INDVIDUAL ASSIGNMENT

COURSE NAME:-

FUNDAMENTAL OF MACHINE LEARNING

COURSE CODE: SEng4091

NAME: MHRET KIROS

DEPARTMENT: SOFTWARE

ID NO: DBU1401795

Submission date:02/06/2017

Submitted to: Derbew F.(MSC)

Table of Contents: **Page**

1. Introduction

In the rapidly evolving world of travel and tourism, making informed and personalized travel decisions is becoming increasingly important. Travelers today have access to a vast array of destination choices, and selecting the right destination can often be a daunting task. Traditional methods of choosing a destination are often based on limited criteria, such as popular tourist spots or generalized recommendations. However, to truly cater to the diverse needs and preferences of individual travelers, a more tailored approach is required.

This project aims to solve that problem by developing an **AI-powered Travel Destination Recommender System**. By leveraging the power of **Machine Learning** algorithms, specifically **Random Forest**, this system predicts the best travel destinations based on a variety of user input factors. The recommender system uses multiple user preferences such as **trip purpose**, **budget**, **accommodation preferences**, **food interests**, **travel season**, and **travel type/group** to offer personalized destination recommendations. The core idea is to build an intelligent system that not only understands travel preferences but also suggests the most suitable destination based on those preferences.

2. Problem Definition

The problem this project addresses is the challenge of recommending a suitable travel destination based on a user's preferences. By leveraging machine learning, the system can predict the best destination for a user, considering various factors like the type of trip (e.g., vacation, adventure), budget, season, accommodation type, and food preferences. The goal is to provide a recommendation system that is both scalable and highly personalized.

3. Data Source and Description

CSV File Resource

The dataset used in this project is in CSV format and contains a variety of columns representing user preferences for travel destinations. It includes both **categorical** and **numerical** features. Here's a brief overview of the key columns:

- **Destination Type**: Type of destination preferred by the user (e.g., Beach, Mountain, City, Historical Sites).
- **Trip Purpose**: Purpose of the trip (e.g., Adventure, Honeymoon, Family Vacation).
- **Budget Range**: Budget classification for the trip (e.g., Budget-Friendly, Mid-Range, Luxury).
- **Travel Season**: The season during which the user plans to travel (e.g., Winter, Spring, Summer, Autumn).
- **Travel Duration**: Duration of the trip (e.g., Weekend, A Month, Long-Term).
- **Accommodation Preference**: Type of accommodation preferred by the user (e.g., Hotel, Resort, Camping).
- **Interest**: Traveler's interests (e.g., Culture, Relaxation, Adventure, Nature).
- **Food Interests**: Dietary preference (e.g., Vegetarian, Non-Vegetarian, Vegan).
- **Travel Type/Group**: The group or type of traveler (e.g., Solo, Family, Business, Friends).

Destination Type	Trip Purpose	Budget Range	Travel Season	Travel Duration	Accommodation Preference	Interest	Food Interests	Travel Type/Group
Beach	Adventure	Budget-Friendly	Summer	Weekend	Hotel	Adventure	Vegetarian	Solo
City	Honeymoon	Luxury	Winter	A Month	Resort	Relaxation	Non-Vegetarian	Couple

Dataset Source:

The dataset used in this project is a custom-created CSV file, designed specifically to support the development and testing of the **Personalized Travel Destination Recommender** system. This dataset was generated based on project requirements to simulate user preferences, travel budgets, and destination data.

4. Exploratory Data Analysis (EDA)

EDA is a crucial part of the project to understand the dataset, identify trends, and prepare for model training.

Key Visualizations:

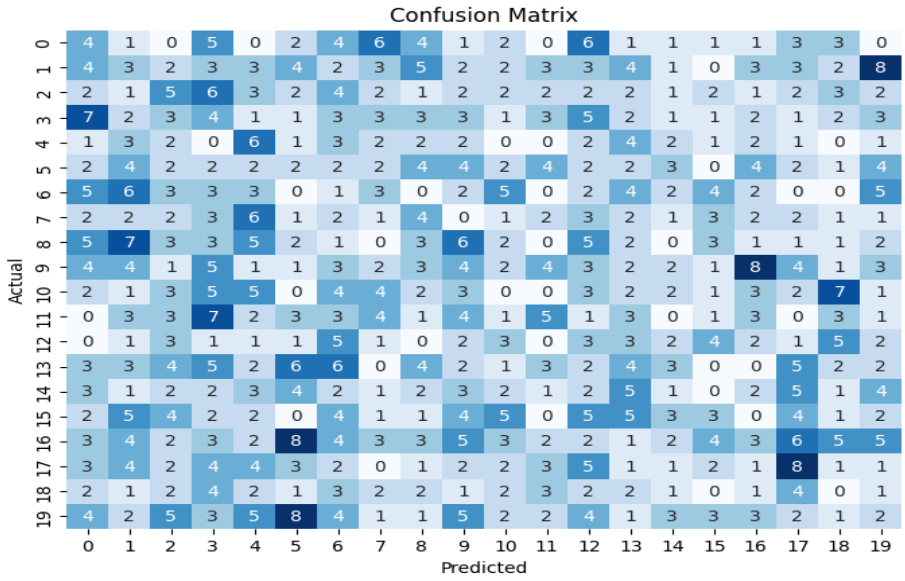
Visualizing Model Performance and Feature Importance

Confusion Matrix

The confusion matrix provides insights into the classification performance of the model by showing the number of correct and incorrect predictions for each class. It helps in evaluating how well the model distinguishes between different categories, making it easier to identify misclassifications and areas for improvement.

```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

plt.title('Confusion Matrix')
plt.show()



Feature Importance

Understanding feature importance helps identify which features contribute the most to model predictions. The bar plot below visualizes the significance of each feature in the Random Forest model. Features with higher importance scores have a stronger influence on predictions, which can be useful for feature selection and model optimization.

*feature_importances = model.feature_importances_*

plt.figure(figsize=(10,6))

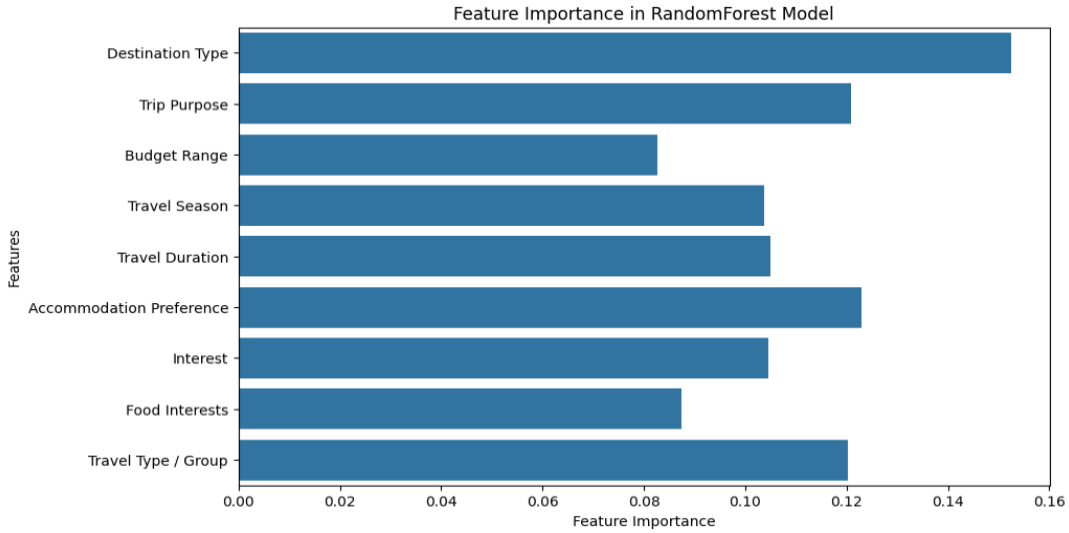
sns.barplot(x=feature_importances, y=X.columns)

plt.xlabel("Feature Importance")

plt.ylabel("Features")

plt.title("Feature Importance in RandomForest Model")

plt.show()



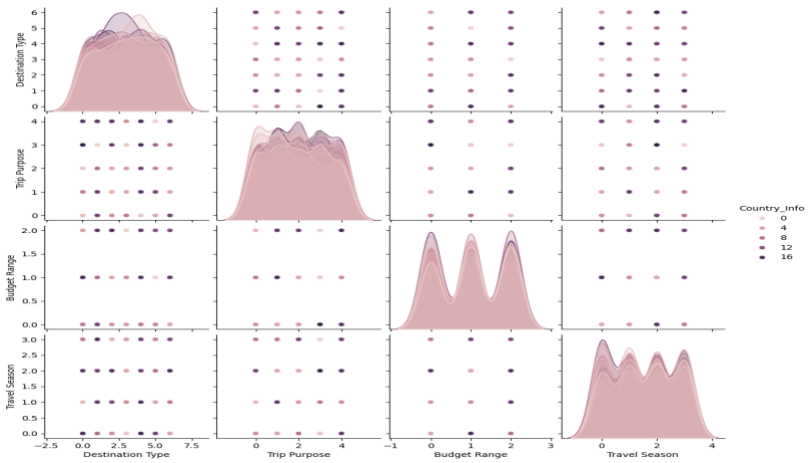
Pairplot for Feature Relationships

To explore relationships between key features and the target variable, a pairplot is generated. This visualization helps in understanding feature distributions and correlations. By analyzing these relationships, we can detect potential multicollinearity and underlying patterns in the data, which can further guide feature engineering and model refinement.

selected_features = ['Destination Type', 'Trip Purpose', 'Budget Range', 'Travel Season']

sns.pairplot(df[selected_features + ['Country_Info']], hue='Country_Info', diag_kind='kde')

plt.show()



5. **Data Preprocessing**

Data preprocessing involves several key steps to ensure the data is suitable for training the model:

1. **Handling Missing Data**: Missing values are filled with the most frequent value (mode) of the respective columns.
2. **Encoding Categorical Data**: We use **LabelEncoder** to convert categorical text data into numerical labels that the model can understand.
3. **Splitting the Dataset**: The data is split into training and testing datasets to evaluate the model's performance.

Example of Encoding and Splitting:

python
CopyEdit

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Encoding categorical features
label_encoder = LabelEncoder()
df['Destination Type'] = label_encoder.fit_transform(df['Destination Type'])

# Split data
X = df.drop('Destination', axis=1)  # Features
y = df['Destination']  # Target variable
```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

6. Model Selection and Training

For this project, we use the **Random Forest Classifier** model, which is an ensemble learning algorithm based on decision trees. It performs well on both numerical and categorical data, making it a good choice for this project.

Model Training:

python
CopyEdit
from sklearn.ensemble import RandomForestClassifier

 # Initialize the model
model = RandomForestClassifier(n_estimators=100, random_state=42)

 # Train the model
model.fit(X_train, y_train)

7. Model Evaluation

After training the model, we evaluate its performance using various metrics, such as **accuracy**, **precision**, **recall**, and **F1-score**.

Example: Evaluation Metrics

python
CopyEdit
from sklearn.metrics import classification_report, confusion_matrix

 # Generate predictions on the test set
y_pred = model.predict(X_test)

 # Classification report
print(classification_report(y_test, y_pred))

```
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt="d")
plt.title("Confusion Matrix")
plt.show()
```

8. Backend API (FastAPI)

We use **FastAPI** to create a RESTful API to serve the trained model and make predictions. This API allows users to send a request with their travel preferences and get a destination recommendation in return.

Example FastAPI Endpoint:

```
python
CopyEdit
from fastapi import FastAPI
import joblib
import pandas as pd

app = FastAPI()

# Load the trained model
model = joblib.load("travel_recommendation_model.pkl")
label_encoders = joblib.load("label_encoders.pkl")

@app.post("/predict/")
async def predict_travel_destination(input_data: TravelInput):
    try:
        # Prepare data and encode features
        encoded_data = {...}  # Encoded input data
        prediction = model.predict(pd.DataFrame([encoded_data]))
        predicted_destination =
label_encoders["Country_Info"].inverse_transform(prediction)[0]

        return {"predicted_destination": predicted_destination}
    except Exception as e:
```

```
    raise HTTPException(status_code=500, detail=str(e))
```

9. Frontend User Interface (Streamlit)

Streamlit is used to create an interactive web interface where users can input their preferences and receive recommendations. The UI is simple, making it easy for users to interact with the model.

Example of Streamlit Form:

```
python
CopyEdit
import streamlit as st
import requests

API_URL = "https://your-api-endpoint/predict/"

st.title("🌍 Travel Destination Recommendation")

# User input form
with st.form("travel_form"):
    destination_type = st.selectbox("Destination Type", ["Beach", "Mountain", "City", "Other"])
    ...
    submit_button = st.form_submit_button("Get Recommendation")

if submit_button:
    input_data = {...}  # Collect user input
    response = requests.post(API_URL, json=input_data)
    if response.status_code == 200:
        st.success(f"Recommended Destination: {response.json()['predicted_destination']}")
    else:
        st.error("Error fetching recommendation")
```

10. Testing the Model

Once the model is deployed, it's important to test both the API and the UI. The FastAPI endpoint can be tested using tools like **Postman**, and the Streamlit UI can be tested by running the app and inputting sample data. Unit tests should be written to validate individual components, such as:

- API response for correct destination predictions.
- Streamlit UI inputs and outputs.
- Model accuracy on the test set.

11. Deployment Details

The system can be deployed in multiple ways:

1. **Backend (FastAPI)**: The FastAPI app can be deployed using cloud services like **Heroku**, **AWS**, or **Render** on this time I deployed by **RENDER**
2. **Frontend (Streamlit)**: Streamlit apps can be deployed using **Streamlit Cloud** or **Heroku**. on this time I deployed by **Streamlit Cloud**

Example JSON for Deployment (FastAPI):

json
CopyEdit

```
{
   "destination_type": "Beach",
   "trip_purpose": "Adventure",
   "budget_range": "Budget-Friendly",
   "travel_season": "Summer",
   "travel_duration": "Weekend",
   "accommodation_preference": "Hotel",
   "interest": "Adventure",
   "food_interests": "Vegetarian",
   "travel_type_group": "Solo"
}
```

The above JSON structure represents a sample input that can be sent to the API to get a recommendation.

12. Conclusion

In conclusion, the **AI-powered Travel Destination Recommender System** represents a significant step forward in how technology can enhance the travel experience. The ability to provide highly personalized travel recommendations based on a wide array of user preferences not only makes travel planning more efficient but also helps individuals make better, more informed decisions. By utilizing **Random Forest** machine learning algorithms, the system is capable of handling a variety of input features and delivering accurate recommendations based on complex decision-making processes.

The project's architecture — consisting of data preprocessing, model development, API integration, and user-friendly interface — provides a robust foundation for a real-world application. The **FastAPI** backend enables quick and efficient handling of user requests, while **Streamlit** ensures a smooth and interactive user interface. Furthermore, the deployment of the system on a cloud platform ensures that it is accessible to a wide audience, paving the way for potential integrations with travel platforms and future scalability.

Links:

- **GitHub Repository**: https://github.com/MhiretKiros/Machine_Learning_Project
- **Deployment URL**: https://mhiretkiros-machine-learning-project-app-v7ztrl.streamlit.app/
- **Render: https://machine-learning-project-11-muoj.onrender.com/docs**