

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import NearestNeighbors
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score

import kagglehub

path = kagglehub.dataset_download("amanmehra23/travel-recommendation-dataset")

print("Path to dataset files:", path)

↗ Path to dataset files: /kaggle/input/travel-recommendation-dataset
```

```
destinations_df = pd.read_csv(f"{path}/Expanded_Destinations.csv")
reviews_df = pd.read_csv(f"{path}/Final_Updated_Expanded_Reviews.csv")
userhistory_df = pd.read_csv(f"{path}/Final_Updated_Expanded_UserHistory.csv")
users_df = pd.read_csv(f"{path}/Final_Updated_Expanded_Users.csv")
```

```
destinations_df.head()
```

↗

	DestinationID	Name	State	Type	Popularity	BestTimeToVisit
0	1	Taj Mahal	Uttar Pradesh	Historical	8.691906	Nov-Feb
1	2	Goa Beaches	Goa	Beach	8.605032	Nov-Mar
2	3	Jaipur City	Rajasthan	City	9.225372	Oct-Mar
3	4	Kerala Backwaters	Kerala	Nature	7.977386	Sep-Mar
4	5	Leh Ladakh	Jammu and Kashmir	Adventure	8.399822	Apr-Jun

```
reviews_df.head()
```

↗

	ReviewID	DestinationID	UserID	Rating	ReviewText
0	1	178	327	2	Incredible monument!
1	2	411	783	1	Loved the beaches!
2	3	927	12	2	A historical wonder
3	4	358	959	3	Incredible monument!
4	5	989	353	2	Loved the beaches!

```
userhistory_df.head()
```

↗

	HistoryID	UserID	DestinationID	VisitDate	ExperienceRating
0	1	525	760	2024-01-01	3
1	2	184	532	2024-02-15	5
2	3	897	786	2024-03-20	2
3	4	470	660	2024-01-01	1
4	5	989	389	2024-02-15	4

```
users_df.head()
```



	UserID	Name	Email	Preferences	Gender	NumberOfAdults	NumberOfChildren
0	1	Kavya	kavya@example.com	Beaches, Historical	Female	1	0
1	2	Rohan	rohan@example.com	Nature, Adventure	Male	2	2
2	3	Kavya	kavya@example.com	City, Historical	Female	2	0
3	4	Anika	anika@example.com	Beaches, Historical	Female	1	0
4	5	Tanvi	tanvi@example.com	Nature, Adventure	Female	2	2

```
total_users = len(users_df)
print(f"Total number of users: {total_users}")
```

```
total_destinations = len(destinations_df)
print(f"Total number of destinations: {total_destinations}")
```

```
destination_types = destinations_df['Type'].unique()
print(f"Destination types: {destination_types}")
```

```
states = destinations_df['State'].unique()
print(f"States: {states}")
```



```
Total number of users: 999
Total number of destinations: 1000
Destination types: ['Historical' 'Beach' 'City' 'Nature' 'Adventure']
States: ['Uttar Pradesh' 'Goa' 'Rajasthan' 'Kerala' 'Jammu and Kashmir']
```

```
def dataset_info(df, name):
    print(f"\n {name} Dataset:")
    print(df.info())
    print(df.describe())
    print("Missing values:", df.isnull().sum())
    print("Duplicates:", df.duplicated().sum())
```

```
dataset_info(destinations_df.head(), "Destinations")
```



```
Destinations Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 6 columns):
#   Column          Non-Null Count  Dtype
---  -
0   DestinationID    5 non-null     int64
1   Name             5 non-null     object
2   State            5 non-null     object
3   Type             5 non-null     object
4   Popularity       5 non-null     float64
5   BestTimeToVisit  5 non-null     object
dtypes: float64(1), int64(1), object(4)
memory usage: 372.0+ bytes
None
      DestinationID  Popularity
count      5.000000    5.000000
mean        3.000000    8.579904
std         1.581139    0.454220
min         1.000000    7.977386
25%         2.000000    8.399822
50%         3.000000    8.605032
75%         4.000000    8.691906
max         5.000000    9.225372
Missing values: DestinationID    0
Name                             0
State                           0
Type                            0
Popularity                      0
BestTimeToVisit                 0
dtype: int64
Duplicates: 0
```

```
dataset_info(users_df.head(), "Users")
```



```
Users Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 7 columns):
```

```

#   Column          Non-Null Count  Dtype
---  -
0   UserID          5 non-null       int64
1   Name            5 non-null       object
2   Email           5 non-null       object
3   Preferences      5 non-null       object
4   Gender          5 non-null       object
5   NumberOfAdults   5 non-null       int64
6   NumberOfChildren 5 non-null       int64
dtypes: int64(3), object(4)
memory usage: 412.0+ bytes
None

```

	UserID	NumberOfAdults	NumberOfChildren
count	5.000000	5.000000	5.000000
mean	3.000000	1.600000	0.800000
std	1.581139	0.547723	1.095445
min	1.000000	1.000000	0.000000
25%	2.000000	1.000000	0.000000
50%	3.000000	2.000000	0.000000
75%	4.000000	2.000000	2.000000
max	5.000000	2.000000	2.000000

```

Missing values: UserID      0
Name                        0
Email                      0
Preferences                0
Gender                    0
NumberOfAdults             0
NumberOfChildren           0
dtype: int64
Duplicates: 0

```

dataset_info(reviews_df.head(), "Reviews")



```

Reviews Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   ReviewID        5 non-null       int64
1   DestinationID    5 non-null       int64
2   UserID          5 non-null       int64
3   Rating          5 non-null       int64
4   ReviewText       5 non-null       object
dtypes: int64(4), object(1)
memory usage: 332.0+ bytes
None

```

	ReviewID	DestinationID	UserID	Rating
count	5.000000	5.000000	5.000000	5.000000
mean	3.000000	572.600000	486.800000	2.000000
std	1.581139	362.927403	380.651021	0.707107
min	1.000000	178.000000	12.000000	1.000000
25%	2.000000	358.000000	327.000000	2.000000
50%	3.000000	411.000000	353.000000	2.000000
75%	4.000000	927.000000	783.000000	2.000000
max	5.000000	989.000000	959.000000	3.000000

```

Missing values: ReviewID      0
DestinationID    0
UserID           0
Rating           0
ReviewText       0
dtype: int64
Duplicates: 0

```

dataset_info(userhistory_df.head(), "User History")



```

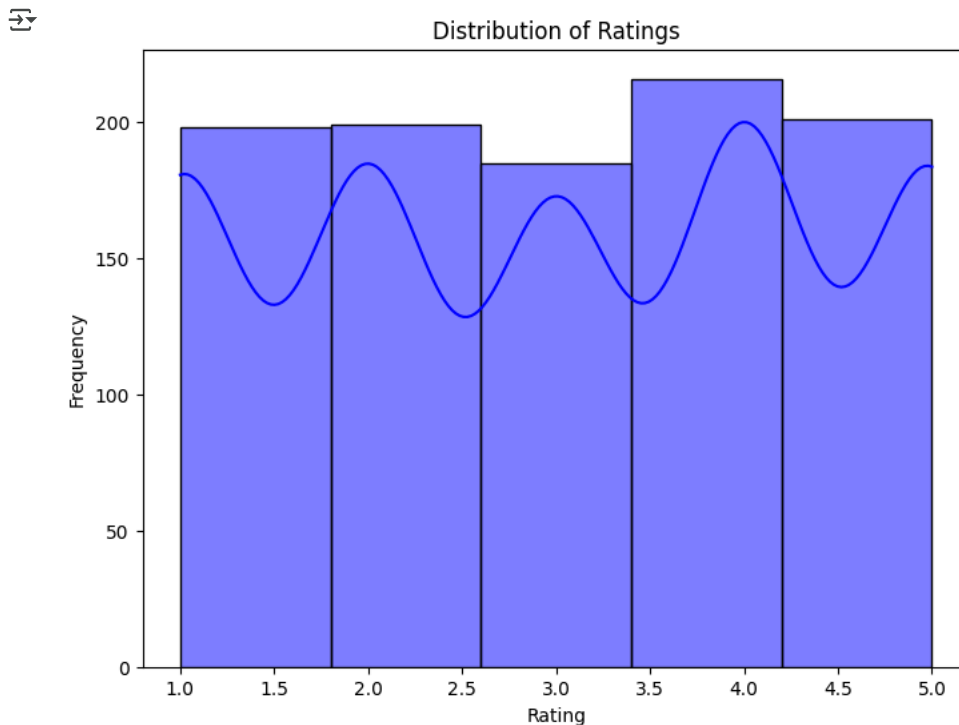
User History Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   HistoryID        5 non-null       int64
1   UserID          5 non-null       int64
2   DestinationID    5 non-null       int64
3   VisitDate        5 non-null       object
4   ExperienceRating  5 non-null       int64
dtypes: int64(4), object(1)
memory usage: 332.0+ bytes
None

```

	HistoryID	UserID	DestinationID	ExperienceRating
count	5.000000	5.000000	5.000000	5.000000
mean	3.000000	613.000000	625.400000	3.000000
std	1.581139	329.49431	165.616424	1.581139
min	1.000000	184.000000	389.000000	1.000000
25%	2.000000	470.000000	532.000000	2.000000
50%	3.000000	525.000000	660.000000	3.000000
75%	4.000000	897.000000	760.000000	4.000000
max	5.000000	989.000000	786.000000	5.000000

Missing values: HistoryID 0
 UserID 0
 DestinationID 0
 VisitDate 0
 ExperienceRating 0
 dtype: int64
 Duplicates: 0

```
# Check distribution of ratings
plt.figure(figsize=(8, 6))
sns.histplot(reviews_df['Rating'], bins=5, kde=True, color='blue')
plt.title('Distribution of Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```



```
# Merge datasets

reviews_destinations = pd.merge(reviews_df, destinations_df, on='DestinationID', how='inner')

reviews_destinations_userhistory = pd.merge(reviews_destinations, userhistory_df, on='UserID', how='inner')

df = pd.merge(reviews_destinations_userhistory, users_df, on='UserID', how='inner')

df
```

	ReviewID	DestinationID_x	UserID	Rating	ReviewText	Name_x	State	Type	Popularity	BestTimeToVisit	HistoryID	Destin
0	1	178	327	2	Incredible monument!	Jaipur City	Rajasthan	City	8.544352	Oct-Mar	79	
1	2	411	783	1	Loved the beaches!	Taj Mahal	Uttar Pradesh	Historical	8.284127	Nov-Feb	834	
2	4	358	959	3	Incredible monument!	Jaipur City	Rajasthan	City	7.738761	Oct-Mar	998	
3	5	989	353	2	Loved the beaches!	Kerala Backwaters	Kerala	Nature	8.208088	Sep-Mar	202	
4	6	473	408	4	A historical wonder	Jaipur City	Rajasthan	City	8.138558	Oct-Mar	331	
...
988	991	701	850	3	Incredible monument!	Taj Mahal	Uttar Pradesh	Historical	8.814029	Nov-Feb	138	
989	991	701	850	3	Incredible monument!	Taj Mahal	Uttar Pradesh	Historical	8.814029	Nov-Feb	643	
990	995	231	346	5	Loved the beaches!	Taj Mahal	Uttar Pradesh	Historical	7.788256	Nov-Feb	454	
991	995	231	346	5	Loved the beaches!	Taj Mahal	Uttar Pradesh	Historical	7.788256	Nov-Feb	556	
992	997	823	858	5	Incredible monument!	Jaipur City	Rajasthan	City	8.501225	Oct-Mar	423	

993 rows × 20 columns

df.shape


(993, 20)

df.to_csv("merge.csv", index=False)

df.duplicated().sum()

np.int64(0)


df.isnull().sum()



	0
ReviewID	0
DestinationID_x	0
UserID	0
Rating	0
ReviewText	0
Name_x	0
State	0
Type	0
Popularity	0
BestTimeToVisit	0
HistoryID	0
DestinationID_y	0
VisitDate	0
ExperienceRating	0
Name_y	0
Email	0
Preferences	0
Gender	0
NumberOfAdults	0
NumberOfChildren	0

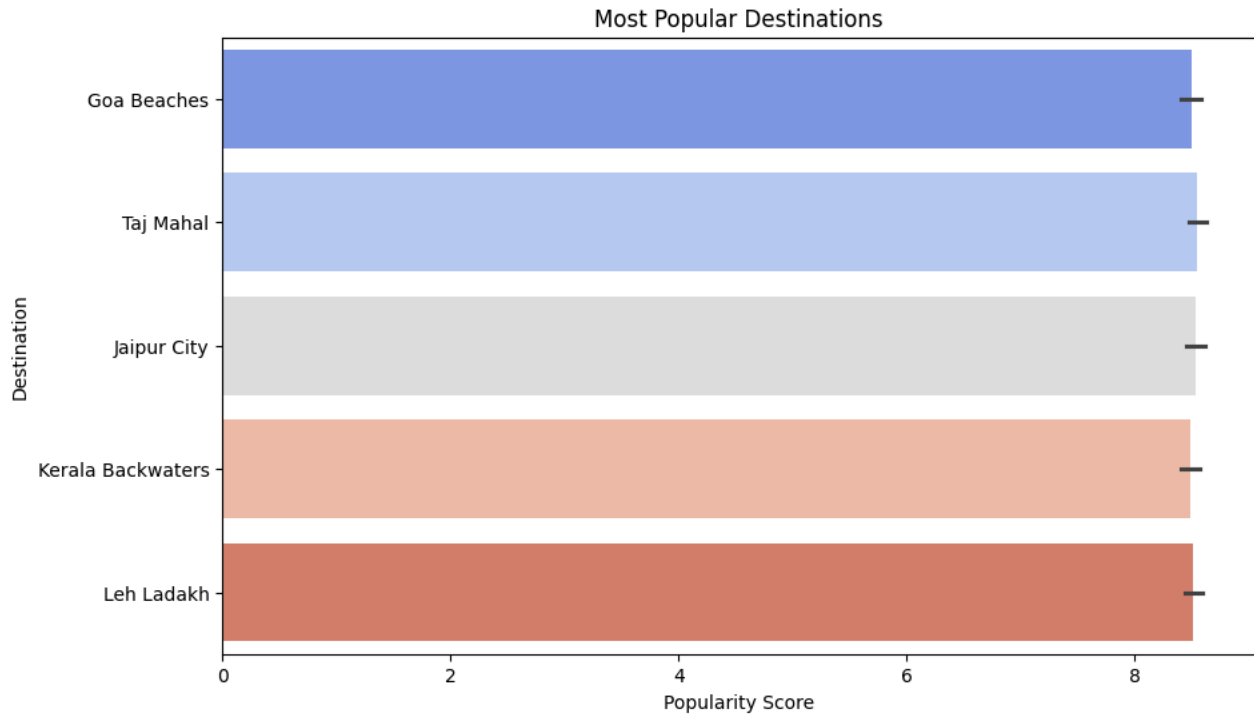
dtype: int64

```
plt.figure(figsize=(10, 6))
sns.barplot(y='Name', x='Popularity', data=destinations_df.sort_values(by='Popularity', ascending=True), palette='coolwarm')
plt.title('Most Popular Destinations')
plt.xlabel('Popularity Score')
plt.ylabel('Destination')
plt.show()
```

 /tmp/ipython-input-214-1003702628.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend`

```
sns.barplot(y='Name', x='Popularity', data=destinations_df.sort_values(by='Popularity', ascending=True), palette='coolwarm')
```




```
destinations_df['Type'].value_counts()
```



	count
Type	
Historical	200
Beach	200
City	200
Nature	200
Adventure	200

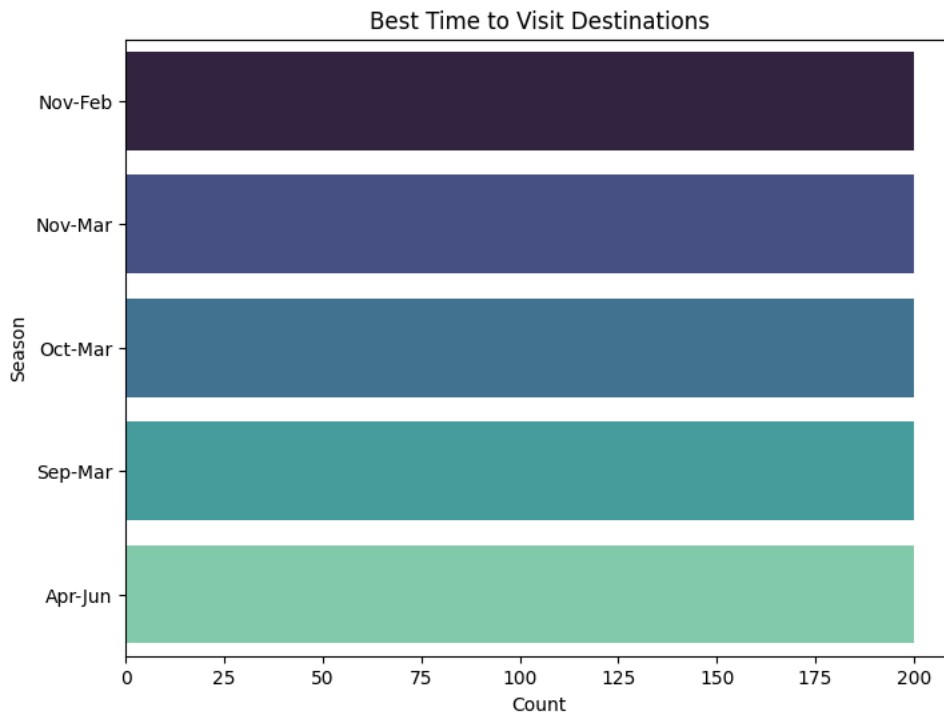
dtype: int64

```
plt.figure(figsize=(8, 6))
sns.countplot(y='BestTimeToVisit', data=destinations_df, order=destinations_df['BestTimeToVisit'].value_counts().index, palette='mako')
plt.title('Best Time to Visit Destinations')
plt.xlabel('Count')
plt.ylabel('Season')
plt.show()
```

 /tmp/ipython-input-216-924054954.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend`

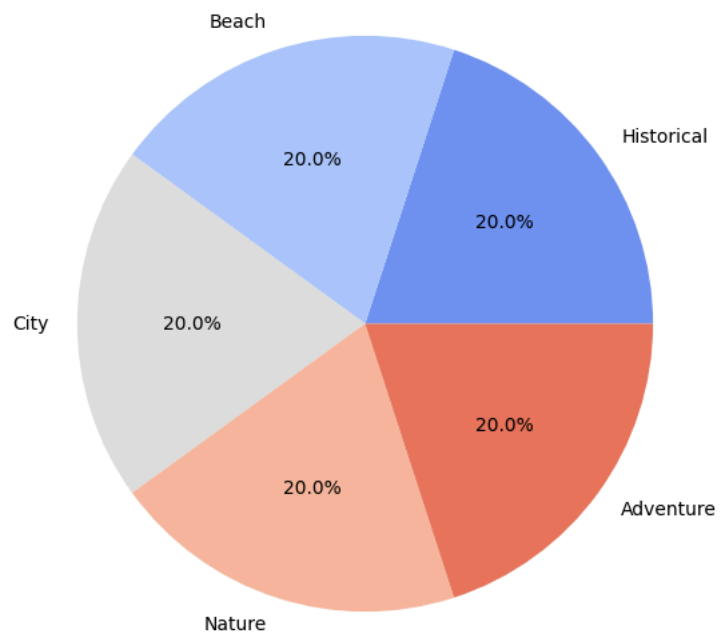
```
sns.countplot(y='BestTimeToVisit', data=destinations_df, order=destinations_df['BestTimeToVisit'].value_counts().index, palette='mako'
```



```
plt.figure(figsize=(7, 7))
destinations_df['Type'].value_counts().plot(kind='pie', autopct='%1.1f%%', colors=sns.color_palette('coolwarm', n_colors=len(destinations_df)
plt.title('Distribution of Destination Types')
plt.ylabel('') # Hides the y-axis label
plt.show()
```



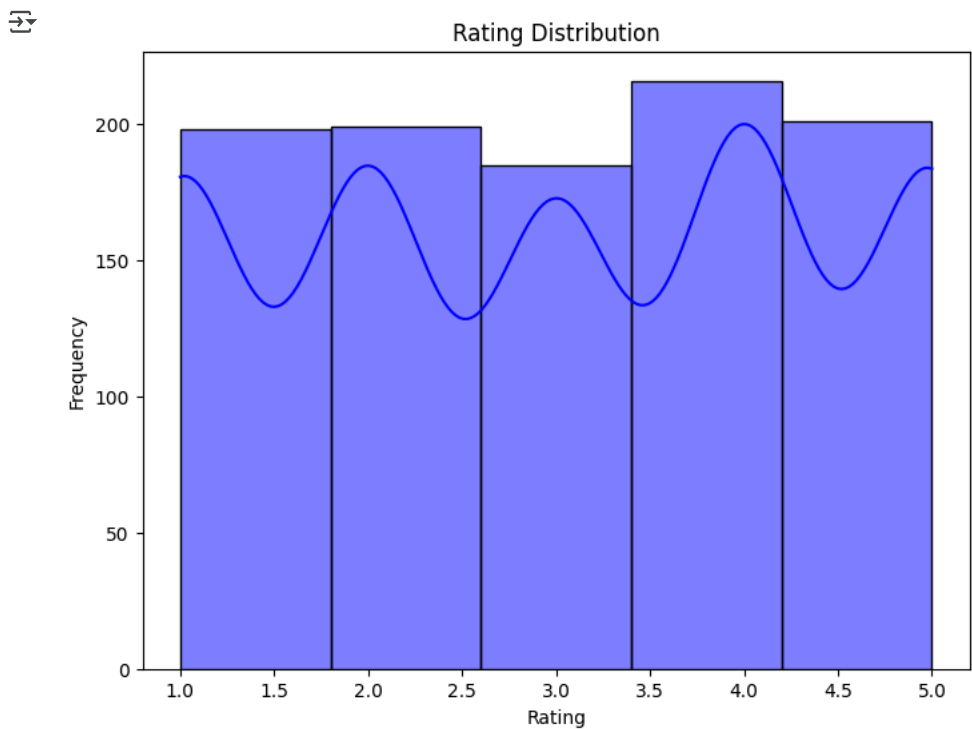
Distribution of Destination Types



```
plt.figure(figsize=(8, 6))
sns.histplot(reviews_df['Rating'], bins=5, kde=True, color='blue')
```



```
plt.title('Rating Distribution')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```




df

	ReviewID	DestinationID_x	UserID	Rating	ReviewText	Name_x	State	Type	Popularity	BestTimeToVisit	HistoryID	Destin
0	1	178	327	2	Incredible monument!	Jaipur City	Rajasthan	City	8.544352	Oct-Mar	79	
1	2	411	783	1	Loved the beaches!	Taj Mahal	Uttar Pradesh	Historical	8.284127	Nov-Feb	834	
2	4	358	959	3	Incredible monument!	Jaipur City	Rajasthan	City	7.738761	Oct-Mar	998	
3	5	989	353	2	Loved the beaches!	Kerala Backwaters	Kerala	Nature	8.208088	Sep-Mar	202	
4	6	473	408	4	A historical wonder	Jaipur City	Rajasthan	City	8.138558	Oct-Mar	331	
...	
988	991	701	850	3	Incredible monument!	Taj Mahal	Uttar Pradesh	Historical	8.814029	Nov-Feb	138	
989	991	701	850	3	Incredible monument!	Taj Mahal	Uttar Pradesh	Historical	8.814029	Nov-Feb	643	
990	995	231	346	5	Loved the beaches!	Taj Mahal	Uttar Pradesh	Historical	7.788256	Nov-Feb	454	
991	995	231	346	5	Loved the beaches!	Taj Mahal	Uttar Pradesh	Historical	7.788256	Nov-Feb	556	
992	997	823	858	5	Incredible monument!	Jaipur City	Rajasthan	City	8.501225	Oct-Mar	423	

993 rows × 20 columns

```
df['features'] = df['Type'] + ' ' + df['State'] + ' ' + df['BestTimeToVisit'] + " " + df['Preferences']
df
```




	ReviewID	DestinationID_x	UserID	Rating	ReviewText	Name_x	State	Type	Popularity	BestTimeToVisit	...	Destinatio
0	1	178	327	2	Incredible monument!	Jaipur City	Rajasthan	City	8.544352	Oct-Mar	...	
1	2	411	783	1	Loved the beaches!	Taj Mahal	Uttar Pradesh	Historical	8.284127	Nov-Feb	...	
2	4	358	959	3	Incredible monument!	Jaipur City	Rajasthan	City	7.738761	Oct-Mar	...	
3	5	989	353	2	Loved the beaches!	Kerala Backwaters	Kerala	Nature	8.208088	Sep-Mar	...	
4	6	473	408	4	A historical wonder	Jaipur City	Rajasthan	City	8.138558	Oct-Mar	...	
...	
988	991	701	850	3	Incredible monument!	Taj Mahal	Uttar Pradesh	Historical	8.814029	Nov-Feb	...	
989	991	701	850	3	Incredible monument!	Taj Mahal	Uttar Pradesh	Historical	8.814029	Nov-Feb	...	
990	995	231	346	5	Loved the beaches!	Taj Mahal	Uttar Pradesh	Historical	7.788256	Nov-Feb	...	
991	995	231	346	5	Loved the beaches!	Taj Mahal	Uttar Pradesh	Historical	7.788256	Nov-Feb	...	
992	997	823	858	5	Incredible monument!	Jaipur City	Rajasthan	City	8.501225	Oct-Mar	...	

993 rows x 21 columns

```
vectorizer = CountVectorizer(stop_words='english')
destination_features = vectorizer.fit_transform(df['features'])
```

```
destination_features.toarray()
```



```
array([[0, 0, 0, ..., 1, 0, 0],
       [0, 0, 0, ..., 0, 0, 1],
       [1, 0, 0, ..., 1, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 1],
```

```
[0, 0, 0, ..., 0, 0, 1],
[0, 0, 0, ..., 1, 0, 0]])
```

```
# Fit KNN model on destination feature matrix
knn_model = NearestNeighbors(n_neighbors=6, metric='cosine') # 5 neighbors + 1 self
knn_model.fit(destination_features)
```

```
NearestNeighbors
NearestNeighbors(metric='cosine', n_neighbors=6)
```

```
destination_features = vectorizer.fit_transform(df['features'])
```

```
df['features']
```

```
features
0      City Rajasthan Oct-Mar City, Historical
1  Historical Uttar Pradesh Nov-Feb City, Historical
2      City Rajasthan Oct-Mar Nature, Adventure
3      Nature Kerala Sep-Mar Nature, Adventure
4      City Rajasthan Oct-Mar City, Historical
...
988  Historical Uttar Pradesh Nov-Feb Beaches, Hist...
989  Historical Uttar Pradesh Nov-Feb Beaches, Hist...
990  Historical Uttar Pradesh Nov-Feb Beaches, Hist...
991  Historical Uttar Pradesh Nov-Feb Beaches, Hist...
992      City Rajasthan Oct-Mar City, Historical
993 rows x 1 columns

dtype: object
```

```
def recommend_destinations_knn(destination_index, df, knn_model, features_matrix):
    """
    Recommends destinations based on content-based filtering using KNN.

    Args:
    - destination_index: Index of the destination in the df and features_matrix for which recommendations are to be made.
    - df: DataFrame containing destination details and features.
    - knn_model: Trained NearestNeighbors model.
    - features_matrix: Feature matrix used to train the KNN model.

    Returns:
    - DataFrame with recommended destinations and their details.
    """
    distances, indices = knn_model.kneighbors(features_matrix[destination_index], n_neighbors=6)

    recommended = []
    for idx in indices.flatten()[1:]: # Skip the first (itself)
        recommended.append({
            'DestinationID': df.iloc[idx]['DestinationID_x'], # Use DestinationID_x
            'Name': df.iloc[idx]['Name_x'], # Use Name_x or Name_y depending on which name is desired
            'State': df.iloc[idx]['State'],
            'Type': df.iloc[idx]['Type'],
            'Popularity': df.iloc[idx]['Popularity']
        })

    # Remove duplicate DestinationIDs if any (can happen due to merging) and keep the first occurrence
    recommended_df = pd.DataFrame(recommended).drop_duplicates(subset=['DestinationID']).reset_index(drop=True)

    return recommended_df

# Example: Recommend places similar to the destination at index 0 in the df
recommendations_knn = recommend_destinations_knn(0, df, knn_model, destination_features)
```

```
print(recommendations_knn)
```

```

DestinationID      Name      State Type Popularity
0          823  Jaipur City  Rajasthan City      8.501225
1          373  Jaipur City  Rajasthan City      9.276957
2          398  Jaipur City  Rajasthan City      8.332950
3          183  Jaipur City  Rajasthan City      8.872499
4          118  Jaipur City  Rajasthan City      7.698986

```

```

user_item_matrix = userhistory_df.pivot(index='UserID', columns='DestinationID', values='ExperienceRating')
user_item_matrix = user_item_matrix.fillna(0)
user_item_matrix

```

```

DestinationID      2      3      4      5      6      7      8      9     10     11     ...     987     988     990     991     993     994     996     997     998     1000
UserID
1          0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2          0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3          0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
5          0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
7          0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
...          ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
990         0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
991         0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
992         0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
996         0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
999         0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  3.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

```

642 rows × 638 columns

```

user_item_matrix = userhistory_df.pivot(index='UserID', columns='DestinationID', values='ExperienceRating')
user_item_matrix = user_item_matrix.fillna(0)

```

```

# Fit KNN model
knn_user = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=6)
knn_user.fit(user_item_matrix)

```

```

NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine', n_neighbors=6)

```

```

def collaborative_recommend_knn(user_id, user_item_matrix, destinations_df, knn_model, num_recommendations=5):
    if user_id not in user_item_matrix.index:
        print("User not found.")
        return []

    user_idx = user_item_matrix.index.get_loc(user_id)
    distances, indices = knn_model.kneighbors(user_item_matrix.iloc[[user_idx]], n_neighbors=6)

    similar_users = user_item_matrix.iloc[indices.flatten()[1:]]
    mean_ratings = similar_users.mean().sort_values(ascending=False)

    already Rated = user_item_matrix.loc[user_id]
    already Rated = already Rated[already Rated > 0].index

    recommendations = mean_ratings.drop(index=already Rated, errors='ignore').head(num_recommendations)
    return destinations_df[destinations_df['DestinationID'].isin(recommendations.index)][['DestinationID', 'Name', 'Type', 'State']]

```

```

user_id = 200001 # Replace with a real UserID from your data
knn_recommendations = collaborative_recommend_knn(user_id, user_item_matrix, destinations_df, knn_user)

```

```

print("Recommended destinations for user:", user_id)
print(knn_recommendations)

```

```

➡ User not found.
Recommended destinations for user: 200001
[]

```

```

# No need to create a copy, we will encode directly on df
# data=df.copy()

```

```

# Predicting popularity
features = ['Name_x', 'State', 'Type', 'BestTimeToVisit', 'Preferences', 'Gender', 'NumberOfAdults', 'NumberOfChildren']
target = 'Popularity'

```

```

# Use df directly
X = df[features]
y = df[target]

```

```

# Use df directly
df[features]

```

```
➡
```

	Name_x	State	Type	BestTimeToVisit	Preferences	Gender	NumberOfAdults	NumberOfChildren
0	Jaipur City	Rajasthan	City	Oct-Mar	City, Historical	Female	1	1
1	Taj Mahal	Uttar Pradesh	Historical	Nov-Feb	City, Historical	Male	1	1
2	Jaipur City	Rajasthan	City	Oct-Mar	Nature, Adventure	Male	1	1
3	Kerala Backwaters	Kerala	Nature	Sep-Mar	Nature, Adventure	Female	2	0
4	Jaipur City	Rajasthan	City	Oct-Mar	City, Historical	Male	2	0
...
988	Taj Mahal	Uttar Pradesh	Historical	Nov-Feb	Beaches, Historical	Male	2	0
989	Taj Mahal	Uttar Pradesh	Historical	Nov-Feb	Beaches, Historical	Male	2	0
990	Taj Mahal	Uttar Pradesh	Historical	Nov-Feb	Beaches, Historical	Male	2	2
991	Taj Mahal	Uttar Pradesh	Historical	Nov-Feb	Beaches, Historical	Male	2	2
992	Jaipur City	Rajasthan	City	Oct-Mar	City, Historical	Male	1	2

993 rows × 8 columns

```

for col in features:
    print(col)

```

```

➡ Name_x
State
Type
BestTimeToVisit
Preferences
Gender
NumberOfAdults
NumberOfChildren

```

```

label_encoders = {}
for col in features:
    # Check if the column exists in df and if its dtype is object
    if col in df.columns and df[col].dtype == 'object':
        le = LabelEncoder()
        # Fit and transform directly on the df DataFrame
        df[col] = le.fit_transform(df[col].astype(str))
        label_encoders[col] = le
    # If the column is already numerical and is in features, keep it as is
    elif col not in df.columns:
        print(f"Warning: Feature '{col}' not found in df columns.")

```

```

# Use df directly
df[features]

```



	Name_x	State	Type	BestTimeToVisit	Preferences	Gender	NumberOfAdults	NumberOfChildren
0	1	3	2	3	1	0	1	1
1	4	4	3	1	1	1	1	1
2	1	3	2	3	2	1	1	1
3	2	2	4	4	2	0	2	0
4	1	3	2	3	1	1	2	0
...
988	4	4	3	1	0	1	2	0
989	4	4	3	1	0	1	2	0
990	4	4	3	1	0	1	2	2
991	4	4	3	1	0	1	2	2
992	1	3	2	3	1	1	1	2

993 rows × 8 columns

```
# Use df directly
X = df[features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
#Train Model
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)
```

```
#Evaluation
y_pred = model.predict(X_test)
print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred):.2f}")
print(f"R² Score: {r2_score(y_test, y_pred):.2f}")
```



Mean Squared Error: 0.30
R² Score: 0.07

```
def recommend_destinations(user_input, model, label_encoders, features, df):
    # Create a dictionary for the encoded input
    encoded_input = {}
    for feature in features:
        if feature in user_input:
            if feature in label_encoders:
                # Use the stored label encoder to transform the input value
                try:
                    encoded_input[feature] = label_encoders[feature].transform([str(user_input[feature])])[0]
                except ValueError as e:
                    print(f"Error encoding feature '{feature}': {e}")
                    print(f"Input value was: '{user_input[feature}]'")
                    print(f"Known classes for this encoder: {label_encoders[feature].classes_}")
                    # Handle unseen labels - for now, we'll raise the error
                    raise e
            else:
                # If not a categorical feature that was encoded, use the input value directly
                encoded_input[feature] = user_input[feature]
        else:
            # Handle cases where a feature in the 'features' list is not in the user_input
            # Depending on the model and feature, you might impute a default value or raise an error
            # For simplicity, let's assume all features in the list are expected in user_input
            print(f"Warning: Feature '{feature}' not provided in user input.")
            # As a placeholder, you might add a default value or handle this case based on your needs
            # For now, we'll assume the input is complete based on the 'features' list
            pass # Or set a default value, e.g., encoded_input[feature] = default_value

    # Convert to DataFrame
    # Ensure the order of columns in the input_df matches the order of features the model was trained on
    input_df = pd.DataFrame([encoded_input])[features]

    # Predict popularity
    predicted_popularity = model.predict(input_df)[0]
```