# Step-by-Step Explanation of the Tourism Data Processing Code (Data Processing Code)

### **1. Importing Necessary Libraries**

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier

from sklearn.metrics import mean\_squared\_error, r2\_score, accuracy\_score, classification\_report

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

import lightgbm as lgb

import xgboost as xgb

import pickle

import warnings

warnings.filterwarnings('ignore')

* **pandas & numpy**: Used for handling and manipulating data.
* **scikit-learn (sklearn)**: Used for data preprocessing, model training, and evaluation.
* **TfidfVectorizer & cosine\_similarity**: Used for recommendation system.
* **lightgbm & xgboost**: Machine learning models for regression and classification.
* **pickle**: Used for model serialization.
* **warnings**: Suppresses warnings for better readability.

### **2. Defining the TourismDataProcessor Class**

class TourismDataProcessor:

def \_\_init\_\_(self):

self.df = None

self.rating\_model = None

self.visit\_mode\_model = None

self.label\_encoder = None

self.recommendation\_system = None

self.X\_reg = None

self.y\_reg = None

self.X\_clf = None

self.y\_clf = None

self.cosine\_sim = None

self.attraction\_indices = None

self.attraction\_features = None

* Initializes class attributes to store data and models.

### **3. Loading Data**

def load(self):

transaction = pd.read\_excel('Transaction.xlsx')

user = pd.read\_excel('User.xlsx')

city = pd.read\_excel('City.xlsx')

attraction\_type = pd.read\_excel('Type.xlsx')

visit\_mode = pd.read\_excel('Mode.xlsx')

continent = pd.read\_excel('Continent.xlsx')

country = pd.read\_excel('Country.xlsx')

region = pd.read\_excel('Region.xlsx')

attraction = pd.read\_excel('Item.xlsx')

return transaction, user, city, attraction\_type, visit\_mode, continent, country, region, attraction

* Reads multiple Excel files containing tourism data.

### **4. Data Cleaning**

def clean(self, df):

cols\_to\_keep = ['UserId', 'VisitYear', 'VisitMonth', 'VisitMode', 'AttractionId', 'Rating',

'Continent', 'Region', 'Country', 'CityName', 'AttractionType', 'Attraction', 'AttractionAddress']

df = df.loc[:, ~df.columns.duplicated()]

df = df[[col for col in cols\_to\_keep if col in df.columns]]

df = df.dropna(subset=['Rating'])

for col in ['VisitMode', 'Continent', 'Region', 'Country', 'CityName', 'AttractionType']:

if col in df.columns:

df[col] = df[col].fillna('Unknown')

df[col] = df[col].astype('category')

return df

* Drops duplicate columns, retains necessary ones, and fills missing values.

### **5. Merging DataFrames**

def merge(self, transaction, user, city, attraction\_type, visit\_mode,

continent, country, region, attraction):

user = user.merge(city, on='CityId', how='left')

user = user.merge(country, on='CountryId', how='left')

user = user.merge(region, on='RegionId', how='left')

user = user.merge(continent, on='ContinentId', how='left')

attraction = attraction.merge(city, left\_on='AttractionCityId', right\_on='CityId', how='left')

attraction = attraction.merge(country, on='CountryId', how='left')

attraction = attraction.merge(region, on='RegionId', how='left')

attraction = attraction.merge(continent, on='ContinentId', how='left')

attraction = attraction.merge(attraction\_type, on='AttractionTypeId', how='left')

df = transaction.merge(user, on='UserId', how='left')

df = df.merge(attraction, on='AttractionId', how='left')

df = df.merge(visit\_mode, on='VisitMode', how='left')

return df

* Merges multiple data sources into a single DataFrame.

### **6. Feature Engineering**

def feature\_engineering(self, df):

user\_stats = df.groupby('UserId').agg({'Rating': 'mean', 'AttractionId': 'count'}).rename(columns={'Rating': 'UserAvgRating'})

attraction\_stats = df.groupby('AttractionId').agg({'Rating': 'mean', 'UserId': 'count'}).rename(columns={'Rating': 'AttractionAvgRating'})

df = df.merge(user\_stats, on='UserId', how='left')

df = df.merge(attraction\_stats, on='AttractionId', how='left')

df['YearsSinceFirstVisit'] = 2025 - df['VisitYear']

return df

* Creates additional features such as user and attraction statistics.

### **7. Preparing Regression Data**

def prepare\_regression(self, df):

features = ['UserAvgRating', 'UserVisitCount', 'AttractionAvgRating', 'YearsSinceFirstVisit']

X = df[features]

y = df['Rating']

return X, y

* Selects features and target variable for rating prediction.

### **8. Training Regression Model**

def train\_regression\_model(self, X, y):

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

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

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

return rf

* Splits data, trains a Random Forest regression model.

### **9. Preparing Classification Data**

def prepare\_classification(self, df):

X = df[['UserAvgRating', 'UserVisitCount', 'AttractionAvgRating']]

y = df['VisitMode']

return X, y

* Selects features and target for visit mode classification.

### **10. Training Classification Model**

def train\_classification\_model(self, X, y):

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

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

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

return rf

* Trains a Random Forest classification model.

### **11. Building Recommendation System**

def build\_recommendation\_system(self, df):

tfidf = TfidfVectorizer()

tfidf\_matrix = tfidf.fit\_transform(df['AttractionType'])

self.cosine\_sim = cosine\_similarity(tfidf\_matrix, tfidf\_matrix)

* Uses TF-IDF and cosine similarity for content-based recommendations.

This document provides a structured breakdown of the TourismDataProcessor class.

# **Step-by-Step Explanation of the Tourism Recommendation System Code(StreamLitApp.py)**

## **1. Import Required Libraries**

import streamlit as st

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import pickle

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics.pairwise import cosine\_similarity

**Explanation:**

* streamlit: Used for building a web-based user interface.
* pandas: Handles data manipulation and analysis.
* numpy: Supports numerical operations.
* matplotlib.pyplot: Enables plotting and visualization.
* seaborn: Enhances data visualization.
* pickle: Loads pre-trained machine learning models.
* LabelEncoder: Encodes categorical features for model input.
* cosine\_similarity: Measures similarity between vectors for recommendations.

## **2. Set Streamlit Page Configuration**

st.set\_page\_config(page\_title="Tourism Recommendation", layout="wide")

**Explanation:**

* Configures the Streamlit app with a custom title and wide layout.

## **3. Load Models and Processed Data**

@st.cache\_data

def load\_models():

with open('tourism\_models.pkl', 'rb') as f:

data = pickle.load(f)

return data

data = load\_models()

**Explanation:**

* @st.cache\_data: Caches the loaded model to improve performance.
* pickle.load(f): Loads a pre-trained model from a .pkl file.
* data: Dictionary containing loaded models and processed data.

## **4. Extract Required Data from the Loaded Models**

df = data['df']

rating\_model = data['rating\_model']

visit\_mode\_model = data['visit\_mode\_model']

label\_encoder = data['label\_encoder']

cosine\_sim = data['cosine\_sim']

attraction\_indices = data['attraction\_indices']

attraction\_features = data['attraction\_features']

X\_reg = data['X\_reg']

X\_clf = data['X\_clf']

**Explanation:**

* Extracts various components from the loaded data dictionary, including:  
  + df: Processed dataset.
  + rating\_model: Regression model for rating prediction.
  + visit\_mode\_model: Classification model for predicting visit mode.
  + label\_encoder: Encodes categorical labels.
  + cosine\_sim: Precomputed similarity matrix for recommendations.
  + attraction\_indices: Maps attractions to indices.
  + attraction\_features: Attraction feature set.
  + X\_reg: Features used for rating prediction.
  + X\_clf: Features used for visit mode classification.

## **5. Define Hybrid Recommendation Function**

def hybrid\_recommendations(user\_id, top\_n=5):

user\_ratings = df[df['UserId'] == user\_id]

if user\_ratings.empty:

return df.groupby('Attraction')['Rating'].mean().sort\_values(ascending=False).head(top\_n)

**Explanation:**

* Retrieves user-specific ratings.
* If no past ratings exist, it returns top attractions based on average ratings.

liked\_attractions = user\_ratings[user\_ratings['Rating'] >= 4]['AttractionId'].unique()

if len(liked\_attractions) == 0:

liked\_attractions = user\_ratings['AttractionId'].unique()

**Explanation:**

* Identifies attractions rated 4+ by the user as "liked".
* If none exist, considers all visited attractions.

sim\_scores = []

for att\_id in liked\_attractions:

if att\_id in attraction\_indices:

idx = attraction\_indices[att\_id]

if isinstance(idx, pd.Series):

idx = idx.iloc[0] # Take the first index if multiple exist

if idx < cosine\_sim.shape[0]:

sim\_scores.append(cosine\_sim[idx])

**Explanation:**

* Fetches attraction indices from attraction\_indices.
* Ensures the index is within valid range before retrieving similarity scores.

if not sim\_scores:

return df.groupby('Attraction')['Rating'].mean().sort\_values(ascending=False).head(top\_n)

avg\_sim\_scores = np.mean(sim\_scores, axis=0)

content\_recs = list(zip(attraction\_features['AttractionId'], avg\_sim\_scores))

content\_recs = sorted(content\_recs, key=lambda x: x[1], reverse=True)

content\_recs = [rec for rec in content\_recs if rec[0] not in user\_ratings['AttractionId'].values]

content\_recs = content\_recs[:top\_n]

**Explanation:**

* Computes an average similarity score for recommendations.
* Filters out attractions the user has already rated.
* Returns top N recommendations.

## **6. Prepare Input for Prediction**

def prepare\_input\_for\_prediction(user\_data, attraction\_data, visit\_mode\_name):

input\_data = {

'UserAvgRating': user\_data['UserAvgRating'],

'UserVisitCount': user\_data['UserVisitCount'],

'UserCommonVisitMode': user\_data['UserCommonVisitMode'],

'UserContinent': user\_data['UserContinent'],

'AttractionAvgRating': attraction\_data['AttractionAvgRating'],

'AttractionVisitCount': attraction\_data['AttractionVisitCount'],

'AttractionCommonVisitMode': attraction\_data['AttractionCommonVisitMode'],

'AttractionType': attraction\_data['AttractionType'],

'VisitMode': visit\_mode\_name,

'Continent': attraction\_data['Continent'],

'YearsSinceFirstVisit': user\_data['YearsSinceFirstVisit']

}

**Explanation:**

* Prepares input features for rating prediction.
* Combines user-specific and attraction-specific features.

## **7. Exploratory Data Analysis (EDA)**

def perform\_eda(df):

st.title("Tourism Data Explorer")

st.subheader("Rating Distribution")

fig, ax = plt.subplots()

sns.histplot(df['Rating'], bins=5, kde=True, ax=ax)

st.pyplot(fig)

**Explanation:**

* Displays data exploration with histograms and bar charts.
* Visualizes rating distribution.

## **8. Streamlit Web Interface**

def run\_app():

st.title("Tourism Recommendation and Prediction System")

user\_id = st.sidebar.selectbox("User ID", df['UserId'].unique())

user\_data = df[df['UserId'] == user\_id].iloc[0] if user\_id in df['UserId'].values else None

**Explanation:**

* Creates a Streamlit-based web interface.
* Retrieves user ID and related data.

if user\_data is not None:

st.subheader("User Profile")

col1, col2, col3 = st.columns(3)

col1.metric("Continent", user\_data['Continent'])

col2.metric("Country", user\_data['Country'])

col3.metric("City", user\_data['CityName'])

**Explanation:**

* Displays user profile details.

recs = hybrid\_recommendations(user\_id)

st.subheader("Personalized Recommendations")

for att\_name, att\_type, country, score in recs:

st.write(f"- \*\*{att\_name}\*\* ({att\_type}, {country}) - Similarity: {score:.2f}")

**Explanation:**

* Displays personalized recommendations.

## **9. Run Application**

if \_\_name\_\_ == '\_\_main\_\_':

run\_app()

**Explanation:**

* Ensures run\_app() executes when the script is run directly.