

```
import pandas as pd
import numpy as np
import gradio as gr
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, VotingClassifier
from sklearn.metrics import accuracy_score
```

Load datasets

```
df1 = pd.read_csv("/content/Expanded_Destinations.csv")
df2 = pd.read_csv("/content/Final_Updated_Expanded_Reviews.csv")
df3 = pd.read_csv("/content/Final_Updated_Expanded_UserHistory.csv")
df4 = pd.read_csv("/content/Final_Updated_Expanded_Users.csv")
```

Concatenate datasets

```
data = pd.concat([df1, df2, df3, df4], ignore_index=True)
for col in data.select_dtypes(include=['number']).columns:
    data[col].fillna(data[col].mean(), inplace=True)
for col in data.select_dtypes(include=['object']).columns:
    data[col].fillna(data[col].mode()[0], inplace=True)
```

⚠ <ipython-input-69-f5c876c0ce17>:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col]

```
data[col].fillna(data[col].mean(), inplace=True)
<ipython-input-69-f5c876c0ce17>:6: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting
```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col]

```
data[col].fillna(data[col].mode()[0], inplace=True)
```

Encode categorical columns

```
label_encoders = {}
categorical_cols = ['State', 'Type', 'BestTimeToVisit', 'Name']
for col in categorical_cols:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le
```

Standardize numerical columns

```
numeric_cols = ['Popularity', 'Rating']
scaler = StandardScaler()
data[numeric_cols] = scaler.fit_transform(data[numeric_cols])
```

```
X = data[['State', 'Type', 'BestTimeToVisit', 'Popularity', 'Rating']]
y = data['Name']
```

Use stratified split for balanced class distribution

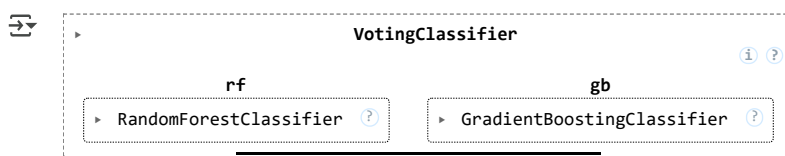
```
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_idx, test_idx in split.split(X, y):
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
```

Train ensemble models with optimized parameters

```
rf = RandomForestClassifier(n_estimators=200, max_depth=7, random_state=42)
gb = GradientBoostingClassifier(n_estimators=200, learning_rate=0.05, max_depth=4, random_state=42)
ensemble_model = VotingClassifier(estimators=[('rf', rf), ('gb', gb)], voting='soft')
```

Fit model

```
ensemble_model.fit(X_train, y_train)
```



```

# Evaluate model
y_pred = ensemble_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

🔄 Accuracy: 0.75

# Manual Recommendations
manual_recommendations = {
    "Goa": {"Beach": ["Calangute Beach", "Baga Beach"], "Historical": ["Basilica of Bom Jesus"], "History": ["Fort Kochi", "Bekal Fort"]},
    "Rajasthan": {"Desert": ["Thar Desert", "Jaisalmer Fort"], "Historical": ["Amber Fort", "City Palace", "Hawa Mahal", "Jantar Mantar"]},
    "Kerala": {"Beach": ["Varkala Beach", "Kovalam Beach"], "Nature": ["Munnar", "Alleppey Backwaters"]},
    "Uttar Pradesh": {"Adventure": ["Dudhwa National Park", "Vindhyachal"], "History": ["Chunar Fort"], "Nature": ["Dudhwa National Park"]},
    "Jammu and Kashmir": {"Adventure": ["Trekking and Hiking"], "Historical": ["Hari Parbat Fort", "Amar Mahal Palace"]}
}

# Gradio interface function
def recommend_places(state, place_type, best_time):
    # Check if all selections are None
    if state == "None" and place_type == "None" and best_time == "None":
        return "Please select at least one option to get recommendations."

    # Convert input text to encoded values
    if state == "None":
        state_encoded = None
    elif state in label_encoders['State'].classes_:
        state_encoded = label_encoders['State'].transform([state])[0]
    else:
        return "Invalid state selection."

    if place_type == "None":
        type_encoded = None
    elif place_type in label_encoders['Type'].classes_:
        type_encoded = label_encoders['Type'].transform([place_type])[0]
    else:
        return "Invalid place type selection."

    if best_time == "None":
        best_time_encoded = None
    elif best_time in label_encoders['BestTimeToVisit'].classes_:
        best_time_encoded = label_encoders['BestTimeToVisit'].transform([best_time])[0]
    else:
        return "Invalid best time selection."

    # Filter based on given inputs
    filtered_data = data.copy()
    if state_encoded is not None:
        filtered_data = filtered_data[filtered_data['State'] == state_encoded]
    if type_encoded is not None:
        filtered_data = filtered_data[filtered_data['Type'] == type_encoded]
    if best_time_encoded is not None:
        filtered_data = filtered_data[filtered_data['BestTimeToVisit'] == best_time_encoded]

    # If no recommendations found, use manual fallback
    if filtered_data.empty:
        if state in manual_recommendations and place_type in manual_recommendations[state]:
            return manual_recommendations[state][place_type]
        else:
            return "No recommendations found. Try different selections."


    # Decode the recommended place names
    recommendations = label_encoders['Name'].inverse_transform(filtered_data['Name'].unique())

    return recommendations if len(recommendations) > 0 else "No recommendations found."

# Gradio UI
demo = gr.Interface(
    fn=recommend_places,
    inputs=[
        gr.Dropdown(choices=["None"] + list(label_encoders['State'].inverse_transform(range(len(label_encoders['State'].classes_)))), label='State'),
        gr.Dropdown(choices=["None"] + list(label_encoders['Type'].inverse_transform(range(len(label_encoders['Type'].classes_)))), label='Place Type'),
        gr.Dropdown(choices=["None"] + ["Nov-Feb", "Mar-Jun", "Jul-Oct"], label='Best Time to Visit')
    ],
    outputs="text",
    title="Travel Destination Recommender"
)

# Launch Gradio
demo.launch()

```

 Running Gradio in a Colab notebook requires sharing enabled. Automatically setting `share=True` (you can turn this off by setting `s

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()

* Running on public URL: <https://45bc687151bf690924.gradio.live>

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the worki

Travel Destination Recommender

State

None ▼

Type

None ▼

Best Time to Visit

None ▼

output

Flag

Clear

Submit

Use via API  · Built with Gradio  · Settings 