Mohammod Abdullah Bin Hossain

Task Assignment

Goal: Predict the price of an Airbnb listing

Overview of this Project

The project is centered on predicting the price of an Airbnb listing by leveraging various features in the dataset. It begins with an in-depth Exploratory Data Analysis (EDA), where the structure and distribution of the data are thoroughly examined. This includes identifying missing values, visualizing the distribution of numerical features such as ratings, reviews, and price, and analyzing the relationships between features through correlation matrices. A detailed investigation is carried out on categorical variables like host names and countries, exploring their impact on the target variable, price. The data cleaning phase follows, focusing on handling missing values and outliers. Missing values in key features, such as host names and check-in/check-out times, are carefully imputed, and outliers in the price and other numerical features are managed using the Interquartile Range (IQR) method to avoid distorting the model's predictions.

For feature engineering, new insightful variables like price per guest and price per bedroom are created to offer deeper insights into pricing patterns. The price feature itself undergoes a log transformation to reduce skewness and stabilize variance, making the distribution more suitable for modeling. Moreover, to transform the continuous price variable into a classification problem, the price is discretized into 10 bins based on its log-transformed values. These bins are then labeled as categories, simplifying the task into a classification problem where the model predicts a price category rather than an exact value. This approach allows the model to focus on price ranges, which is beneficial given the diverse pricing strategies in the Airbnb listings.

Principal Component Analysis (PCA) is applied for dimensionality reduction, capturing 98% of the variance, thus reducing the complexity of the dataset while retaining crucial information. Features are scaled using Z-score standardization to ensure all variables contribute equally to the model. The development of the model follows, where two different Artificial Neural Network (ANN) architectures are tested and evaluated. Hyperparameter optimization, cross-validation, and performance metrics such as accuracy, precision, recall, F1-score, and ROC curves are used to assess the models' effectiveness.

To enhance predictive accuracy, an ensemble approach combining the strengths of ANN and Random Forest models is employed. The predictions from both models are stacked and passed to a meta-model, XGBoost, which refines the results, offering a more robust and accurate prediction. This ensemble method helps mitigate individual model weaknesses, ensuring better generalization and stability. The final submission includes a comprehensive workflow of the entire process, from data preparation to model evaluation, with clear explanations of each step taken to improve performance.

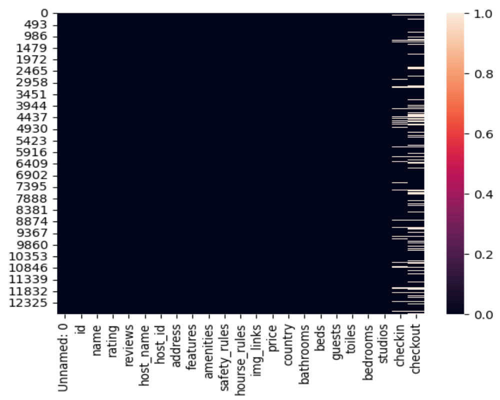
Techniques perform to complete this project

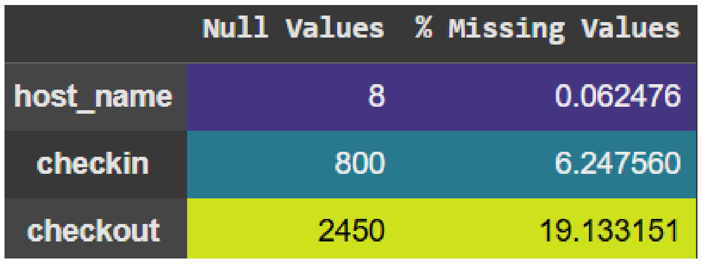
Exploratory Data Analysis (EDA)

1. Checking for Missing Values

A heatmap was used to visually identify missing values across the entire dataset. This helps in quickly spotting features with missing data.

The missing values were further summarized in a DataFrame showing the count and percentage of missing values for each feature, and only features with missing values were displayed for clarity.





	Null Values	% Missing Values
host_name	8	0.062476
checkin	800	6.247560
checkout	2450	19.133151

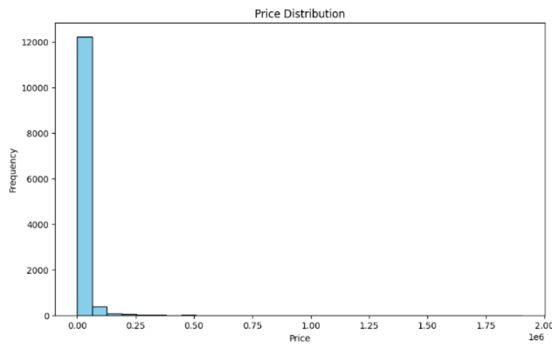
2. Replacing 'New' in Rating

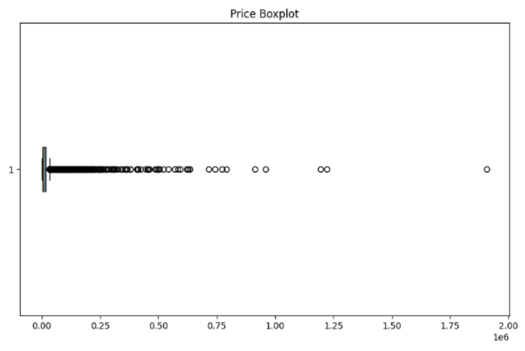
The rating feature had some entries marked as 'New'. These were replaced with 0 to treat them as valid numerical entries, allowing the feature to be processed correctly.

3. Price Distribution Visualizations

Histogram of price: A histogram was plotted to visualize the distribution of prices across all listings. This gives insight into the frequency of different price ranges.

Boxplot of price: A boxplot was created to visually inspect the spread and identify potential outliers in the price data.

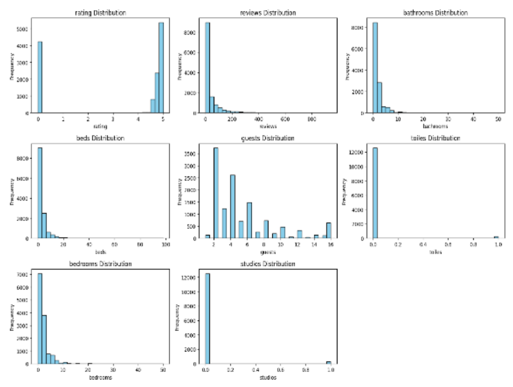


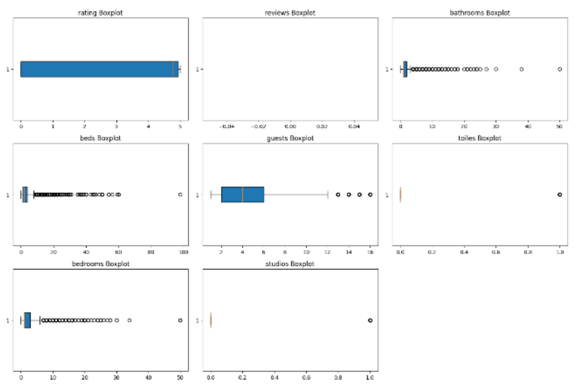


4. Data Type and Visualizing Distribution of Numerical Features

- The data types of numerical features such as rating, reviews, bathrooms, beds, guests, toiles, bedrooms, and studios were inspected to ensure they were appropriate for analysis.
- Some features, like rating and reviews, were converted to float and int types, respectively, for consistency and further analysis.

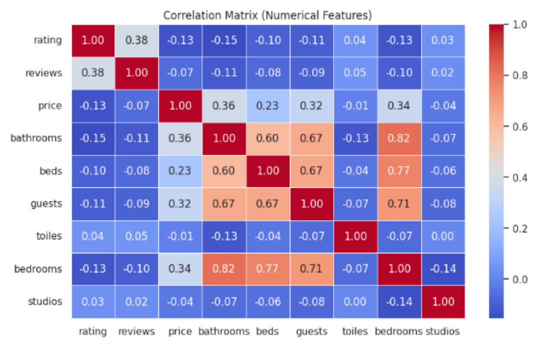
- **Histograms:** A set of histograms was created for several numerical features (rating, reviews, bathrooms, etc.) to examine their distributions. This helps in understanding the spread of data and detecting skewness or potential data issues.
- **Boxplots:** Boxplots for the same numerical features were plotted to visualize the spread, central tendency, and potential outliers.





5. Correlation Matrix Between Price and Other Numerical Features

- A correlation matrix was calculated and visualized using a heatmap. This allowed the identification of strong correlations between price and other numerical features like rating, reviews, bathrooms, etc. It helps understand which features are most related to price and should be considered for predictive modeling.

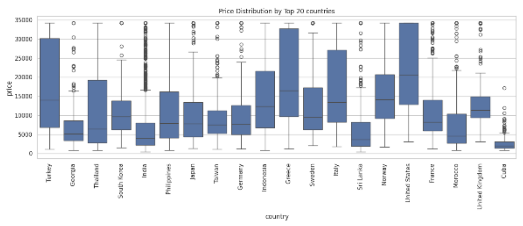


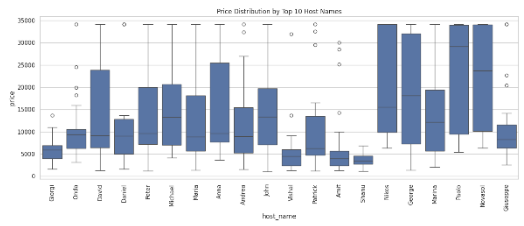
	rating	reviews	price	bathrooms	beds	guests	toiles	bedrooms	studios
rating	1.00	0.38	-0.13	-0.15	-0.10	-0.11	0.04	-0.13	0.03
reviews	0.38	1.00	-0.07	-0.11	-0.08	-0.09	0.05	-0.10	0.02
price	-0.13	-0.07	1.00	0.36	0.23	0.32	-0.01	0.34	-0.04
bathrooms	-0.15	-0.11	0.36	1.00	0.60	0.67	-0.13	0.82	-0.07
beds	-0.10	-0.08	0.23	0.60	1.00	0.67	-0.04	0.77	-0.06
guests	-0.11	-0.09	0.32	0.67	0.67	1.00	-0.07	0.71	-0.08
toiles	0.04	0.05	-0.01	-0.13	-0.04	-0.07	1.00	-0.07	0.00
bedrooms	-0.13	-0.10	0.34	0.82	0.77	0.71	-0.07	1.00	-0.14
studios	0.03	0.02	-0.04	-0.07	-0.06	-0.08	0.00	-0.14	1.00

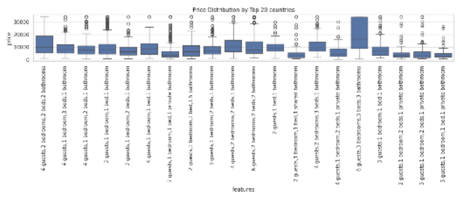
6. Visualizing Price Distribution Across Hosts, Countries, Features, and Global Listings

Visualizing price distribution was done across various factors. A boxplot was created to visualize how prices vary across different host names, allowing the comparison of pricing distributions for top hosts and revealing whether certain hosts charge significantly more or less than others. Similarly, a boxplot was used to visualize price distributions by country, focusing on the top 20 countries with the most listings, helping to compare how pricing varies across different countries and identifying geographical pricing trends. Boxplots were also used to visualize how price varies across different amenities and features categories, helping identify any patterns or trends in pricing based on available amenities or features. Lastly, a choropleth map was created to show the distribution of

Airbnb listings across different countries, providing a visual representation of the concentration of listings in various regions and highlighting global trends and densities.



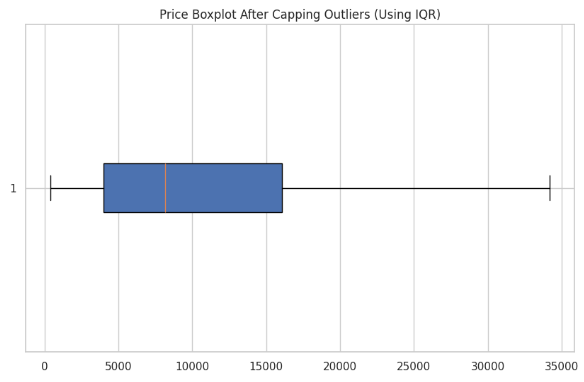




Data Handling & Cleaning

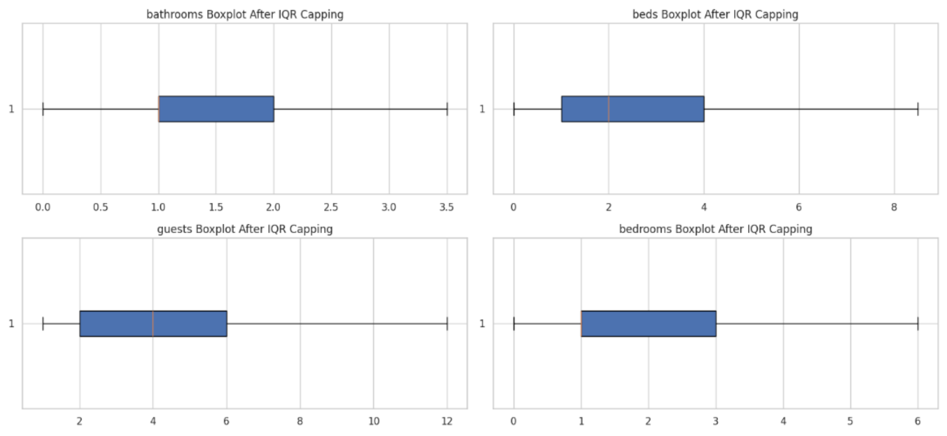
1. Handling Outliers in the price Feature

Outliers in the price feature were handled using the **Interquartile Range (IQR)** method. The lower and upper bounds were calculated, and outliers were capped by replacing values outside these bounds with the nearest valid values. This approach prevents extreme values from skewing the model while preserving the data's underlying distribution.



2. Handling Outliers in Other Numerical Features

The same **IQR method** was applied to other numerical features, such as bathrooms, beds, guests, and bedrooms. Outliers in these features were capped to ensure the model focuses on general trends and does not get biased by extreme data points.



3. Handling Missing Values

- **For host_name**, missing values were filled with 'Unknown' to provide a meaningful placeholder.
- **For checkin**, missing values were filled with the most frequent value (mode) in the column, assuming that the most common check-in time is representative.
- **For checkout**, missing values were replaced with '12:00 pm' as a default.
- **For reviews**, missing values were filled with a fixed value of 1003.

This ensures no data is lost due to missing values, and the imputation respects the structure of the dataset.

4. Creating Binary Features for Flexibility in checkin and checkout

New binary features, checkin_flexible and checkout_flexible, were created to capture whether the check-in and check-out times are flexible. This allows the model to better understand flexibility as a factor in pricing.

5. Handling Time-Related Features (checkin and checkout)

Time-related features were standardized into a numeric form by calculating the duration in minutes between the check-in and check-out times. Non-standard time formats were also processed to ensure consistency. After processing, these features were dropped to avoid redundancy.

6. Dropping Unnecessary Columns

Irrelevant columns such as 'Unnamed: 0', 'img_links', 'id', 'host_id', 'safety_rules', 'checkin', 'checkout', 'house_rules', and 'reviews' were dropped from the dataset. This process streamlined the dataset by removing redundant or non-contributory features, ensuring that only the most relevant and meaningful data was retained for modeling, which helped improve model performance and reduce unnecessary complexity.

Data Preprocessing Overview

1. Categorical Encoding (One-Hot Encoding)

- **Step Taken:** Selected categorical columns such as host_name, country, features, amenities, name, and address for One-Hot Encoding. The pd.get_dummies() function was applied to these columns, converting them into binary format.

  One-Hot Encoding ensures that categorical features are transformed into a numerical format that machine learning models can interpret. The drop_first=True parameter avoids multicollinearity by dropping the first category of each column.

2. Log Transformation of price

- **Step Taken:** A log transformation was applied to the price column using np.log1p() to handle skewed data and stabilize variance.

  Log transformation is commonly used to make a highly skewed distribution more normal. This helps the model learn better from the data and avoids issues where large **price** values dominate the learning process.

3. Discretizing the Continuous Target (price)

- **Step Taken:** The price feature was transformed into a discrete target feature (price_category) by creating 10 bins based on the log-transformed price values. The bins were then labeled as categories (1-11).

  Discretizing the continuous target variable into categories simplifies the prediction problem into classification. It allows the model to predict a range of prices instead of specific continuous values, making the task more manageable.

4. Initiative for Classifying the Continuous Price Target Variable

To convert the continuous `price` target variable into a discrete classification target, I implemented a two-step process involving **log transformation** and **binning**.

- **Log Transformation:** The `price` variable was transformed using the log1p function (`np.log1p(df_encoded['price'])`). This transformation helps handle the skewness of the price data and addresses the issue of zero prices, as `log(0)` is undefined. The `log1p` function ensures all values are positive and properly scaled, making the data more suitable for modeling.
- **Binning into Discrete Categories:** After applying the log transformation, I divided the `log_price` into **10 equal-width bins** using the `pd.cut()` function. These bins represent different price ranges and are assigned labels from 1 to 10. This transformation effectively converts the continuous target variable into discrete categories, making it suitable for classification tasks.

This approach simplifies the task of predicting price categories (such as low, medium, and high) rather than predicting a precise price, aligning with the goal of classification. It also improves model interpretability by grouping similar price values into categories, making it easier to understand the model's predictions.

5. Feature Engineering

- **Step Taken:** New features, such as price_per_guest, price_per_bed, and price_per_bedrooms, were created to provide more insights into pricing.

  These engineered features allow the model to capture relationships between price and other features (like guests, beds, and bedrooms), potentially improving predictive accuracy by offering more granular information.

6. Dimensionality Reduction with PCA

- **Step Taken:** Principal Component Analysis (PCA) was used for dimensionality reduction after scaling the data with StandardScaler. PCA components were retained to explain 98% of the variance in the data.

PCA helps reduce the complexity of the model by transforming the features into a smaller number of uncorrelated components that still capture the majority of the variance. This is particularly helpful when dealing with high-dimensional data, as it reduces noise and computational overhead.

7. Reconstructing Data from PCA Components

The original data was reconstructed from the PCA components and compared with the original dataset.This step ensures that the dimensionality reduction process retains essential information and allows verification that the PCA transformation did not lose significant data.

8. Handling Missing Values

- **Step Taken:** Missing values in the price, price_category, and log_price columns were checked. If any missing values were found, they were handled using imputation techniques.

  Handling missing values is crucial to avoid disrupting the learning process. In this case, missing data was filled based on the feature's characteristics (e.g., mode or median).

9. Feature Scaling

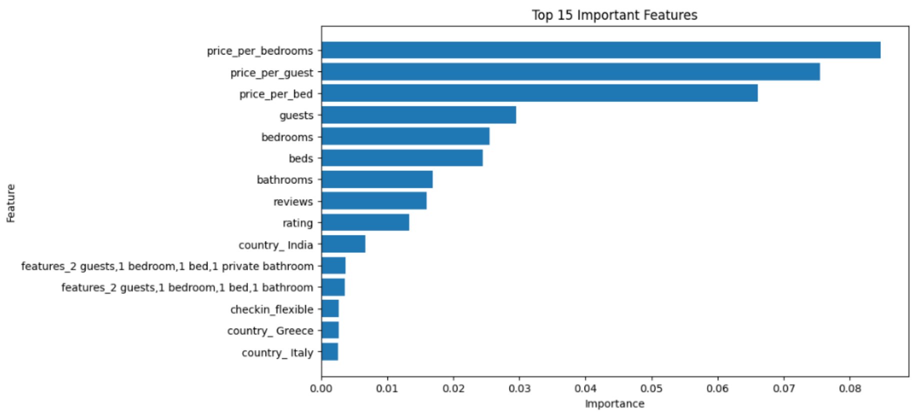
- **Step Taken:** The continuous features (**price_per_bedrooms, price_per_guest**, etc.) were scaled using **Z-score standardization** with StandardScaler to ensure all features have a mean of 0 and a standard deviation of 1.

  That Scaling the data ensures that features are on the same scale, which is important for machine learning models like PCA and others that are sensitive to the magnitude of input data.

10. Identifying Feature Importance

- **Step Taken:** A **Random Forest classifier** was used to evaluate the importance of each feature based on its contribution to predicting the target variable (price_category).

  Random Forests are powerful ensemble models that provide feature importance scores, helping to identify which features most strongly influence the target variable. This information is critical for understanding the model and selecting relevant features.



11. PCA Loadings and Feature Contribution

- **Step Taken:** The loadings from the PCA components were examined to understand how each feature contributed to the principal components.

PCA loadings reveal how strongly each original feature contributes to each principal component. This helps in identifying the most influential features after dimensionality reduction.

12. Final Evaluation of Reduced Features

- **Step Taken:** After dimensionality reduction with PCA, the top features contributing to each principal component were identified, and a summary of the overall importance of features was provided.This step ensures that the most important features are retained in the reduced dataset, allowing the model to focus on the most relevant information while ignoring less important data.

Train-Test Split & Cross-Validation

- **Train-Test Split:** The dataset is split into 80% training and 20% testing. The target variable (price_category) is one-hot encoded for multi-class classification.
- **Handling Missing Values:** Infinite values are replaced with NaN, ensuring clean data for training.
- **Feature Scaling: Standard Scaling** is applied to normalize features, ensuring equal contribution in model training.
- **Stratified K-Fold Cross-Validation:** The dataset is split into 3 folds, maintaining consistent class distribution across folds. This prevents bias and provides reliable performance evaluation.

ANN Model with Cross-Validation, Hyperparameter optimization & Evaluation

1.1. Model Architecture (ANN Model 1)

- **Input Layer:** The model starts with a dense layer of **512** neurons and uses ReLU activation. Batch normalization is applied to stabilize training.
- **Hidden Layers:** Three additional dense layers with **512, 256,** and **128** neurons, respectively, are added. Each is followed by a dropout layer (with a rate of 0.4) and batch normalization to prevent overfitting and maintain stable training.
- **Output Layer:** The final layer uses a **softmax** activation function to predict probabilities for each class, as it is a multi-class classification problem.
- **Model Compilation:** The model is compiled with the Adam optimizer **(learning rate = 0.0005)** and categorical cross-entropy loss, with accuracy as the evaluation metric.
- **Model Training:** The model is trained using the .fit() function, with the training data provided for a specified number of epochs (e.g., 50). The batch size is set to **32**, meaning the data is processed in batches of 32 samples. The model is validated after each epoch using a separate validation dataset. Training progress is displayed with verbose set to 1, allowing for real-time updates on the training process. The training and validation accuracy and loss are tracked to monitor performance and prevent overfitting.

1.2. Model Architecture (ANN Model 2)

- ✓ Input **Layer:** The model starts with a dense layer of **256** neurons and uses ReLU activation. Batch normalization is applied to stabilize the training process.
- ✓ Hidden **Layers:** Three additional dense layers are added with **128, 64**, and **32** neurons, respectively. Each hidden layer is followed by a dropout layer **(with a rate of 0.3)** and batch normalization to prevent overfitting and ensure stable training.
- ✓ Output **Layer:** The final layer uses a softmax activation function to predict probabilities for each class, as it is a multi-class classification problem.
- ✓ Model **Compilation:** The model is compiled with the Adam optimizer **(learning rate = 0.001)** and categorical cross-entropy loss, with accuracy as the evaluation metric.

- ✓ **Model Training:** The model is trained using the `.fit()` function, with the training data provided for a specified number of epochs (e.g., 45). The batch size is set to **16**, meaning the data is processed in batches of 16 samples. The model is validated after each epoch using a separate validation dataset. Training progress is displayed with a verbose set to 1, allowing for real-time updates on the training process. The training and validation accuracy and loss are tracked to monitor performance and prevent overfitting.

1.3. Ensemble Model for Classification (ANN + Random Forest)

Objective

To improve model performance by combining the predictions of multiple base models (ANN and Random Forest) using an ensemble approach. The goal is to leverage the strengths of each model and create a more robust classifier through **stacking** and a **meta-model** (XGBoost).

Steps to Build the Ensemble Model

1. **Base Models Training:**
   - **ANN Model:** A neural network with dense layers is trained on the feature set. It uses ReLU activation for hidden layers and softmax for multi-class classification.
   - **Random Forest Model:** A random forest classifier is trained on the same training data to predict the target variable.
2. **Prediction Generation:**
   - Predictions from both the **ANN** and **Random Forest** models are obtained for the test set.
   - **Class labels** are extracted using `np.argmax()` for multi-class classification.
3. **Stacking Predictions:**
   - The predictions from both models are stacked together into a new feature set, which serves as input for the **meta-model** (XGBoost).
4. **Meta-Model Training (XGBoost):**
   - The stacked predictions are used to train an **XGBoost classifier**, which learns how to best combine the outputs from the base models.
5. **Ensemble Model Evaluation:**
   - The ensemble model is evaluated on the test set. Accuracy and loss are calculated to assess performance.

Reason for Building the Ensemble Model

The decision to build the ensemble model using a **meta-model** (XGBoost) was driven by the need to combine the strengths of different base models (ANN and Random Forest) and improve the overall prediction accuracy. Base models often make different types of errors, and by stacking their predictions, we can mitigate their individual weaknesses and capitalize on their complementary strengths. The **meta-model** acts as a final decision-maker, learning how to optimally combine the outputs from both base models, effectively improving generalization and robustness. XGBoost, known for its strong performance in classification tasks, is chosen as the meta-model to refine the base model predictions, leading to a more accurate and stable ensemble model. This approach enhances performance by reducing both overfitting and bias, making the model more capable of handling diverse data patterns and providing more reliable predictions compared to individual models.

2. Training & Cross-Validation

- The model is trained using **Stratified K-Fold Cross-Validation** with 3 folds. This method ensures that each fold has a proportional distribution of the target variable classes.

- **Feature Scaling:** Before training, the features are scaled using **StandardScaler**, which normalizes them to have a mean of 0 and a standard deviation of 1.

3. Model Evaluation

- **Training & Validation Accuracies:** During each fold, training and validation accuracies are tracked and visualized. These plots help monitor the model's learning process and how well it generalizes to unseen data.
- **Training & Validation Losses:** Training and validation losses are also tracked to ensure the model is learning and not overfitting.

4. Confusion Matrix & Classification Report

- **Training Data Evaluation:** A confusion matrix and classification report for the training set are generated to evaluate how well the model performs on data it has seen.
- **Test Data Evaluation:** Similarly, the model's performance on the test set is evaluated with a confusion matrix and classification report. These metrics help understand the precision, recall, and F1-score for each class.

5. ROC AUC Curve

- The **ROC AUC curve** is plotted for each class to visualize the model's ability to distinguish between classes. The **AUC (Area Under the Curve)** provides an overall assessment of the model's classification performance, with higher values indicating better performance.

Comparison of ANN Model Architectures and Performance Metrics

Metric/Aspect	Model 1	Model 2	Random Forest	Ensemble Model
Average Train Accuracy	0.96	0.90	0.96	0.97
Average Validation Accuracy	0.95	0.93	N/A	N/A
Average Train Loss	0.13	0.24	0.03	0.03
Average Validation Loss	0.15	0.21	N/A	N/A
Test Accuracy	0.95	0.89	0.93	0.97
Test Loss	0.14	0.26	0.06	0.03

All the evaluation results, including images, curves, and confusion matrices, Explainable AI(EXAI) results are displayed in the code.