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)
```

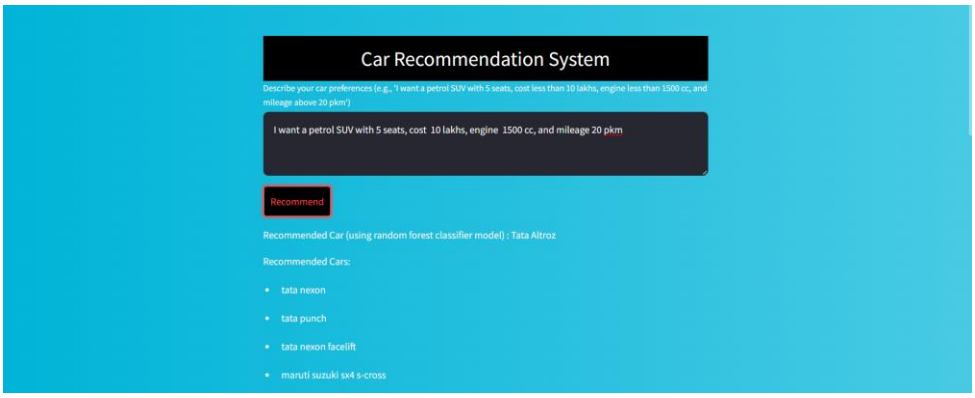
```
    # Make predictions
    y_pred = rf_classifier.predict(X_test)

    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




```
Checking segment: List me the cars with cost  equal to 10 lakhs
Tokens and their POS tags:
Token: List, POS: VERB, Tag: VB
Token: me, POS: PRON, Tag: PRP
Token: the, POS: DET, Tag: DT
Token: cars, POS: NOUN, Tag: NNS
Token: with, POS: ADP, Tag: IN
Token: cost, POS: NOUN, Tag: NN
Token:  , POS: SPACE, Tag: _SP
Token: equal, POS: ADJ, Tag: JJ
Token: to, POS: ADP, Tag: IN
Token: 10, POS: NUM, Tag: CD
Token: lakhs, POS: NOUN, Tag: NNS
```

```
Chunk: the cars, Root: cars, Root Dep: dobj, Head Text: List
Chunk: cost, Root: cost, Root Dep: pobj, Head Text: with
Chunk: 10 lakhs, Root: lakhs, Root Dep: dobj, Head Text: List
```

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.