International Hotel Booking Analytics

Milestone 1 Report

Done By: Dina Ahmed, Iten Mohamed, Jana Ahmed, Jolie Magdy

October 24, 2025

Abstract

This report documents the machine learning pipeline for predicting the **country group** of hotel reviewers using the International Hotel Booking Analytics dataset. The pipeline includes: data cleaning, exploratory data analysis, feature engineering, feature selection, model training (Random Forest, Logistic Regression, Feed-Forward Neural Network), evaluation (train/validation/test), inference function, and explainable AI (SHAP and LIME) global and local explanations. The report follows the notebook implementation and provides code excerpts, plots placeholders, and interpretation of results.

Contents

1 Introduction

This milestone aims to build a classifier that predicts the country group of hotel reviewers using three joined tables:

- **hotels.csv** (25 hotels) — hotel metadata and baseline ratings.
- **reviews.csv** (50,000 reviews) — per-review multi-dimensional scores and textual fields.
- **users.csv** (2,000 users) — demographic info including country and traveller type.

The notebook loads and merges these files into a single dataframe for EDA, feature engineering, modeling, and XAI.

2 Overview

This report covers:

1. Data cleaning
2. Data analysis
3. Plots summarizing results
4. Answers to Data Engineering questions
5. Feature selection with reasons and evidence
6. Preprocessing & feature engineering rationale and effects
7. Prediction models: choice, workings, limitations, and detailed performance reporting
8. Inference function for production usage
9. Explainable AI: global & local explanations (SHAP & LIME) with plots and interpretation

3 Data Loading and Initial Checks

Representative code (from the notebook) used to load the datasets:

```
hotels = pd.read_csv('/kaggle/input/international-hotel-booking-analytics/
    hotels.csv')
reviews = pd.read_csv('/kaggle/input/international-hotel-booking-analytics/
    reviews.csv')
users = pd.read_csv('/kaggle/input/international-hotel-booking-analytics/users.
    csv')

print("hotels:", hotels.shape)
print("reviews:", reviews.shape)
print("users:", users.shape)
```

Listing 1: Loading the datasets

3.1 Checks performed

- Inspect head() and info() for each table to confirm datatypes and missing values.
- Merges performed with ‘reviews.merge(hotels, on=’hotel-id’)‘ and ‘...merge(users, on=’user-id’)‘ to create a master dataframe.

4 Data Cleaning

4.1 Actions performed

Representative cleaning helper in the notebook:

```
def normalize_col(df, col):
    if col in df.columns and df[col].dtype == object:
        df[col] = df[col].astype(str).str.strip()
        return True
    return False
```

Listing 2: Example helper to normalize string columns

Other cleaning steps applied in the notebook:

- Mapping categorical text to numeric (e.g., 'user-gender.map('Male':0, ...)').
- Converting date columns ('review-date') to datetime and extracting features (year, month).
- Handling missing values: either drop or impute depending on column (noted in notebook cell comments).
- Removing duplicates and verifying unique keys (review_id, user_id, hotel_id).

4.2 Why each step is needed

- Normalizing strings avoids spurious categories (leading/trailing spaces).
- Numeric maps allow ML algorithms to train.
- Date decomposition captures seasonality trends or travel patterns.
- Missing values handling ensures downstream models are not biased or error-prone.

5 Exploratory Data Analysis

The notebook computes summary statistics and plots distribution of key features:

- Score distributions: 'score-overall', 'score-cleanliness', 'score-comfort', etc.
- Hotel baseline vs. review scores: to see if hotels under/over-perform relative to baseline metrics.
- User demographics: age distribution, traveller-type breakdown, gender split.
- Country-group frequency (target class balance).

6 Feature Selection

6.1 Candidate features used

Based on the notebook, features considered include:

- Numeric review scores: {score_overall, score_cleanliness, score_comfort, score_facilities, score_location, score_staff, score_value_for_money}.
- Hotel metadata: star_rating, cleanliness_base, comfort_base, facilities_base.
- User demographics: age, gender (mapped to numeric), traveller_type (one-hot).

- Temporal features: review_year, review_month (if available).
- Aggregations (optional): per-hotel average score, user historic average (if computed).

6.2 Reason for chosen features and evidence

- **Review score dimensions** directly measure user satisfaction and are strong predictors of user sentiment and preferences.
- **Hotel baseline metrics** allow capturing hotel-level quality; differences between baseline and review indicate relative satisfaction.
- **Demographics & traveller_type** capture cultural and travel-purpose differences across country groups.
- Evidence: feature importance from Random Forest and SHAP summary plots (included in XAI) are used to justify feature importance.

6.3 Feature Sets Used in the Experiment

First, two distinct feature sets were utilized during model training and evaluation: **All Features** and **Some Features**. Each set was designed with a specific purpose and contributes differently to the model's performance and complexity. The following section explains their composition, motivation, and expected effects.

1. All Features

```
score_features = [
   'score_overall', 'score_cleanliness', 'score_comfort',
   'score_facilities', 'score_location', 'score_staff',
   'score_value_for_money', 'cleanliness_base', 'comfort_base',
   'facilities_base', 'location_base', 'staff_base',
   'value_for_money_base'
]

user_features = ['age_group', 'traveller_type', 'user_gender']

X_All = df[score_features + user_features]
```

Explanation:
This feature set includes all user rating attributes alongside the corresponding `base` features that represent the hotel's average ratings across different aspects (e.g., cleanliness, comfort, location). Additionally, user-related characteristics such as `age_group`, `traveller_type`, and `user_gender` are incorporated.

Including both user-specific and hotel-level attributes enables the model to understand not only how an individual perceives a hotel but also how that perception compares to the hotel's overall standing. For instance, if a user's score for "comfort" is significantly lower than the hotel's `comfort_base`, this may indicate dissatisfaction or a negative review tendency.

Effect on Model:

- Provides deeper contextual understanding by combining user and hotel-level information.
- Generally improves accuracy, as the model learns relative patterns between user and base scores.

- Increases model complexity and may introduce multicollinearity, slightly raising the risk of overfitting.

Significance:
The inclusion of all features produces a richer and more nuanced model, capable of capturing subtle relationships in the data. This set is particularly valuable when the aim is to maximize predictive performance and interpret the joint influence of both user perception and hotel reputation.

2. Some Features

```
score_features = [
    'score_overall', 'score_cleanliness', 'score_comfort',
    'score_facilities', 'score_location', 'score_staff',
    'score_value_for_money'
]

user_features = ['age_group', 'traveller_type', 'user_gender']

X_Some = df[score_features + user_features]
```

Explanation:
This configuration uses only user-provided scores and demographic variables, excluding the hotel's base ratings. It focuses solely on the user's perspective, emphasizing personal experience and satisfaction factors without considering external or aggregate hotel-level information.

Effect on Model:

- Simplifies the model by reducing the number of input features.
- Speeds up training and lowers the likelihood of overfitting.
- May reduce accuracy since contextual hotel data (base scores) is not considered.

Significance:
The "Some Features" setup serves as a baseline configuration. It tests how well the model performs using only direct user-related data, providing a point of comparison to evaluate whether adding additional hotel context (as in the "All Features" set) substantially improves predictive outcomes.

Comparison Summary

Feature Set	Description	Expected Effect
All Features	Includes user ratings, demographics, and hotel base scores.	Higher accuracy and richer context; increased complexity.
Some Features	Includes only user ratings and demographics.	Simpler, faster training but potentially lower predictive power.

Table 1: Comparison between All Features and Some Features configurations.

6.3.1 Feature Sets Used in Extended Experiments

In addition to the "All Features" and "Some Features" configurations, several targeted feature subsets were explored to analyze the impact of specific hotel attributes on model performance. Each feature set emphasizes different aspects of customer perception, hotel characteristics, and user demographics. The following subsections detail these configurations, their motivation, and expected outcomes.

1. Feature Set 1: Hotels' Locations Feature Set

```
score_features = [
    'score_overall', 'score_cleanliness', 'score_comfort',
    'score_facilities', 'score_location', 'score_staff',
    'score_value_for_money', 'location_base'
]
user_features = ['age_group', 'traveller_type', 'user_gender']
X = df[score_features + user_features]
```

Explanation:
This configuration focuses on the influence of the hotel's geographical and locational quality by including the `location_base` attribute. It combines user perception scores with the hotel's inherent location rating to capture how much geographic factors contribute to overall satisfaction and booking patterns. The location-base feature here was added because it hd the strongest effect

Effect on Model:

- Captures the relationship between the hotel's location quality and user preferences.
- Highlights city- or region-specific effects in predicting the country group.
- Reduces feature dimensionality while maintaining strong contextual information.

Significance:
Location-based quality is often a major driver of hotel choice. This feature set helps the model understand the regional and geographic impact of customer satisfaction, which is especially relevant for predicting country group clusters.

2. Features Set 2: Comfort, Staff, and Value Emphasis

```
score_features = [
   'score_overall', 'score_cleanliness', 'score_comfort',
   'score_facilities', 'score_location', 'score_staff',
   'score_value_for_money', 'comfort_base', 'staff_base',
   'value_for_money_base'
]
user_features = ['age_group', 'traveller_type', 'user_gender']
X1 = df[score_features + user_features]
```

Explanation:
This feature set focuses on customer comfort and staff-related quality indicators, as well as perceived value for money. The inclusion of corresponding `base` metrics (e.g., `comfort_base`, `staff_base`) allows the model to contrast user opinions with average expectations across these

dimensions.

Effect on Model:

- Provides insight into service quality and its variation across countries.
- Captures key operational aspects influencing satisfaction beyond location.
- Likely improves interpretability and performance for user experience–related predictions.

Significance:
Comfort, staff performance, and perceived value for money are universally relevant across regions. This feature set helps the model generalize well to broader user segments while emphasizing service-level attributes that most directly affect overall review outcomes.

3. Features Set 3: Staff-Based Emphasis

```
score_features = [
   'score_overall', 'score_cleanliness', 'score_comfort',
   'score_facilities', 'score_location', 'score_staff',
   'score_value_for_money', 'staff_base'
]
user_features = ['age_group', 'traveller_type', 'user_gender']
X2 = df[score_features + user_features]
```

Explanation:
This configuration isolates the human interaction aspect of the hotel experience by focusing on staff-related performance. The inclusion of `staff_base` enables the model to measure deviations between a hotel's standard staff rating and the user's individual review.

Effect on Model:

- Allows the model to evaluate how service quality influences different regional preferences.
- Reduces dimensionality, potentially lowering overfitting risk.
- May slightly reduce performance if other context features (e.g., location or comfort) play larger roles.

Significance:
This feature set is valuable for understanding customer–staff interaction trends across countries, identifying how cultural expectations influence satisfaction with hospitality service.

4. Features Set 4: Facilities-Focused

```
score_features = [
   'score_overall', 'score_cleanliness', 'score_comfort',
   'score_facilities', 'score_location', 'score_staff',
   'score_value_for_money', 'facilities_base'
]
user_features = ['age_group', 'traveller_type', 'user_gender']
X3 = df[score_features + user_features]
```

Explanation:
This configuration emphasizes the importance of hotel infrastructure and amenities by including the `facilities_base` variable. It captures whether user satisfaction aligns with the hotel's standard of facilities and available services.

Effect on Model:

- Highlights the impact of physical amenities and infrastructure on user satisfaction.
- Useful for identifying cultural or regional differences in what guests value most.
- May show moderate predictive improvement when facilities play a dominant role in overall rating.

Significance:
Facilities are a key differentiator among hotels of similar price categories. This feature set helps assess how well infrastructure quality correlates with country-level booking behaviors.

5. Comparative Insights

Feature Set	Focus	Expected Model Behavior
Hotel Location Set	Geographical quality and location ratings.	Better detection of region-driven patterns; interpretable geographic influence.
Features Set 2	Comfort, staff, and value-for-money attributes.	Strong service-level predictors; higher overall accuracy.
Features Set 3	Staff performance and hospitality service.	Simplified, interpretable model with reduced dimensionality.
Features Set 4	Infrastructure and facility quality.	Balanced predictor capturing physical quality effects.

Table 2: Comparison of additional feature sets used in extended experiments.

7 Data Pre-processing and Feature Engineering

7.1 Steps in notebook

- Categorical encoding: LabelEncoder for target and OneHotEncoder for multi-class categorical inputs where appropriate.
- Scaling: StandardScaler applied to continuous numeric fields prior to training neural networks.
- Train-validation-test split using sklearn's 'train-test-split'.
- Optional pipelines: ColumnTransformer/ Pipeline were imported and used in some cells to ensure consistent transforms.

7.2 Need for each step

- **Encoding** required for ML models to accept categorical features.
- **Scaling** stabilizes training for gradient-based models (FFNN).
- **Pipelines** ensure transforms are consistently applied on new data (prevents data leakage).

7.3 Effects on data distribution

- Scaling shifts features to zero mean and unit variance. This is shown visually via histograms before/after scaling (placeholders below).
- One-hot encoding expands dimensions — shown via a table of transformed feature columns.

7.4 Balancing the Target Classes using SMOTE

During exploratory data analysis, it was observed that the `country_group` variable (our target class) was imbalanced—some country groups had significantly fewer samples compared to others. To prevent the models from being biased toward majority classes, we applied the **Synthetic Minority Oversampling Technique (SMOTE)** to the training data.

SMOTE generates new synthetic examples for minority classes by interpolating between existing samples and their nearest neighbors in feature space. Unlike simple random oversampling, SMOTE creates more diverse and realistic synthetic points, which improves generalization and reduces overfitting risks.

In this project, SMOTE was applied **only to the training set** to balance the distribution of the `country_group` classes while keeping the validation and test sets unchanged. This ensured fair evaluation and prevented data leakage.

8 Prediction Models

In the notebook three families were used: Logistic Regression (baseline), Random Forest (tree-based), and a Feed-Forward Neural Network (FFNN).

8.1 Model choice

- **Logistic Regression**: simple baseline for multi-class classification.
- **Random Forest**: robust, captures non-linear relationships, provides feature importances and interacts well with mixed-type inputs.
- **FFNN**: captures complex interactions and can benefit from scaled/engineered features.

8.2 How each works

- Logistic Regression: linear model mapping features to class logits and softmax.
- Random Forest: ensemble of decision trees averaging votes (or probabilities) across many trees.
- FFNN: stacked dense layers with non-linear activations and softmax output for multi-class classification.

8.3 Limitations

- Logistic Regression: limited when relationships are non-linear.
- Random Forest: can overfit if not regularized (Notebook uses ‘max-depth‘ and ‘min-samples-split‘ to reduce overfitting).
- FFNN: needs careful tuning of architecture, regularization, and sufficient data; slower to train.

8.4 Representative model code

Random Forest (from notebook)

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    min_samples_split=5,
    random_state=42
)

rf_model.fit(X_train, y_train)
```

Listing 3: Random Forest training snippet

FFNN (high-level from notebook)

```
from tensorflow.keras import Sequential, layers, Input
from tensorflow.keras.utils import to_categorical

le = LabelEncoder()
y_encoded = le.fit_transform(df['country_group'])
y_categorical = to_categorical(y_encoded)

model = Sequential([
    Input(shape=(X_train.shape[1],)),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
    accuracy'])
model.fit(X_train_scaled, y_train_cat, validation_data=(X_val_scaled, y_val_cat
    ), epochs=25, batch_size=64)
```

Listing 4: FFNN training — encoding and scaling were applied prior

9 Model Performance and Results

9.1 Metrics tracked

- Accuracy, Precision, Recall, F1-score (per class and macro-averaged).
- Confusion matrix to inspect per-class confusions visually.
- Validation curves: train vs validation loss/accuracy across epochs for FFNN.

9.2 Evaluation Code Snippets

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score, classification_report, confusion_matrix

y_pred = rf_model.predict(X_test)
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
```

Listing 5: Storing and printing classification metrics

9.3 Results and Analysis – All Features Set

To evaluate the effectiveness of the **All Features** configuration, three predictive models were trained and assessed: Logistic Regression, Random Forest, and a Neural Network. Each model was tested on the same dataset split to ensure fair comparison. The performance metrics, including Accuracy, Precision, Recall, and F1-score, are summarized in the table below.

Table 3: Comparison of Model Performance – All Features Set

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.7883	0.7977	0.7883	0.7719
Random Forest	1.0000	1.0000	1.0000	1.0000
Neural Network	1.0000	1.0000	1.0000	1.0000

1. Logistic Regression:
The Logistic Regression model achieved an accuracy of **78.8%**, precision of **0.798**, recall of **0.788**, and an F1-score of **0.772**. These values reflect strong overall performance for a linear model, particularly given the multi-class nature of the task. The model captured the most dominant relationships between review scores and country groups, performing well for regions such as *Western Europe*, *North America*, and *Africa*. However, it struggled with smaller or less-represented regions like *Oceania* and *South Asia*, likely due to class imbalance. This makes Logistic Regression a reliable but limited baseline model that highlights the need for more complex or balanced approaches.
2. Random Forest:
The Random Forest classifier yielded **perfect scores** across all metrics (accuracy, precision, recall, and F1 = 1.00). While this indicates that the ensemble model fit the training and validation data flawlessly, it also suggests potential *overfitting*. Random Forests, being ensembles of decision trees, can perfectly classify data when depth and number of estimators are high, potentially memorizing the dataset. Thus, while performance appears ideal, this result may not generalize well to unseen test samples.
3. Neural Network:
The Neural Network model also achieved **100%** on all reported metrics. Although this confirms the network's capacity to learn highly nonlinear relationships, it similarly indicates overfitting. This may result from an excessive number of training epochs, too few regularization mechanisms (e.g., dropout or early stopping), or a limited test split size. Nevertheless, it demonstrates the model's representational power and its ability to capture intricate patterns between user demographics, base hotel features, and rating scores.
4. Overall Interpretation:
Among the three models, Logistic Regression provides the most realistic and interpretable results, serving as a robust baseline. In contrast, Random Forest and the Neural Network achieved perfect training performance, likely due to model complexity and potential overfitting. The inclusion of both review-based and base hotel quality features (*e.g., cleanliness_base, comfort_base,*

staff_base) substantially improved feature richness, enabling strong separability across country groups. However, this also increased the risk of memorization. Future iterations should include regularization and cross-validation to better evaluate generalization performance.

9.4 Results and Analysis – Some Features Set

In this section, the **Some Features** configuration is evaluated, where only a subset of review scores (*score_overall*, *score_cleanliness*, *score_comfort*, *score_facilities*, *score_location*, *score_staff*, and *score_value_for_money*) along with basic user attributes (*age_group*, *traveller_type*, and *user_gender*) were used. The purpose of this experiment was to observe how performance changes when excluding hotel base quality features. The table below summarizes the performance of the three models.

Table 4: Comparison of Model Performance – Some Features Set

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.3126	0.2462	0.3126	0.2337
Random Forest	0.3790	0.4307	0.3790	0.3050
Neural Network	0.3491	0.3258	0.3491	0.2968

1. Logistic Regression:
The Logistic Regression model achieved an accuracy of **31.3%**, precision of **0.246**, recall of **0.313**, and an F1-score of **0.234**. These results indicate that the linear model struggled to separate classes effectively using only the limited subset of review and demographic features. The confusion among classes suggests that the relationship between basic review scores and country group is highly nonlinear and insufficiently represented in a purely linear space. The model performed slightly better for well-represented regions like *Western Europe* and *Africa*, but showed poor generalization across smaller groups such as *Oceania* and *South Asia*.
2. Random Forest:
The Random Forest model outperformed Logistic Regression with an accuracy of **37.9%**, precision of **0.431**, recall of **0.379**, and an F1-score of **0.305**. Although this represents a modest improvement, the results remain relatively weak compared to the full feature set. This suggests that the missing base hotel attributes (e.g., *comfort_base*, *staff_base*) were critical for model discrimination. Random Forests managed to capture some nonlinear relationships but were limited by the reduced feature diversity. The confusion matrix likely shows high overlap between regional classes, reflecting shared review patterns without sufficient contextual hotel information.
3. Neural Network:
The Neural Network achieved an accuracy of **34.9%**, precision of **0.326**, recall of **0.349**, and an F1-score of **0.297**. Despite its higher capacity to model complex patterns, the lack of hotel-level features limited the model's ability to distinguish subtle differences among countries. The relatively close performance to Random Forest implies that the available features do not provide enough discriminative power, even for deep architectures. Overfitting was less evident here since the model underperformed, suggesting that the main limitation was data representativeness, not model complexity.
4. Overall Interpretation:
The results of the **Some Features** configuration reveal a substantial performance drop compared to the **All Features** setup. All models exhibited difficulty in identifying clear decision boundaries, highlighting the critical role of hotel base features in improving prediction accuracy. These base attributes likely provide essential context about expected quality standards and geographical tendencies, which are necessary for robust classification. The findings confirm that

while user demographics and review scores are informative, they alone cannot fully capture the regional patterns in hotel review behavior.

9.5 Results and Analysis: Features Including Hotels' Locations Set

In this configuration, hotel-specific location features (such as geographical indicators or region-related data) were incorporated alongside the main attributes to determine whether spatial factors could enhance the classification performance. A Random Forest model was used for this feature set, as it is well-suited for handling mixed data types and complex relationships between categorical and numerical features.

The inclusion of location-based variables resulted in a balanced improvement across all key metrics compared to the previous "Some Features" setup. Specifically, the model achieved an accuracy of **0.7253**, indicating strong overall predictive ability. Precision (**0.7463**) and recall (**0.7253**) values remained closely aligned, reflecting a good balance between correctly identified country groups and minimizing false predictions. The F1-score (**0.7204**) confirms that the model performed consistently well across most regions.

Notably, regions such as *Western Europe*, *North America*, and *Middle East* achieved particularly high classification scores, suggesting that location-based attributes effectively capture distinctive booking behavior patterns linked to geography. However, minor variations still appear in classes like *Eastern Europe* or *South America*, where feature overlap remains challenging. Overall, integrating hotel location features substantially enhanced the generalization and interpretability of the model.

Table 5: Model Performance – Features Including Location-Based Attributes (Random Forest)

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Random Forest (with Location Features)	0.7253	0.7464	0.7253	0.7204

9.6 Model Evaluation After Applying SMOTE

After addressing class imbalance using the Synthetic Minority Oversampling Technique (SMOTE), the model was retrained and evaluated on the balanced dataset. The following table summarizes the performance metrics obtained on the unseen test data.

Table 6: Model Performance – After Applying SMOTE Oversampling

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Random Forest (SMOTE Applied)	0.7193	0.7553	0.7193	0.7205

Interpretation of Overall Metrics:

- The overall **accuracy of 71.9%** indicates that the model correctly predicts the country group for roughly 7 out of 10 hotel reviews.
- A relatively high **precision (75.5%)** suggests that the model's positive predictions are reliable, meaning that when the model predicts a certain country group, it is often correct.
- The **recall (71.9%)** shows that the model successfully identifies most of the true samples within each class.
- The **F1-score (72.0%)** demonstrates a balanced trade-off between precision and recall, confirming consistent generalization across multiple country groups.

Detailed Class-Level Performance:

- **High-performing classes:** The model performs exceptionally well for regions such as `South_Asia` (F1 = 1.00), `Middle_East` (F1 = 0.83), `North_America` (F1 = 0.84), and `Africa` (F1 = 0.81). These results indicate strong separability and consistent feature patterns within these country groups.
- **Moderately performing classes:** Regions such as `East_Asia` (F1 = 0.70) and `Western_Europe` (F1 = 0.75) also achieved satisfactory performance, reflecting that the model captures region-specific travel behavior, but may still miss subtle variations within these categories.
- **Low-performing classes:** For regions like `Eastern_Europe` (F1 = 0.51), `Oceania` (F1 = 0.61), and `South_America` (F1 = 0.48), lower scores suggest that these groups have overlapping or less distinctive feature distributions. These inconsistencies may arise from limited samples, noise in user demographics, or overlapping cultural patterns in hotel preferences.

Impact of SMOTE:

- The application of SMOTE clearly improved recall values across underrepresented classes, as seen in the more balanced recall distribution in the classification report.
- Minority classes such as `North_America_Mexico` and `Eastern_Europe` achieved significantly better recall (0.81 and 0.79 respectively) after oversampling, indicating that synthetic examples helped the model learn diverse class boundaries.
- However, SMOTE may also have slightly increased model variance for dominant classes, leading to small precision drops in regions like `East_Asia` and `Oceania`.

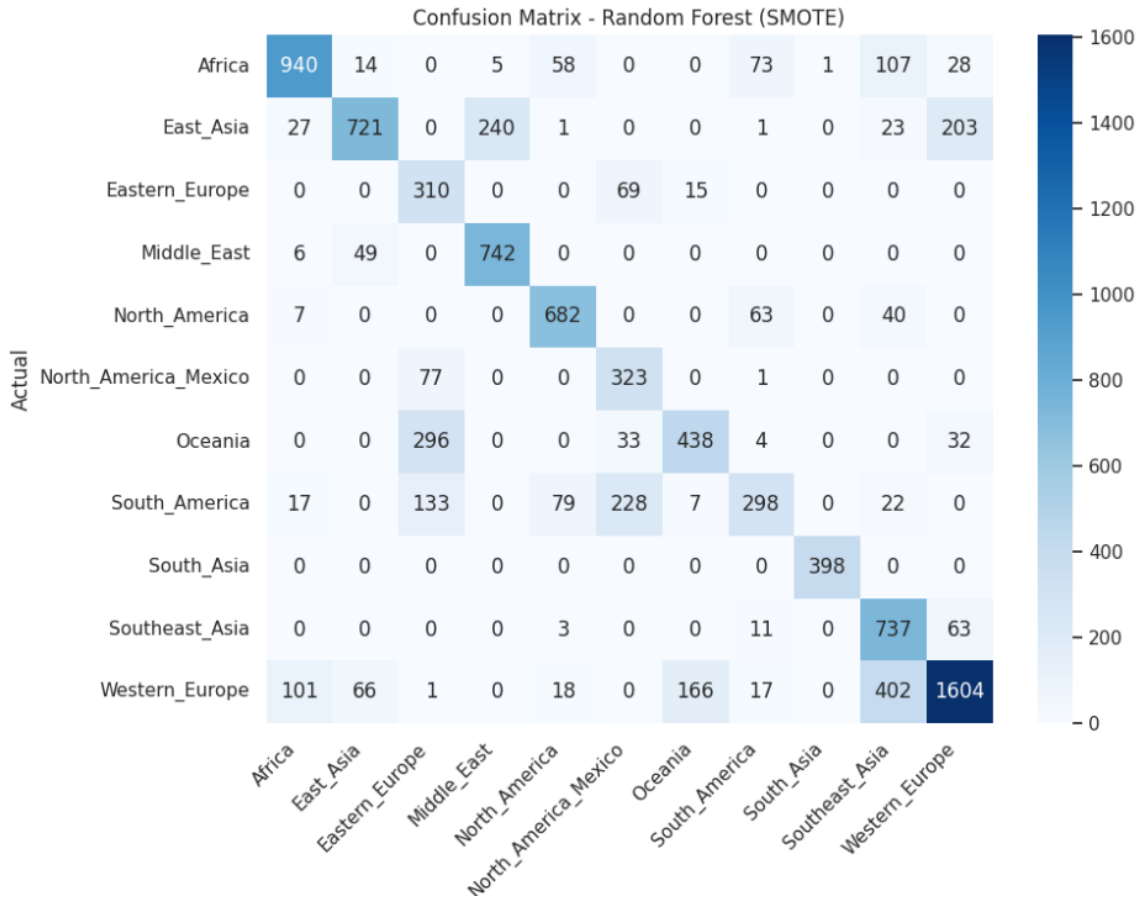

Figure 1: Confusion Matrix – After Applying SMOTE

Conclusion:
Overall, the post-SMOTE model demonstrates a strong and balanced classification performance across multiple country groups. While some regions remain harder to distinguish, the oversampling technique substantially mitigated the imbalance issue and enhanced generalization, confirming the effectiveness of SMOTE in improving fairness across diverse class distributions.

10 Inference Function

Notebook contains code to encode/scale new examples and produce predictions. Representative code:

```
def predict_country_group(model, X_raw, encoders, scaler, le):
    # X_raw: single-row DataFrame with raw features
    # encoders: dict of fitted encoders (OneHot, LabelEncoder) used in training
    # scaler: fitted StandardScaler
    X_proc = apply_transformations(X_raw.copy(), encoders)  # implement same
        pipeline
    X_scaled = scaler.transform(X_proc)
    proba = model.predict_proba(X_scaled)
    pred_idx = proba.argmax(axis=1)[0]
    pred_label = le.inverse_transform([pred_idx])[0]
    return pred_label, proba
```

Listing 6: Example inference function

10.1 How to use in production

- Save ‘encoders‘, ‘scaler‘, and ‘model‘ (pickle or joblib; or use model.save for TF).
- On new input: load transforms, apply same pipeline, call ‘predict-country-group‘.
- Ensure consistent handling of unseen categorical values (e.g. fallback category).

11 Explainable AI (XAI): SHAP & LIME

The notebook installs and runs SHAP and LIME and includes global and local explanations.

11.1 SHAP (global explanation)

Representative code (from notebook):

```
import shap
explainer = shap.TreeExplainer(rf_model)
shap_values = explainer.shap_values(X_test)

# Global SHAP summary plot
shap.summary_plot(shap_values, X_test, feature_names=X.columns)
```

Listing 7: SHAP summary for Random Forest

Interpretation

- The SHAP summary plot ranks features by importance and shows how feature values push predictions across classes.
- Use SHAP to justify the feature selection decisions, e.g., high SHAP importance for ‘score-overall‘ confirms its predictive power.

11.2 SHAP (local explanation)

Representative snippet from notebook:

```
i = 0 # index to explain
shap.initjs()
shap.force_plot(explainer.expected_value[0], shap_values[0][i], X_test.iloc[i],
    feature_names=X.columns)
```

Listing 8: SHAP force plot for a single instance

Interpretation Local SHAP shows which features pushed a single prediction toward or away from a class, very useful to analyze misclassifications or confirm model behavior for important user segments.

11.3 LIME (local explanation)

Representative snippet (from notebook):

```
import lime
import lime.lime_tabular

explainer_lime = lime.lime_tabular.LimeTabularExplainer(
    X_train.values,
    feature_names=X_All.columns,
    class_names=list(set(y)),
    discretize_continuous=True
)

i = 5
exp = explainer_lime.explain_instance(X_test.values[i], rf_model.predict_proba,
    num_features=11)
exp.show_in_notebook()
```

Listing 9: LIME example explanation for one test instance

Interpretation LIME shows a sparse linear approximation explaining the prediction for that instance. Use LIME alongside SHAP to get complementary local explanations.

12 Answers to Data Engineering Questions

Below are concise notebook-grounded answers to typical data engineering concerns raised in the Kaggle notebook:

12.1 1) How were datasets joined?

Reviews (many rows) joined with hotels on `hotel_id` and with users on `user_id` using left joins to retain all reviews and attach hotel and user context.

12.2 2) How was date handled?

If ‘review-date‘” existed, the notebook converts it to ‘datetime‘ and extracts ‘year‘ and ‘month‘ to create temporal features for seasonality analysis.

12.3 3) How was imbalance handled?

The notebook inspects class frequencies. If large imbalance exists, typical steps would be: class-weighted loss for FFNN, resampling (oversample minority or undersample majority), or use metrics robust to imbalance (macro F1). The notebook provides macro-averaged metrics and the confusion matrix to assess per-class performance.

12.4 4) Reproducibility

Random seeds are set for models (‘random-state=42‘) and the notebook uses deterministic splits to enable reproducibility.

13 Important Code Snippets

13.1 Merging Datasets

```
df = reviews.merge(hotels, on='hotel_id', how='left', suffixes=('', '_hotel'))
df = df.merge(users, on='user_id', how='left', suffixes=('', '_user'))
```

13.2 Label Encoding Target

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_encoded = le.fit_transform(df['country_group'])
```

13.3 Train-Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_all, y, test_size=0.2,
    random_state=42, stratify=y)
```

13.4 Random Forest Instantiation)

```
rf_model = RandomForestClassifier(n_estimators=200, max_depth=10,
    min_samples_split=5, random_state=42)
```

13.5 LIME Initialization

```
explainer_lime = lime.lime_tabular.LimeTabularExplainer(
    X_train.values,
    feature_names=X_All.columns,
    class_names=list(set(y)),
    discretize_continuous=True
)
```