**CAR RECOMMENDATION SYSTEM**

**Problem statement**

This project uses Natural Language Processing (NLP) to develop a car recommendation system. The system analyzes user input to extract vehicle preferences, such as fuel type, seating capacity, car category, cost, engine size, and mileage. Using these attributes, the program filters car data to recommend suitable models. Additionally, a machine learning model is built using a Random Forest Classifier to predict car models based on historical data. The goal is to create a seamless experience for users to inquire about car options and receive relevant suggestions by interpreting their queries using NLP-based syntax and semantic analysis.

**Tools Used**

* **Pandas:** Data manipulation and CSV data handling.
* **spaCy:** NLP for syntax and semantic parsing.
* **Streamlit:** User interface for taking inputs and displaying results.
* **Scikit-learn:** Machine learning model creation, including Random Forest Classifier and performance metrics

**Syntax Analysis**

The syntax analysis is implemented using spaCy to identify key components within the user's query, such as nouns, numbers, and phrases that signal user preferences. By tokenizing the input, the code classifies specific keywords related to fuel type, seating capacity, and vehicle category, parsing each token based on its part of speech and syntactic role. Syntax-based rules are established to distinguish the numeric and categorical values, allowing extraction of comparative terms such as "greater than" or "below." This parsing strategy enables efficient filtering by mapping the user's linguistic structure to specific car attributes.

**Semantic Analysis**

Semantic analysis in this project is conducted by identifying relationships between words in the user's input that express comparisons or preferences. The code uses predefined keywords and comparison operators (like "above," "maximum," or "equal to") to translate the user's intent into actionable filters for car attributes. By interpreting the context around terms, the system matches semantic meanings to database filters, enabling a more nuanced understanding of preferences, such as budget constraints or performance criteria. This semantic mapping allows for precise alignment of user intent with available car models, enhancing the recommendation relevance.

**Code**

import pandas as pd

import spacy

import streamlit as st

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.preprocessing import LabelEncoder

# Load the spaCy English model

nlp = spacy.load("en\_core\_web\_sm")

# Load car dataset from CSV

def load\_car\_dataset(file\_path):

    return pd.read\_csv(file\_path)

# Function to extract attributes from user input

def extract\_query\_attributes(user\_input):

    fuel\_type = None

    seating\_capacity = None

    car\_category = None

    cost = None

    engine\_size = None

    mileage = None

    # Initialize comparison operators

    comparison\_operators = {

        "greater than": "gt",

        "less than": "lt",

        "equal to": "eq",

        "greater than or equal to": "gte",

        "less than or equal to": "lte",

        "above": "gt",

        "below": "lt",

        "minimum": "gte",

        "maximum": "lte"

    }

    # Define keywords for syntax-based extraction

    fuel\_keywords = {"petrol", "diesel", "electric", "hybrid"}

    category\_keywords = {"suv", "sedan", "hatchback", "truck", "ev","mpv"}

    # Keywords for cost, engine size, and mileage

    cost\_keywords = {"cost", "lakhs","rate","costs"}

    engine\_keywords = {"engine", "cc", "kwh"}

    mileage\_keywords = {"mileage", "pkm", "kmpl","kilo meter"}

    # Use spaCy to process the user input

    doc = nlp(user\_input)

    for token in doc:

        # Extract fuel type based on keywords

        if token.text.lower() in fuel\_keywords:

            fuel\_type = token.text.title()  # Capitalize the first letter

        # Extract seating capacity if token is a number

        if token.pos\_ == "NUM":

            num = int(token.text)

            if 2 < num <= 7:  # Adjusted to limit to 5 seats

                seating\_capacity = num

        # Extract car category based on keywords

        if token.text.lower() in category\_keywords:

            car\_category = token.text.title()  # Capitalize the first letter

        # Extract cost, engine size, and mileage based on comparisons

        if token.text.lower() in cost\_keywords:

            for previous\_token in doc[max(token.i - 1, 0):token.i]:

                if previous\_token.pos\_ == "NUM":

                    cost = float(previous\_token.text)

                    for next\_token in doc[token.i + 1:]:

                        if next\_token.text.lower() in comparison\_operators:

                            comparison = comparison\_operators[next\_token.text.lower()]

                            if comparison == "lt":

                                cost = f"< {cost}"

                            elif comparison == "gt":

                                cost = f"> {cost}"

                            elif comparison == "lte":

                                cost = f"<= {cost}"

                            elif comparison == "gte":

                                cost = f">= {cost}"

                            elif comparison == "eq":

                                cost = f"= {cost}"

                            break

        # Extract engine size

        if token.text.lower() in engine\_keywords:

            for previous\_token in doc[max(token.i - 1, 0):token.i]:

                if previous\_token.pos\_ == "NUM":

                    engine\_size = float(previous\_token.text)

                    for next\_token in doc[token.i + 1:]:

                        if next\_token.text.lower() in comparison\_operators:

                            comparison = comparison\_operators[next\_token.text.lower()]

                            if comparison == "lt":

                                engine\_size = f"< {engine\_size}"

                            elif comparison == "gt":

                                engine\_size = f"> {engine\_size}"

                            elif comparison == "lte":

                                engine\_size = f"<= {engine\_size}"

                            elif comparison == "gte":

                                engine\_size = f">= {engine\_size}"

                            elif comparison == "eq":

                                engine\_size = f"= {engine\_size}"

                            break

        # Extract mileage

        if token.text.lower() in mileage\_keywords:

            for previous\_token in doc[max(token.i - 1, 0):token.i]:

                if previous\_token.pos\_ == "NUM":

                    mileage = float(previous\_token.text)

                    for next\_token in doc[token.i + 1:]:

                        if next\_token.text.lower() in comparison\_operators:

                            comparison = comparison\_operators[next\_token.text.lower()]

                            if comparison == "lt":

                                mileage = f"< {mileage}"

                            elif comparison == "gt":

                                mileage = f"> {mileage}"

                            elif comparison == "lte":

                                mileage = f"<= {mileage}"

                            elif comparison == "gte":

                                mileage = f">= {mileage}"

                            elif comparison == "eq":

                                mileage = f"= {mileage}"

                            break

    print(fuel\_type, seating\_capacity, car\_category, cost, engine\_size, mileage)

    return fuel\_type, seating\_capacity, car\_category, cost, engine\_size, mileage

# Function to recommend cars based on user input

def recommend\_car(user\_input, car\_data):

    # Step 1: Extract query attributes

    fuel\_type, seating\_capacity, car\_category, cost, engine\_size, mileage = extract\_query\_attributes(user\_input)

    print(f"\nExtracted Query Attributes: Fuel Type={fuel\_type}, Seating={seating\_capacity}, Category={car\_category}, Cost={cost}, Engine Size={engine\_size}, Mileage={mileage}")

    filtered\_cars = car\_data

    # Standardizing the column names to lowercase without spaces

    filtered\_cars.columns = filtered\_cars.columns.str.strip().str.lower()

    columns\_to\_lowercase = ['car model', 'fuel type', 'car category']

    for col in columns\_to\_lowercase:

        if col in filtered\_cars.columns:

            filtered\_cars[col] = filtered\_cars[col].str.strip().str.lower()

    print("Step 2: Standardized Column Names")

    print(filtered\_cars)

    if fuel\_type=='petrol':

        fuel\_type\_numeric=1

    elif fuel\_type=='diesel':

        fuel\_type\_numeric=2

    elif fuel\_type=='electric':

        fuel\_type\_numeric=3

    else:

        fuel\_type\_numeric=0

    if car\_category=='hatchback':

        car\_category\_numeric=1

    elif car\_category=='sedan':

        car\_category\_numeric=2

    elif car\_category=='suv':

        car\_category\_numeric=3

    elif car\_category=='mpv':

        car\_category\_numeric=4

    else:

        car\_category\_numeric=0

    # Filtering by fuel type, seating capacity, car category, cost, engine size, and mileage

    if fuel\_type and 'fuel type' in filtered\_cars.columns:

        filtered\_cars = filtered\_cars[filtered\_cars['fuel type'] == fuel\_type.lower()]

        print("Step 3: Filtered by Fuel Type")

        print(filtered\_cars)

    if seating\_capacity and 'seating capacity' in filtered\_cars.columns:

        filtered\_cars = filtered\_cars[filtered\_cars['seating capacity'] == seating\_capacity]

        print("Step 4: Filtered by Seating Capacity")

        print(filtered\_cars)

    if car\_category and 'car category' in filtered\_cars.columns:

        filtered\_cars = filtered\_cars[filtered\_cars['car category'] == car\_category.lower()]

        print("Step 5: Filtered by Car Category")

        print(filtered\_cars)

    # Filtering based on cost, engine size, and mileage

    if cost and 'cost\_lakhs' in filtered\_cars.columns:

        if '<' in str(cost):

            limit = float(cost.split('<')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['cost\_lakhs'] < limit]

        elif '>' in str(cost):

            limit = float(cost.split('>')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['cost\_lakhs'] > limit]

        elif '<=' in str(cost):

            limit = float(cost.split('<=')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['cost\_lakhs'] <= limit]

        elif '>=' in str(cost):

            limit = float(cost.split('>=')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['cost\_lakhs'] >= limit]

        elif '=' in str(cost):

            limit = float(cost.split('=')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['cost\_lakhs'] == limit]

        print("Step 6: Filtered by Cost")

        print(filtered\_cars)

    if engine\_size and 'engine\_cc\_kwh' in filtered\_cars.columns:

        if '<' in str(engine\_size):

            limit = float(engine\_size.split('<')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['engine\_cc\_kwh'] < limit]

        elif '>' in str(engine\_size):

            limit = float(engine\_size.split('>')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['engine\_cc\_kwh'] > limit]

        elif '<=' in str(engine\_size):

            limit = float(engine\_size.split('<=')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['engine\_cc\_kwh'] <= limit]

        elif '>=' in str(engine\_size):

            limit = float(engine\_size.split('>=')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['engine\_cc\_kwh'] >= limit]

        elif '=' in str(engine\_size):

            limit = float(engine\_size.split('=')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['engine\_cc\_kwh'] == limit]

        print("Step 7: Filtered by Engine Size")

        print(filtered\_cars)

    if mileage and 'milege\_pkm\_pc' in filtered\_cars.columns:

        if '<' in str(mileage):

            limit = float(mileage.split('<')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['milege\_pkm\_pc'] < limit]

        elif '>' in str(mileage):

            limit = float(mileage.split('>')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['milege\_pkm\_pc'] > limit]

        elif '<=' in str(mileage):

            limit = float(mileage.split('<=')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['milege\_pkm\_pc'] <= limit]

        elif '>=' in str(mileage):

            limit = float(mileage.split('>=')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['milege\_pkm\_pc'] >= limit]

        elif '=' in str(mileage):

            limit = float(mileage.split('=')[1].strip())

            filtered\_cars = filtered\_cars[filtered\_cars['milege\_pkm\_pc'] == limit]

        print("Step 8: Filtered by Mileage")

        print(filtered\_cars)

    if filtered\_cars.empty:

        print("No matches found.")

        return ["No matches found"]

    # Return the list of recommended car models

    recommendations = filtered\_cars['car model'].tolist()

    print("Final Recommendations:")

    print(recommendations)

    # Load your dataset

    data = pd.read\_csv('D:\\NLP Project\\prediction\_dataset\_numeric.csv')  # Replace with your dataset path

    label\_encoders = {}

    for column in ['Car Model']:  # Add other categorical columns if necessary

        le = LabelEncoder()

        data[column] = le.fit\_transform(data[column])

        label\_encoders[column] = le

    # Features and target variable

    X = data.drop(columns=['Car Model'])  # Replace with your target variable

    y = data['Car Model']  # Assuming Car Model is your target variable

    # Split the dataset into training and testing sets

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    # Initialize and train the Random Forest Classifier

    rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

    rf\_classifier.fit(X\_train, y\_train)

    # Make predictions

    y\_pred = rf\_classifier.predict(X\_test)

    fuel\_type\_numeric = fuel\_type\_numeric if fuel\_type\_numeric is not None else 0  # assuming fuel type cannot be numeric

    seating\_capacity = seating\_capacity if seating\_capacity is not None else 0

    car\_category\_numeric = car\_category\_numeric if car\_category\_numeric is not None else 0  # assuming car category cannot be numeric

    cost = cost if cost is not None else 0

    engine\_size = engine\_size if engine\_size is not None else 0

    mileage = mileage if mileage is not None else 0

    input\_data = pd.DataFrame({

        'Fuel Type': [fuel\_type\_numeric],

        'Seating Capacity': [seating\_capacity],

        'Car Category': [car\_category\_numeric],

        'Cost\_lakhs': [cost],

        'Engine\_cc\_kwh': [engine\_size],

        'Milege\_pkm\_pc': [mileage]

    })

    prediction = rf\_classifier.predict(input\_data)

    # Reverse label encoding to get original car model

    predicted\_car\_model = label\_encoders['Car Model'].inverse\_transform(prediction)

    print("As per classification model : ",predicted\_car\_model[0])

    return recommendations,predicted\_car\_model[0]

def car\_recommendation\_page():

    st.markdown("""

        <style>

        .stApp {

            background: linear-gradient(90deg, #00b4d8, #48cae4);

        }

        header {

            background-color: black;

            padding: 10px;

            text-align: center;

            color: white;

            font-size: 2em;

        }

        div.stButton > button {

            background-color: black;

            color: white;

            border: 2px solid white;

            padding: 10px;

            border-radius: 5px;

        }

        </style>

    """, unsafe\_allow\_html=True)

    st.markdown("""

        <header>Car Recommendation System</header>

    """, unsafe\_allow\_html=True)

    user\_input = st.text\_area("Describe your car preferences (e.g., 'I want a petrol SUV with 5 seats, cost less than 10 lakhs, engine less than 1500 cc, and mileage above 20 pkm')")

    # Load car dataset

    car\_data = load\_car\_dataset("D:\\NLP Project\\car\_data\_final.csv")

    if st.button("Recommend"):

        if user\_input:

            recommended\_cars,predictedone = recommend\_car(user\_input, car\_data)

            st.write("Recommended Car (using random forest classifier model) : "+predictedone)

            st.write("Recommended Cars:")

            for car in recommended\_cars:

                st.write("- " + car)

        else:

            st.warning("Please enter your car preferences.")

    # Button to go back to the home screen

    if st.button("Back to Home"):

        st.session\_state.page = "home"

def home\_page():

    st.markdown("""

        <style>

        .container {

            display: flex;

            flex-direction: column; /\* Align items vertically \*/

            align-items: center; /\* Center horizontally \*/

            height: 100vh;

            justify-content: flex-start; /\* Align items at the top \*/

            background: linear-gradient(90deg, #00b4d8, #48cae4);

        }

        .car-images {

            display: flex;

            justify-content: space-around; /\* Space images evenly \*/

            width: 100%; /\* Full width \*/

            margin-bottom: 20px; /\* Space below images \*/

        }

        .center-text {

            font-size: 2.5em;

            font-weight: bold;

            color: white;

            text-align: center;

            margin-bottom: 20px; /\* Space below the title \*/

        }

        .button {

            padding: 10px;

            background-color: black;

            color: white;

            border: none;

            border-radius: 5px;

            cursor: pointer;

            font-size: 1.2em; /\* Font size for the button \*/

        }

        </style>

    """, unsafe\_allow\_html=True)

    st.markdown('<div class="car-images">'

                '<img src="https://via.placeholder.com/150" width="150" height="150">'

                '<img src="https://via.placeholder.com/150" width="150" height="150">'

                '<img src="https://via.placeholder.com/150" width="150" height="150">'

                '<img src="https://via.placeholder.com/150" width="150" height="150">'

                '</div>', unsafe\_allow\_html=True)

    st.markdown('<div class="center-text">Car Recommendation System</div>', unsafe\_allow\_html=True)

    # Button to go to car recommendation system

    if st.button("Go to Car Recommendation System", key='home\_button'):

        st.session\_state.page = "recommendation"

def main():

    # Initialize session state

    if "page" not in st.session\_state:

        car\_recommendation\_page()

    elif st.session\_state.page == "recommendation":

        car\_recommendation\_page()

# Run the app

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output**

A screenshot of a computer

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

A group of white text on a black background

Description automatically generated

**Conclusion**

The integration of syntax and semantic analysis enables this project to interpret user queries effectively, extracting key preferences for a tailored car recommendation. Through NLP, the project successfully dissects user input and applies specific filtering criteria to suggest relevant car models. The Random Forest Classifier complements this by predicting car models based on historical data, enhancing the accuracy and scope of recommendations. Overall, this system showcases the effectiveness of NLP in transforming user input into actionable insights, demonstrating practical applications of syntax and semantic analysis for improved user experience in vehicle selection.