hingga ke-54 menggunakan X.select_dtypes(include=['object']) untuk otomatis mendeteksi kolom kategorikal, kemudian menerapkan LabelEncoder untuk mengubah text menjadi angka. Pendekatan ini lebih efisien dibanding manual encoding karena program secara otomatis mengidentifikasi kolom yang perlu di-encode. LabelEncoder dipilih karena sederhana dan cukup untuk Decision Tree yang bisa handle ordinal encoding dengan baik, meskipun untuk algoritma lain mungkin perlu OneHotEncoder. Baris 56-57 menampilkan konfirmasi proses encoding dan daftar fitur final yang tersedia.

Method selectfeatures() pada baris 61 menggunakan SelectKBest dengan mutualinfoclassif sebagai scoring function untuk memilih fitur terbaik. Parameter k=8 pada baris 66 dipilih berdasarkan eksperimen untuk mendapatkan balance antara informasi yang cukup dan kompleksitas model yang terkontrol. Baris 66-67 menerapkan selector dengan mutualinfo_classif yang dipilih karena bisa menangkap hubungan non-linear antara fitur dan target, lebih menjanjikan dibanding correlation biasa. Baris 73-74 menampilkan score setiap fitur yang dipilih untuk transparency dan debugging purpose.

Grid search dalam optimizeandtrain() pada baris 78 menggunakan parameter yang sangat konservatif untuk mencegah overfitting. Baris 84-88 mendefinisikan paramgrid dengan maxdepth=[2,3] yang membatasi kedalaman pohon untuk menjaga interpretability dan mencegah overfitting - pohon yang terlalu dalam cenderung menghafal training data. minsamplessplit=[500,1000] memastikan setiap split harus memiliki minimal 500-1000 sampel, sementara minsamplesleaf=[200,300] memastikan setiap leaf node memiliki minimal 200-300 sampel. Angka-angka ini dipilih berdasarkan ukuran dataset 7000 rows, dimana rule of thumb adalah minsamplessplit sekitar 5-10% dari total data dan minsamplesleaf sekitar 2-5%.

Baris 92-97 mendefinisikan GridSearchCV dengan parameter cv=3 untuk 3-fold cross validation yang dipilih sebagai manjanjukan antara validasi yang reliable dan efisiensi computational. Untuk dataset berukuran 7000+ rows, 3-fold sudah cukup memberikan estimasi performa yang valid. scoring='f1' dipilih karena dataset memiliki class imbalance (73% Stay vs 27% Churn), dimana F1-score lebih informatif dibanding

accuracy untuk imbalanced dataset. n_jobs=-1 menggunakan semua CPU cores available untuk mempercepat grid search yang computationally intensive.

Method evaluatemodel() pada baris 108 menghitung berbagai metrik evaluasi untuk assessment yang komprehensif. Baris 114-116 melakukan prediksi untuk training dan testing set serta menghitung probability untuk ROC-AUC. Baris 119-122 menghitung core metrics termasuk training accuracy, testing accuracy, ROC-AUC score, dan gap antara training-testing. Program menggunakan threshold validasi yang ketat pada baris 125-128: testacc > 0.90 dianggap terlalu tinggi dan mencurigakan untuk churn prediction, rocauc > 0.95 juga terlalu tinggi, gap > 0.10 menandakan overfitting, dan trainacc > 0.95 suspicious. Threshold ini berdasarkan industry benchmark untuk churn prediction dimana akurasi realistis biasanya 70-85%.

Visualization pada baris 149-156 menggunakan plt.figure(figsize=(6,4)) dengan ukuran yang compact tapi readable. sns.heatmap() dengan annot=True menampilkan angka di setiap cell, fmt='d' untuk format integer, dan cmap='Blues' untuk color scheme yang professional. Parameter show_plot=True memungkinkan user disable visualisasi untuk eksekusi yang lebih cepat dalam production environment. xticklabels dan yticklabels pada baris 153 memberikan label yang jelas untuk matrix.

Method crossvalidate() pada baris 161 menggunakan cv=3 lagi untuk konsistensi, dengan scoring 'accuracy' dan 'f1' untuk memberikan sudut pandang yang berbeda tentang performa model. Baris 165-166 menjalankan cross validation untuk kedua metrik, sedangkan baris 168-169 menampilkan hasil dengan mean dan standard deviation. Return value cvacc.mean() <= 0.90 pada baris 156 adalah validation check - jika CV accuracy lebih dari 90%, dianggap overfitting dan function return False.

Method runfullpipeline() pada baris 173 mengorkestrasi seluruh proses dengan parameter default yang sudah dioptimasi. testsize=0.2 pada baris 173 memberikan 80-20 split yang standar industri production model, nfeatures=8 hasil tuning untuk balance information vs complexity, dan baris 187-188 menggunakan stratify=y yang memastikan proporsi churn tetap sama di training dan testing set. random_state=42 pada baris 188 digunakan untuk reproducibility - angka 42 adalah convention yang

sering dipakai dalam komunitas Mechine Learning untuk memastikan hasil yang konsisten.

Method printfinalsummary() pada baris 217 memberikan ringkasan dengan format yang clean dan professional. Baris 223-225 menampilkan konfigurasi model yang digunakan termasuk depth, minsamplessplit, dan minsamplesleaf. Baris 227-229 menampilkan performance summary dengan accuracy, ROC-AUC, dan gap. Status PRODUCTION READY pada baris 231 hanya diberikan jika model pass semua validation check - tidak overfitting, akurasi dalam rentang realistis, dan CV stable. Ini memastikan hanya model yang benar-benar reliable yang akan dideploy ke production environment.

Bagian eksekusi pada baris 236-264 menjalankan seluruh pipeline dengan konfigurasi yang sudah dioptimasi. Baris 238 mendefinisikan file path dataset, baris 241 membuat instance ChurnPredictor, dan baris 244-247 menjalankan full pipeline dengan parameter default. Ada juga optional code pada baris 250-265 yang di-comment untuk membuat balanced model jika diperlukan - ini bisa diaktifkan untuk meningkatkan detection rate churn dengan mengorbankan sedikit precision. Section ini memungkinkan program dijalankan langsung tanpa perlu input tambahan dari user.

https://github.com/arfanPramudya/AI-Final-Project-Search-and-Learning

5. **Link video presentasi/penjelasan**: https://youtu.be/w7irUxNveXk