Tourism Experience Analytics: Classification, Prediction, and Recommendation System

Technical Documentation
Machine Learning & Deployment Architecture Guide

1. Project Overview

Tourism Experience Analytics is a production-ready Machine Learning application built to support tourism platforms in predicting user satisfaction, identifying visitor travel modes, and generating personalized attraction recommendations.

The system integrates:

- Regression modeling (Rating Prediction)
- Classification modeling (Visit Mode Prediction)
- Recommendation engine (Attraction Suggestions)
- Model comparison framework
- Production-grade Streamlit deployment
- Automated best-model selection
- Robust preprocessing pipeline
- Error-safe fallback architecture

This document explains the technical architecture, file structure, model pipeline, deployment logic, and system flow.

2. Problem Statement

Tourism agencies and travel platforms aim to enhance user experiences by leveraging data to provide personalized recommendations, predict user satisfaction, and classify potential user behavior. This project involves analyzing user preferences, travel patterns, and attraction features to achieve three primary objectives: regression, classification, and recommendation.

3. System Architecture

The application follows a modular ML deployment architecture.

Raw Dataset
↓
Data Cleaning & Feature Engineering (Notebook)
↓
Model Training & Evaluation
↓
Model Comparison & Best Model Selection
↓
Save Models + Scaler + Encoders
↓
Save Best Model Config (JSON)
↓
Streamlit Application
↓
User Input → Preprocessing → Prediction → UI Rendering

The system separates training and deployment phases to ensure reproducibility and maintainability.

4. Project Structure

```
Tourism-Experience-Analytics/
│
├── data/
│   └── processed/
│       └── master_dataset.csv
│
├── models/
│   ├── linear_regression.pkl
│   ├── random_forest_regression.pkl
│   ├── gradient_boosting_regression.pkl
│   ├── logistic_regression.pkl
│   ├── random_forest_clf.pkl
│   ├── gradient_boosting_clf.pkl
│   ├── xgboost_clf.pkl
│   ├── lightgbm_clf.pkl
│   ├── scaler.pkl
│   └── label_encoders.pkl
│
├── reports/
│   ├── regression_model_comparison.csv
│   ├── classification_model_comparison.csv
│   ├── best_regression_model.json
│   └── best_classification_model.json
│
├── app.py
├── requirements.txt
└── README.md
```

5. Data Pipeline

5.1 Data Cleaning

Performed in Jupyter Notebook:

- Removed missing values
- Validated rating range
- Encoded categorical variables
- Ensured numeric feature consistency
- Removed invalid records

5.2 Feature Engineering

Key engineered features:

- VisitYear
- VisitMonth
- Attraction_Popularity
- Visit_Frequency
- User_Experience
- Encoded Continent, Region, Country, City

Classification-specific:

- Rating_Class (Low / Medium / High)

5.3 Feature Scaling

StandardScaler was applied on numeric columns:

- Rating
- User_Avg_Rating
- Attraction_Popularity
- Visit_Frequency

Scaler saved as: *models/scaler.pkl*
Important: The app performs inverse transformation before displaying rating predictions.

6. Machine Learning Models

6.1 Regression Models

Purpose: Predict attraction rating.
Models Trained:

- Linear Regression
- Random Forest Regressor
- Gradient Boosting Regressor (Tuned)

Evaluation Metric:

- R^2 Score

Best Model:

- Gradient Boosting (Tuned)

Stored in: *models/gradient_boosting_regression.pkl*

6.2 Classification Models

Purpose: Predict visit mode.
Models Trained:

- Logistic Regression
- Random Forest Classifier
- Gradient Boosting Classifier
- XGBoost Classifier
- LightGBM Classifier

Evaluation Metrics:

- Accuracy
- Precision
- Recall
- F1 Score

Best Model:

- XGBoost (Tuned)

Stored in: *models/xgboost_clf.pkl*

7. Model Comparison Framework

All trained models are evaluated and saved into CSV files:
reports/regression_model_comparison.csv
reports/classification_model_comparison.csv
Best models are selected automatically and saved into JSON configuration files:

reports/best_regression_model.json
reports/best_classification_model.json
Example JSON structure:

```
{
  "task": "regression",
  "model_name": "Gradient Boosting (Tuned)",
  "model_path": "./models/gradient_boosting_regression.pkl",
  "metric": "R2",
  "score": 0.7453
}
```

This allows the Streamlit app to dynamically load the best model.

8. Streamlit Application Architecture

8.1 UI Structure

The application consists of:

- Tab 1: Rating Prediction
- Tab 2: Visit Mode Prediction
- Tab 3: Recommendation Engine
- Sidebar: Model Control Panel & System Status

8.2 Model Selection System

The app supports two modes:

1. Automatic Mode
    - Loads best model from JSON
2. Manual Mode
    - User selects model from dropdown

Each tab independently controls model selection.

8.3 Preprocessing Pipeline

Before prediction:

1. Input captured from UI
2. Label encoding applied
3. Feature alignment using model.feature_names_in_
4. Scaling applied (if available)
5. Model prediction executed

For regression:

- Output is inverse-scaled
- Clamped to rating range (1–5)

9. Recommendation Engine Logic

The recommendation engine:

- Identifies user's most visited AttractionType
- Filters attractions with same type
- Ranks by popularity
- Returns results

User can adjust number of recommendations using slider.

10. Error Handling & Fallback System

The system includes:

- Safe model loading (try/except)
- Dataset existence validation
- Scaler & encoder validation
- Version mismatch handling
- JSON path validation
- Independent tab-level failure isolation

If one model fails, other modules remain functional.

11. Deployment Environment

The project was trained in Google Colab and deployed locally.
Important:
All libraries must match training environment.
Recommended versions:
streamlit==1.32.2
pandas==2.2.2
joblib==1.3.2
matplotlib==3.8.4
seaborn==0.13.2
xgboost==2.0.3
lightgbm==4.3.0y
scipy==1.13.0
numpy==2.0.0
scikit-learn==1.6.1
Version mismatch may cause model loading errors.

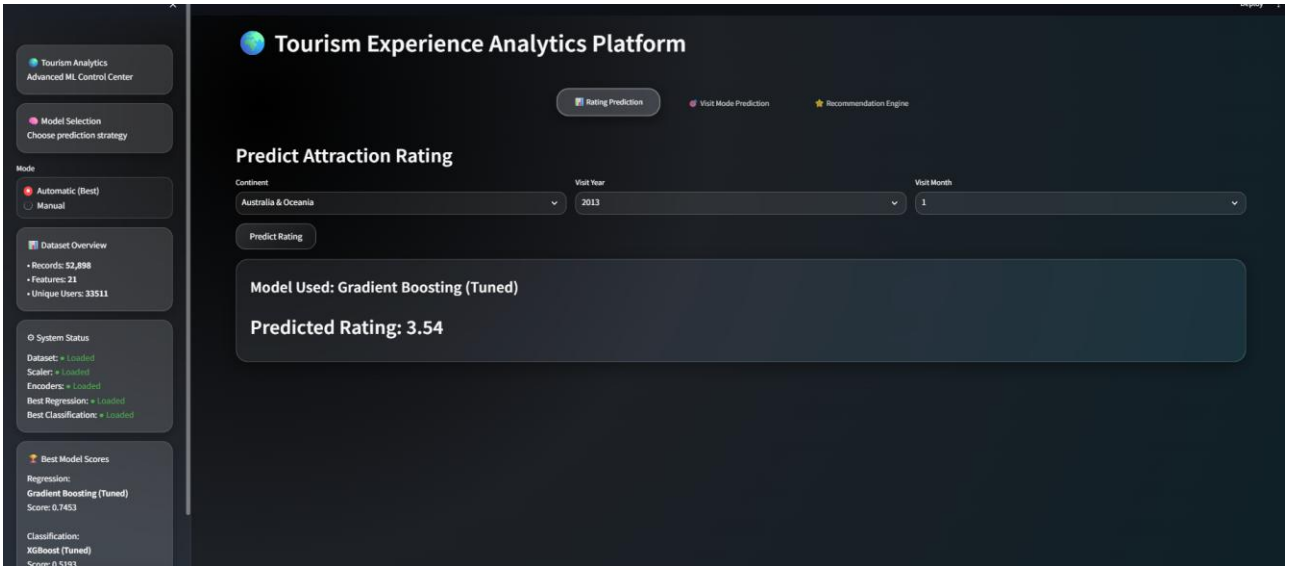
12. System Flow (Runtime)

When user clicks prediction:

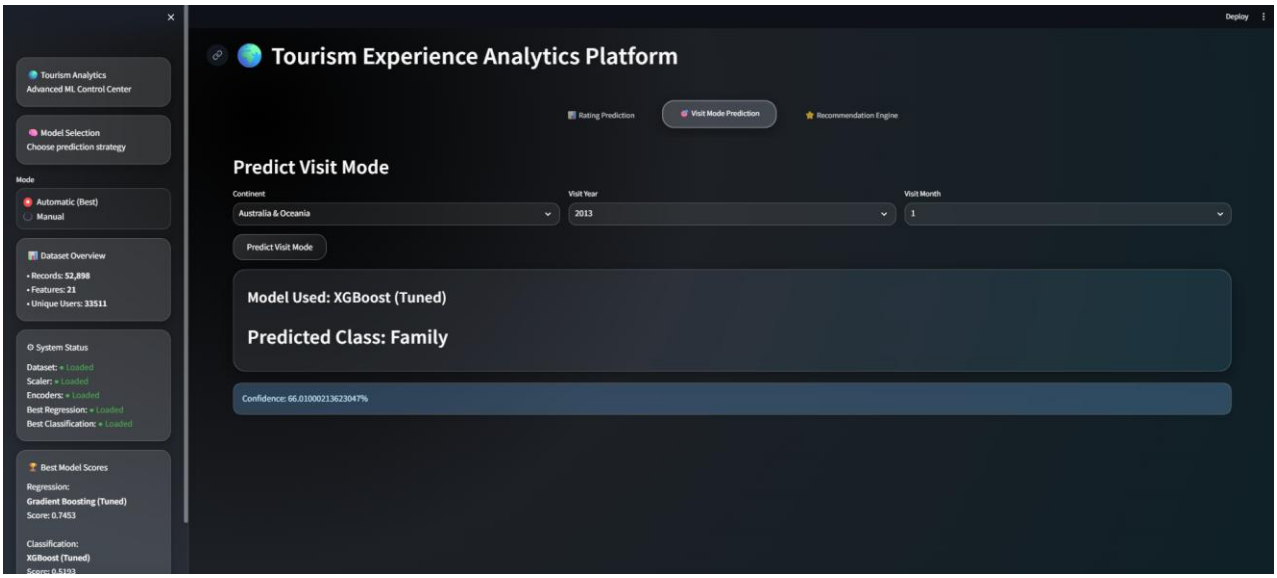
User Input
↓
Input DataFrame Creation
↓
Preprocessing Pipeline
↓
Model Prediction
↓
Post-processing (inverse scaling / confidence)
↓
Glass Card Output Rendering

13. Output

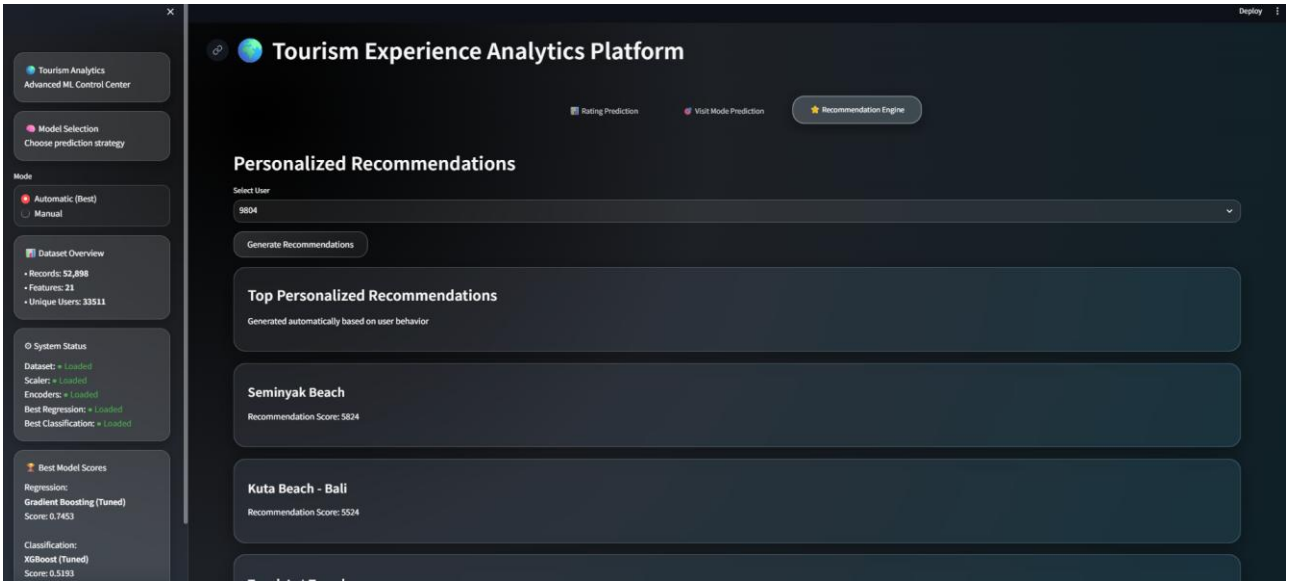
Rating predication:



Visit Mode Predication:



Recommendation Engine:



14. Security & Maintainability

The project follows best practices:

- Models separated from UI
- JSON-driven configuration
- Modular preprocessing
- No hard-coded paths
- Graceful failure handling
- Version-controlled dependencies

15. Limitations

- Recommendation system is popularity-based (not collaborative filtering)
- No deep learning models implemented
- No real-time external API integration
- Requires version consistency for model loading

16. Future Improvements

- Add collaborative filtering recommendation engine
- Add SHAP model explainability
- Integrate real-time tourism APIs
- Add database backend (PostgreSQL)
- Add user authentication
- Add dashboard analytics charts
- Add model confidence explanation layer
- Docker containerization for cloud deployment

17. Conclusion

Tourism Experience Analytics is a fully functional, production-ready ML application integrating:

- Supervised regression
- Multi-class classification
- Recommendation systems
- Model comparison automation
- Dynamic model loading
- Glassmorphism UI
- Robust deployment pipeline

The project demonstrates real-world ML system design including training, evaluation, version control, deployment, monitoring, and error-safe architecture.
It stands as a scalable foundation for tourism intelligence platforms.