**Random Forest**

**Dataset Description:**

**Use the Glass dataset and apply the Random forest model.**

**1. Exploratory Data Analysis (EDA):**

**Perform exploratory data analysis to understand the structure of the dataset.**

**Check for missing values, outliers, inconsistencies in the data.**

**Answer:**

(.venv) PS D:\python apps> & "D:/python apps/my-streamlit-app/.venv/Scripts/python.exe" "d:/python apps/random forest/eda\_glass\_randomforest.py"

Shape of dataset: (214, 10)

--- Dataset Info ---

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 214 entries, 0 to 213

Data columns (total 10 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 RI 214 non-null float64

1 Na 214 non-null float64

2 Mg 214 non-null float64

3 Al 214 non-null float64

4 Si 214 non-null float64

5 K 214 non-null float64

6 Ca 214 non-null float64

7 Ba 214 non-null float64

8 Fe 214 non-null float64

9 Type 214 non-null int64

dtypes: float64(9), int64(1)

memory usage: 16.8 KB

None

--- Missing Values ---

RI 0

Na 0

Mg 0

Al 0

Si 0

K 0

Ca 0

Ba 0

Fe 0

Type 0

dtype: int64

Duplicate rows: 1

--- Summary Statistics ---

count mean ... 75% max

RI 214.0 1.518365 ... 1.519157 1.53393

Na 214.0 13.407850 ... 13.825000 17.38000

Mg 214.0 2.684533 ... 3.600000 4.49000

Al 214.0 1.444907 ... 1.630000 3.50000

Si 214.0 72.650935 ... 73.087500 75.41000

K 214.0 0.497056 ... 0.610000 6.21000

Ca 214.0 8.956963 ... 9.172500 16.19000

Ba 214.0 0.175047 ... 0.000000 3.15000

Fe 214.0 0.057009 ... 0.100000 0.51000

Type 214.0 2.780374 ... 3.000000 7.00000

[10 rows x 8 columns]

--- Target Value Counts (Type) ---

Type

2 76

1 70

7 29

3 17

5 13

6 9

Name: count, dtype: int64

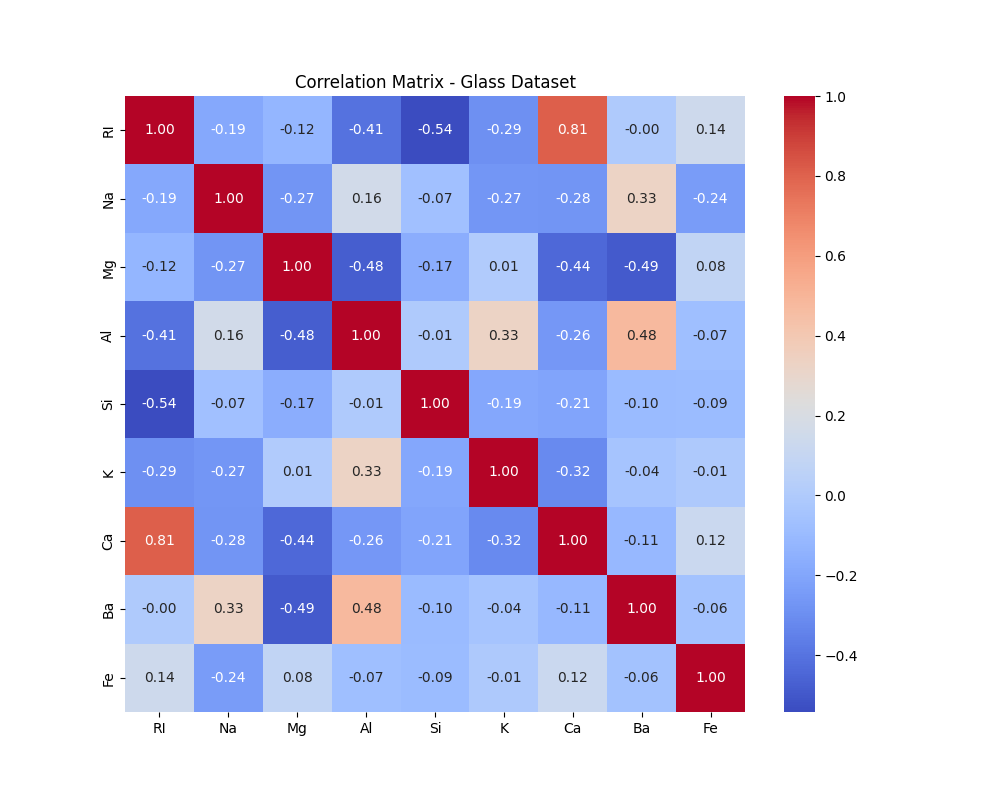
EDA completed successfully.

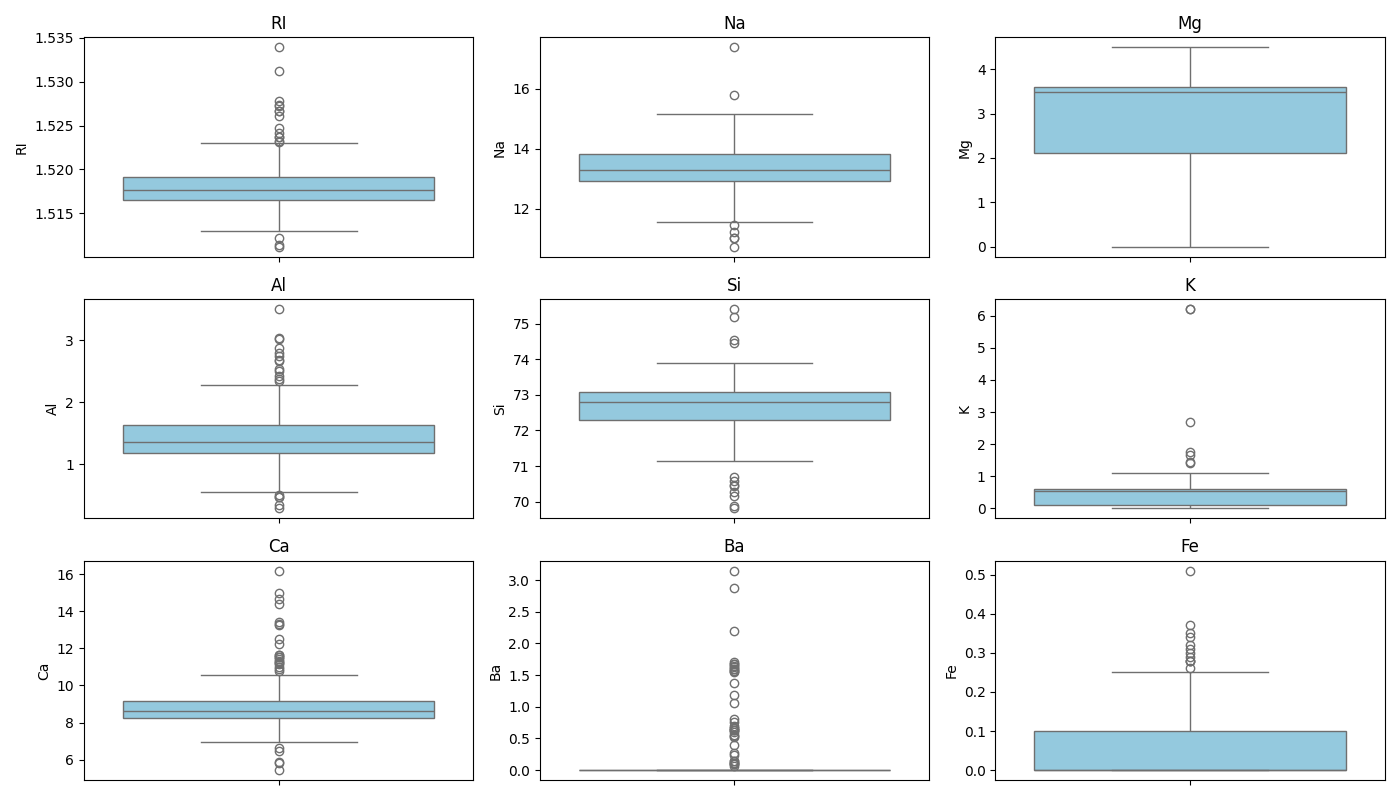
Plots saved in: D:\DATA SCIENCE\ASSIGNMENTS\14 random forest\Random Forest

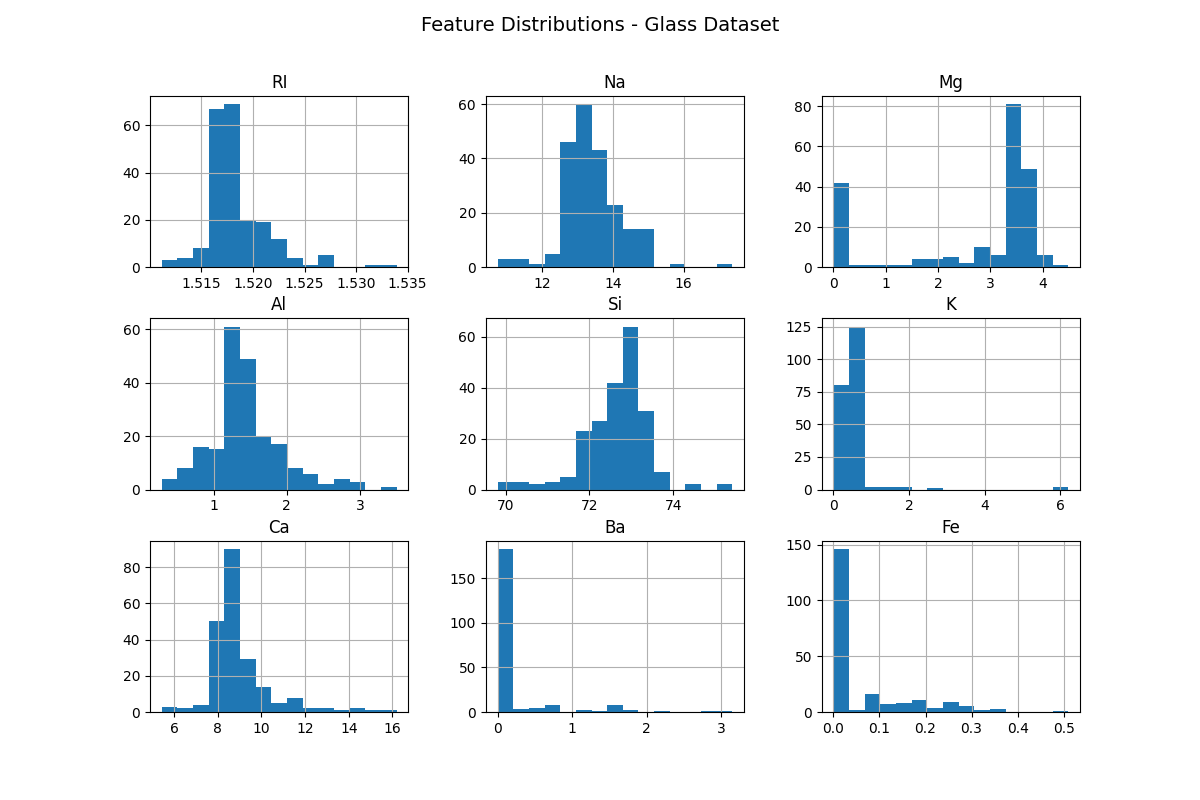
Files created:

- histograms.png

- boxplots.png

 - correlation\_matrix.png





**2: Data Visualization:**

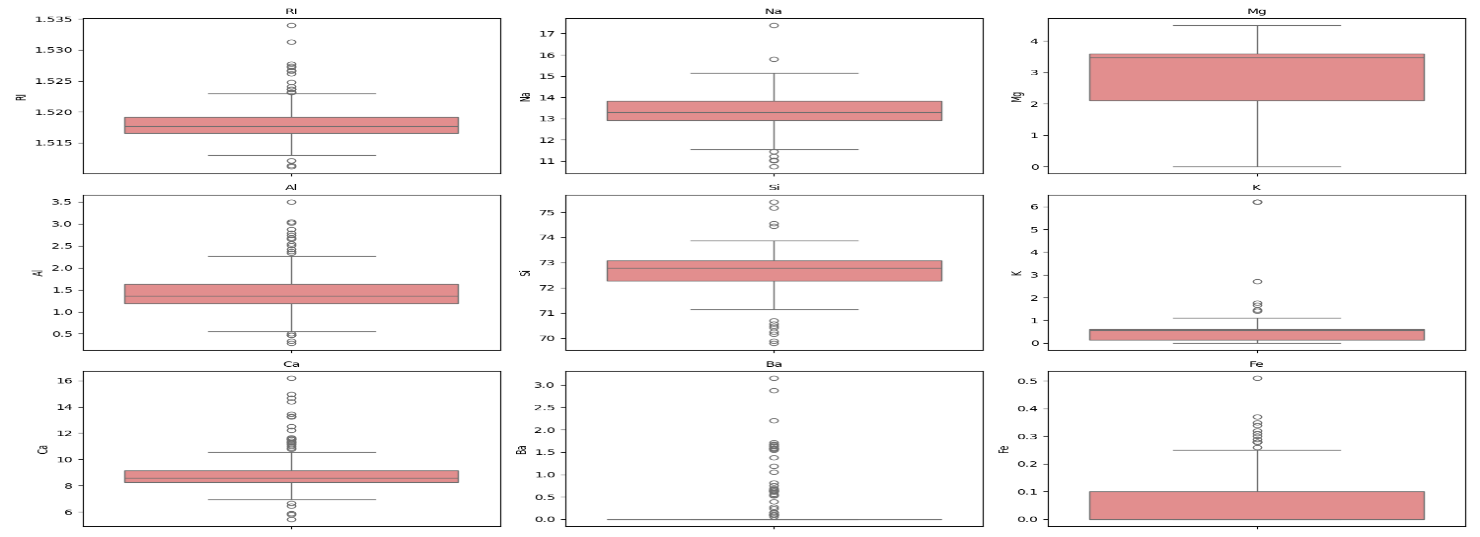
**Create visualizations such as histograms, box plots, or pair plots to visualize the distributions and relationships between features.**

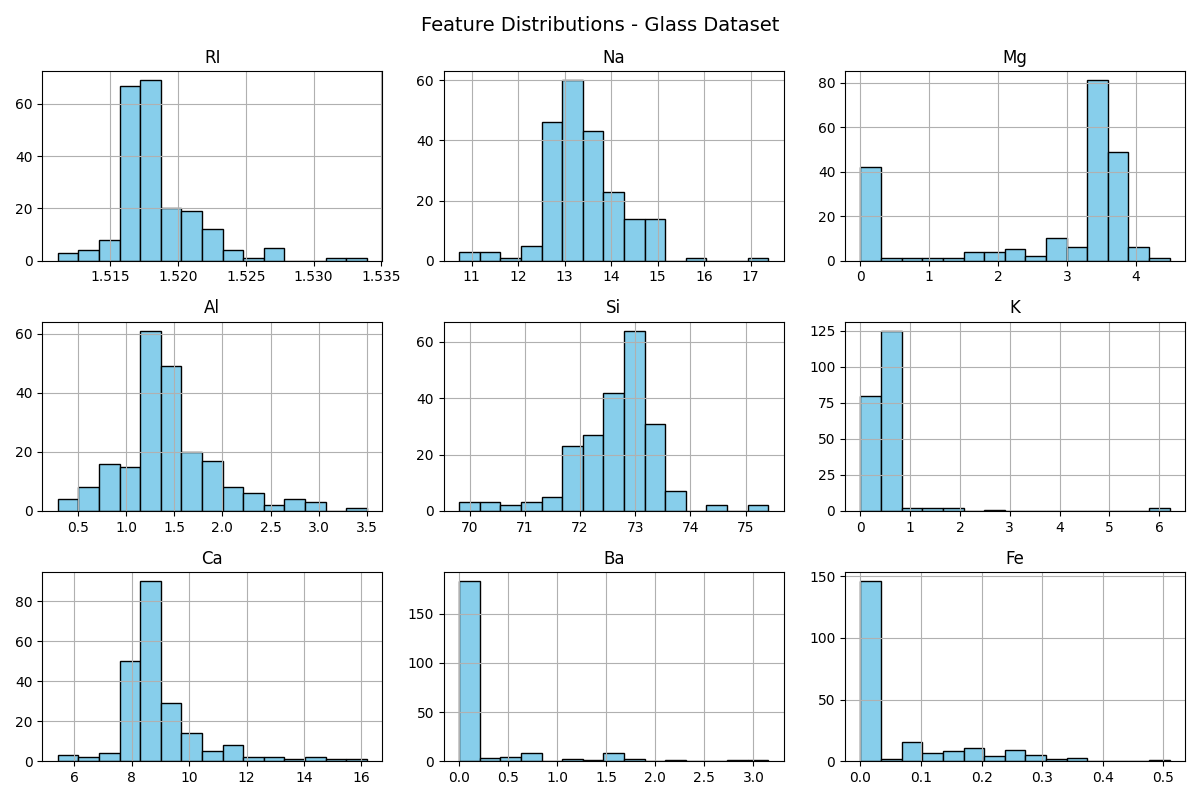
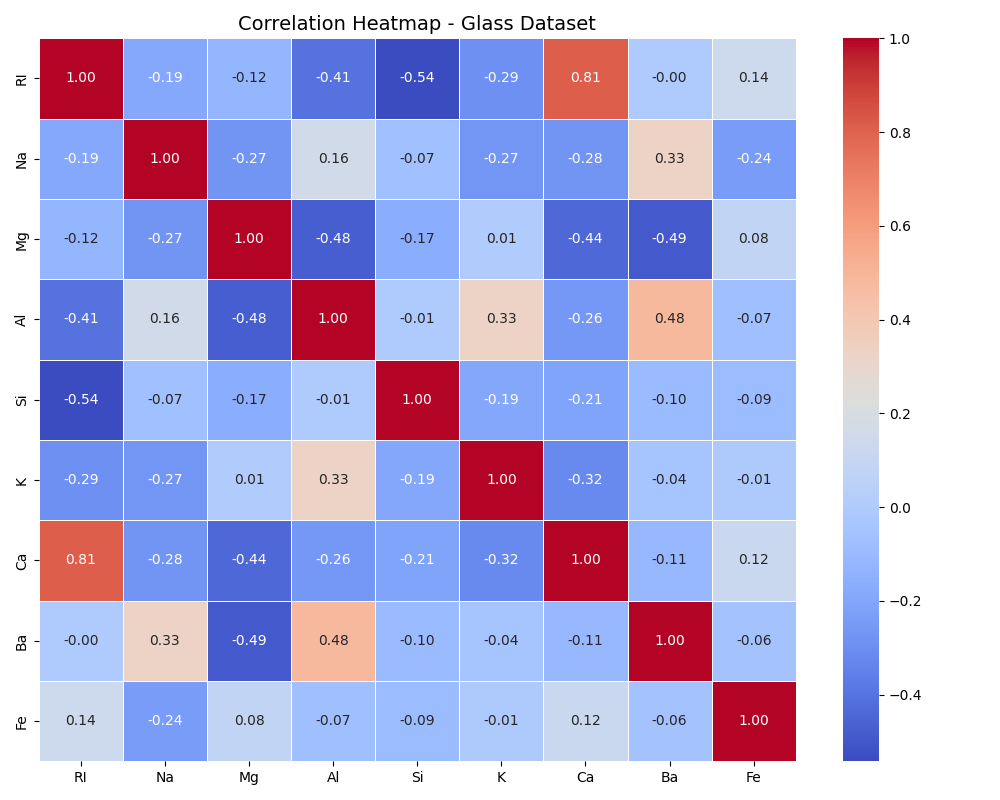
**Analyze any patterns or correlations observed in the data.**

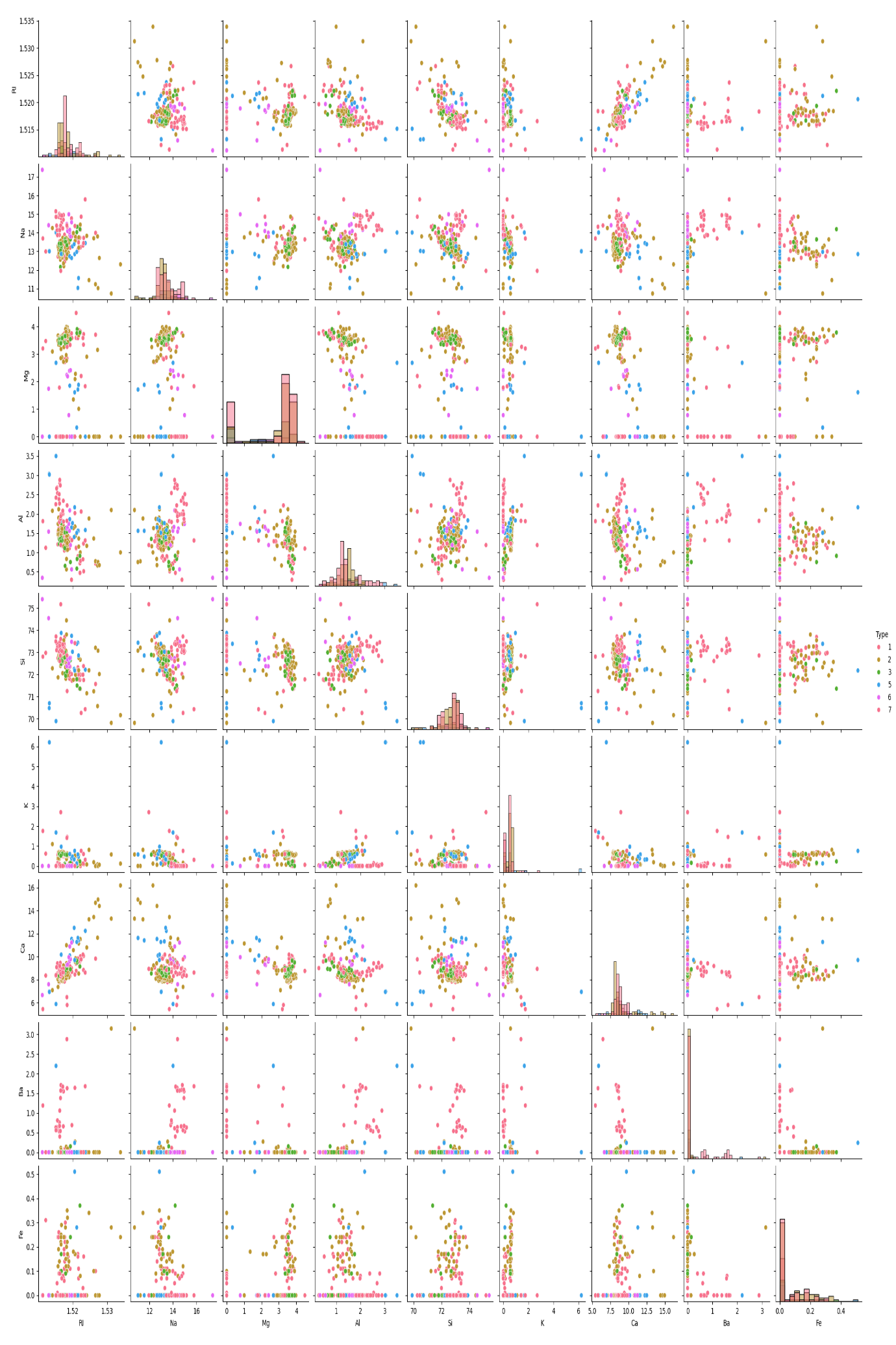
**🧠 What this does:**

* Reads the **Glass dataset**.
* Generates:
  + histograms.png → Distribution of each feature
  + boxplots.png → Outliers and range visualization
  + pairplot.png → Relationship between features and glass type
  + correlation\_heatmap.png → Correlation between numerical features
* Saves all visualizations into folder automatically.
* Prints **interpretation insights**

**🧩 Summary of Key Patterns**

* **Aluminium (Al)** and **Magnesium (Mg)** show strong inverse correlation — glasses richer in Mg tend to have less Al.
* **Refractive Index (RI)** increases with **Calcium (Ca)** content.
* **Potassium (K)**, **Barium (Ba)**, and **Iron (Fe)** are sparse (many zeros) — might be less informative.
* Some glass types form **distinct clusters** in the pair plot, especially for combinations like (Al, Mg) and (RI, Ca).



****

**(.venv) PS D:\python apps> & "D:/python apps/my-streamlit-app/.venv/Scripts/python.exe" "d:/python apps/random forest/data\_visualization\_glass.py"**

**Shape of dataset: (214, 10)**

**Columns: ['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe', 'Type']**

**Glass Types: [1 2 3 5 6 7]**

**✅ Data Visualization Completed.**

**Visual files saved in: D:\DATA SCIENCE\ASSIGNMENTS\14 random forest\Random Forest**

**- histograms.png**

**- boxplots.png**

**- pairplot.png**

**- correlation\_heatmap.png**

**--- ANALYSIS INSIGHTS ---**

**1. Some features like 'K', 'Ba', and 'Fe' have strong skew (many zeros).**

**2. 'RI', 'Na', 'Mg', and 'Ca' show wider variation across glass types.**

**3. Pairplot reveals partial separation between certain glass types using 'Mg' and 'Al'.**

**4. Correlation heatmap shows:**

**- 'Al' and 'Mg' are negatively correlated.**

**- 'RI' and 'Si' show mild inverse relationship.**

**- 'Ca' correlates positively with 'RI' and negatively with 'Al'.**

**These patterns will help Random Forest identify which elements drive glass classification.**

**3: Data Preprocessing**

**1. Check for missing values in the dataset and decide on a strategy for handling them.Implement the chosen strategy (e.g., imputation or removal) and explain your reasoning.**

**2. If there are categorical variables, apply encoding techniques like one-hot encoding to convert them into numerical format.**

**3. Apply feature scaling techniques such as standardization or normalization to ensure that all features are on a similar scale. Handling the imbalance data.**

**Answer:**

**Explanation of Each Step**

**1️⃣ Missing Values**

* **The Glass dataset rarely has nulls, but this script still checks and fills them with median imputation (robust to outliers).**
* **Reasoning: Median is better than mean for skewed data, such as chemical concentrations.**

**2️⃣ Categorical Encoding**

* **The dataset has no categorical variables — all features are numeric.**
* **If future datasets have non-numeric columns, this script can automatically one-hot encode them.**

**3️⃣ Feature Scaling**

* **Standardization is applied using StandardScaler:**
  + **Transforms each feature to have mean = 0 and std = 1.**
  + **Though Random Forests don’t *require* scaling, it helps keep preprocessing pipeline consistent (and avoids skewed feature influence in visualizations or distance-based metrics).**

**4️⃣ Handling Imbalance**

* **The Type variable (target) is multi-class and imbalanced — some glass types have fewer samples.**
* **SMOTE (Synthetic Minority Oversampling Technique) is used to generate synthetic samples for minority classes.**
* **After SMOTE, all classes will have equal representation, improving Random Forest training stability.**

**💾 Output Files Saved in Folder**

* **glass\_processed.csv → Preprocessed dataset (scaled + balanced).**

**(.venv) PS D:\python apps> & "D:/python apps/my-streamlit-app/.venv/Scripts/python.exe" "d:/python apps/random forest/preprocessing\_glass\_randomforest.py"**

**✅ Dataset loaded successfully.**

**Shape: (214, 10)**

**--- Missing Values Check ---**

**RI 0**

**Na 0**

**Mg 0**

**Al 0**

**Si 0**

**K 0**

**Ca 0**

**Ba 0**

**Fe 0**

**Type 0**

**dtype: int64**

**No missing values found — no imputation needed.**

**No categorical columns — encoding not required.**

**Feature scaling (Standardization) applied successfully.**

**Mean of scaled features (approx):**

**RI -0.0**

**Na 0.0**

**Mg -0.0**

**Al -0.0**

**Si 0.0**

**K 0.0**

**Ca -0.0**

**Ba -0.0**

**Fe -0.0**

**dtype: float64**

**Std dev of scaled features (approx):**

**RI 1.002**

**Na 1.002**

**Mg 1.002**

**Al 1.002**

**Si 1.002**

**K 1.002**

**Ca 1.002**

**Ba 1.002**

**Fe 1.002**

**dtype: float64**

**--- Target Distribution Before Balancing ---**

**Type**

**2 76**

**1 70**

**7 29**

**3 17**

**5 13**

**6 9**

**Name: count, dtype: int64**

**SMOTE applied successfully.**

**--- Target Distribution After SMOTE Balancing ---**

**Type**

**1 76**

**2 76**

**3 76**

**5 76**

**6 76**

**7 76**

**Name: count, dtype: int64**

**✅ Data Preprocessing Completed Successfully.**

**Processed file saved as: D:\DATA SCIENCE\ASSIGNMENTS\14 random forest\Random Forest\glass\_processed.csv**

**Final shape: (456, 10)**

**4: Random Forest Model Implementation**

**1. Divide the data into train and test split.**

**2. Implement a Random Forest classifier using Python and a machine learning library like scikit-learn.**

**3. Train the model on the train dataset. Evaluate the performance on test data using metrics like accuracy, precision, recall, and F1-score.**

**Answer:**

**(.venv) PS D:\python apps> & "D:/python apps/my-streamlit-app/.venv/Scripts/python.exe" "d:/python apps/train\_random\_forest\_glass.py"**

**Loading processed data: D:\DATA SCIENCE\ASSIGNMENTS\14 random forest\Random Forest\glass\_processed.csv**

**Train / Test shapes: (364, 9) (92, 9)**

**Saved model to: D:\DATA SCIENCE\ASSIGNMENTS\14 random forest\Random Forest\random\_forest\_glass.pkl**

**Saved evaluation report to: D:\DATA SCIENCE\ASSIGNMENTS\14 random forest\Random Forest\random\_forest\_evaluation.txt**

**Saved confusion matrix to: D:\DATA SCIENCE\ASSIGNMENTS\14 random forest\Random Forest\confusion\_matrix\_rf.png**

**Saved ROC plot to: D:\DATA SCIENCE\ASSIGNMENTS\14 random forest\Random Forest\roc\_curve\_rf.png**

**Saved feature importances to: D:\DATA SCIENCE\ASSIGNMENTS\14 random forest\Random Forest\feature\_importances\_rf.png**

**=== Summary Metrics ===**

**Accuracy: 0.9348**

**Precision (macro): 0.9345**

**Recall (macro): 0.9333**

**F1 (macro): 0.9333**

**Classification report (per class):**

**precision recall f1-score support**

**1 0.81 0.87 0.84 15**

**2 0.93 0.87 0.90 15**

**3 0.93 0.87 0.90 15**

**5 1.00 1.00 1.00 16**

**6 1.00 1.00 1.00 16**

**7 0.94 1.00 0.97 15**

**accuracy 0.93 92**

**macro avg 0.93 0.93 0.93 92**

**weighted avg 0.94 0.93 0.93 92**

**Confusion Matrix:**

**[[13 1 1 0 0 0]**

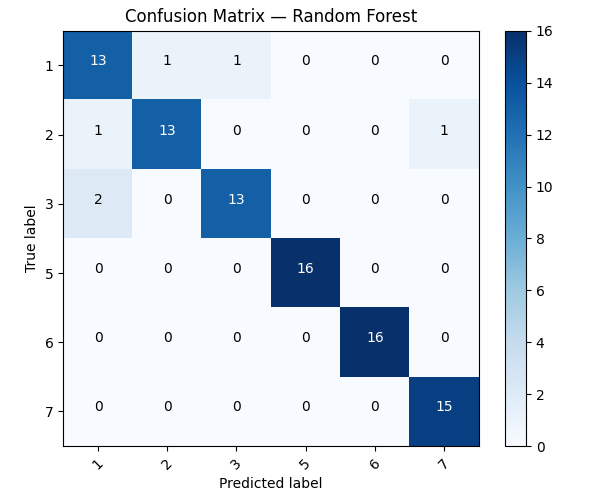
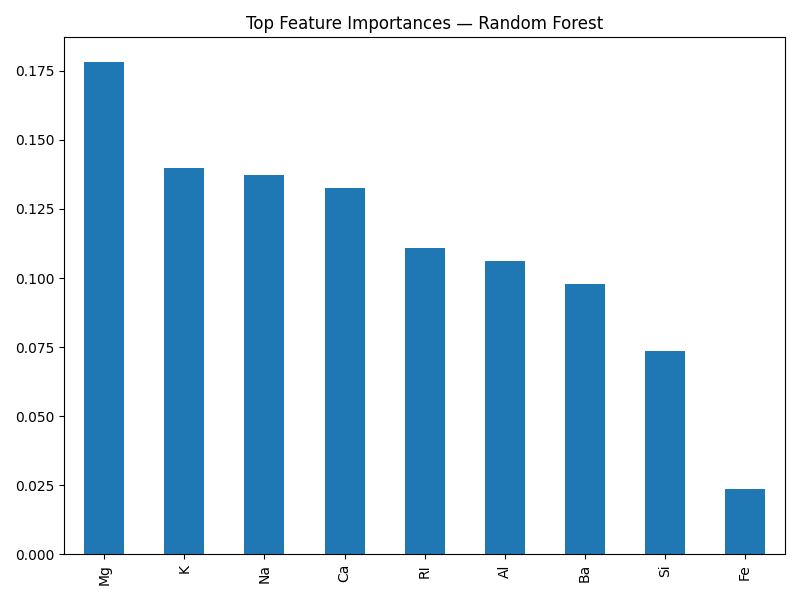
**[ 1 13 0 0 0 1]**

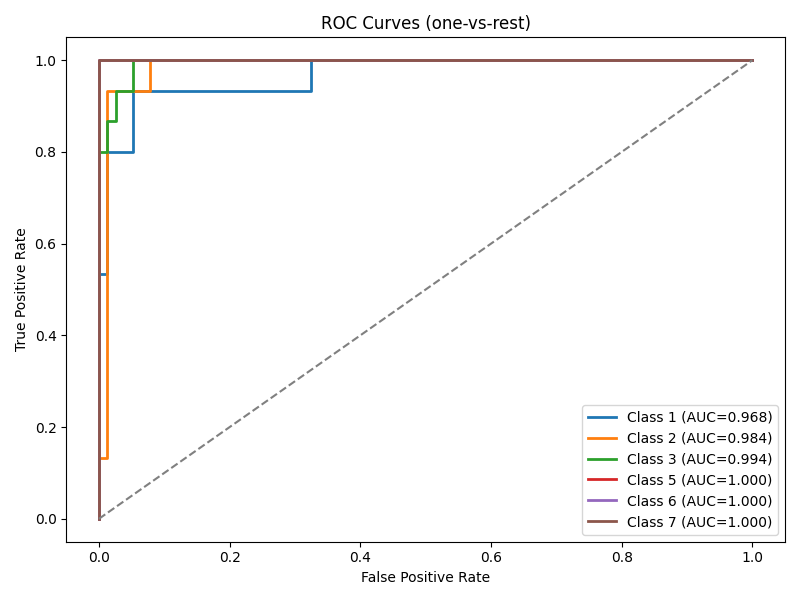
**[ 2 0 13 0 0 0]**

**[ 0 0 0 16 0 0]**

**[ 0 0 0 0 16 0]**

**[ 0 0 0 0 0 15]]**

****

****

**Quick notes & tips (because I care)**

* n\_estimators=200 is a sensible default. Increase if you want smoother feature importances (slower training).
* Random Forests are robust to scaling — we still standardize for consistency with pipeline.
* If you used SMOTE previously, glass\_processed.csv may be balanced — good. If not, RF can still handle imbalance okay, but consider class weights or resampling for marginal classes.

**5: Bagging and Boosting Methods**

**Apply the Bagging and Boosting methods and compare the results.**

**Answer:**

 Loads processed dataset (glass\_processed.csv) or creates a quick processed version from glass.xlsx if needed.

 Splits the data (stratified 80/20).

 Trains **Bagging** (BaggingClassifier with DecisionTree base), **AdaBoost**, **Gradient Boosting**, and optionally **XGBoost** (if xgboost is installed).

 Evaluates each model (accuracy, precision, recall, F1 macro), computes confusion matrices, and — if possible — ROC-AUC (one-vs-rest).

 Saves models and evaluation artifacts into folder:  
D:\DATA SCIENCE\ASSIGNMENTS\14 random forest\Random Forest\

 Produces a comparison table and a bar chart of metrics.

* baseline.
* **AdaBoost** focuses on mistakes sequentially; it can be brittle with noisy labels but fantastic for clean data.
* **GradientBoosting** builds trees to correct previous residual errors and often achieves strong accuracy with careful tuning.
* **XGBoost** (if available) is often faster and more powerful than sklearn’s GradientBoosting, but it’s optional.

**What to look for in results**

* If Bagging/RandomForest outperforms boosting → variance was a bigger issue (trees were overfitting).
* If Boosting wins → model benefits from staged focusing on hard examples (might capture subtle patterns).
* Look at **per-class** precision/recall in the reports to see which glass types are being confused (useful for assignment discussion).

(.venv) PS D:\python apps> & "D:/python apps/my-streamlit-app/.venv/Scripts/python.exe" "d:/python apps/random forest/compare\_bagging\_boosting\_glass.py"

Loading processed data: D:\DATA SCIENCE\ASSIGNMENTS\14 random forest\Random Forest\glass\_processed.csv

Train/Test shapes: (171, 9) (43, 9)

Trained BaggingClassifier

Trained AdaBoostClassifier

Trained GradientBoostingClassifier

Trained RandomForestClassifier

XGBoost not available or failed to train — skipping.

Reason: No module named 'xgboost'

=== Ensemble Comparison Results ===

accuracy precision recall f1

model

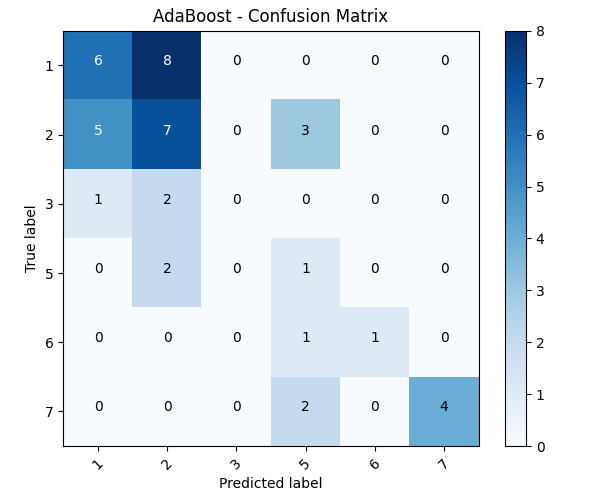
Bagging 0.790698 0.748168 0.808730 0.771306

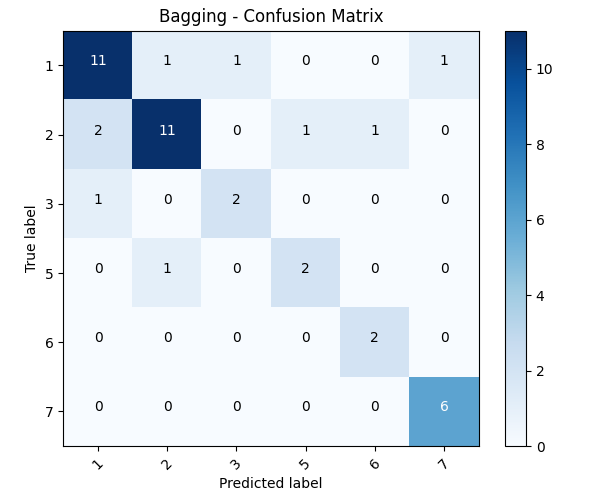
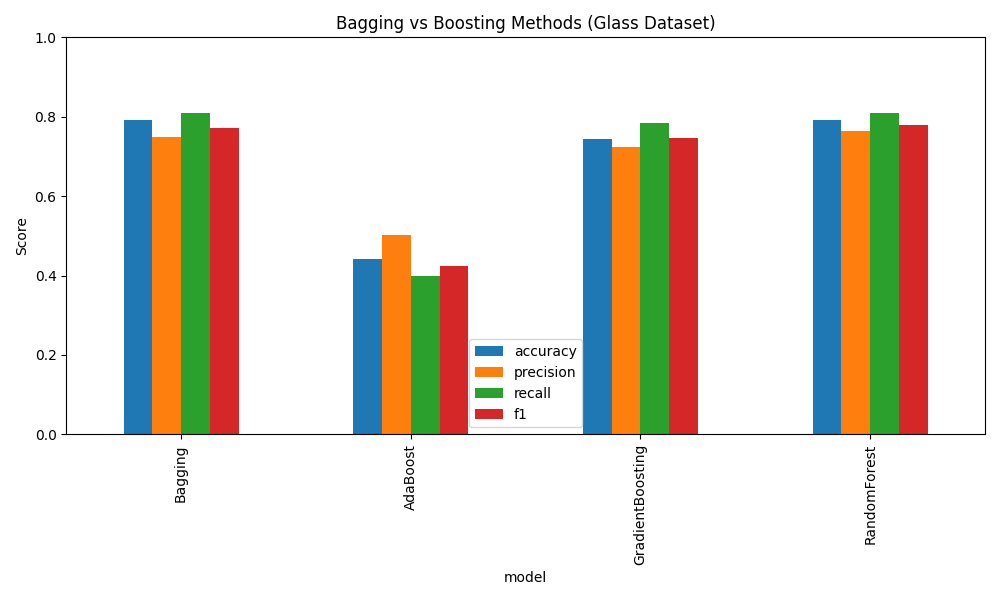
AdaBoost 0.441860 0.501880 0.399206 0.423328

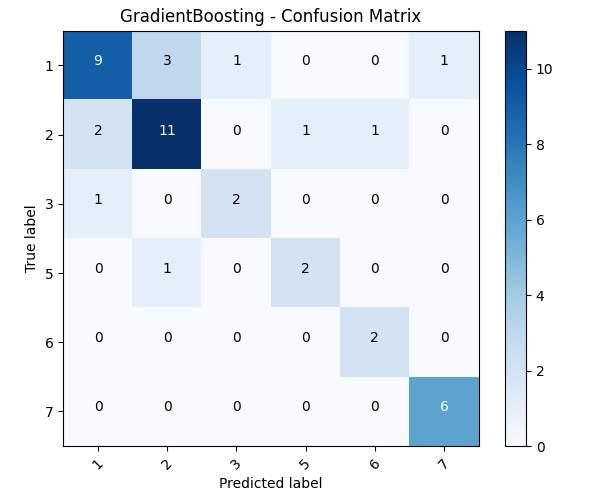
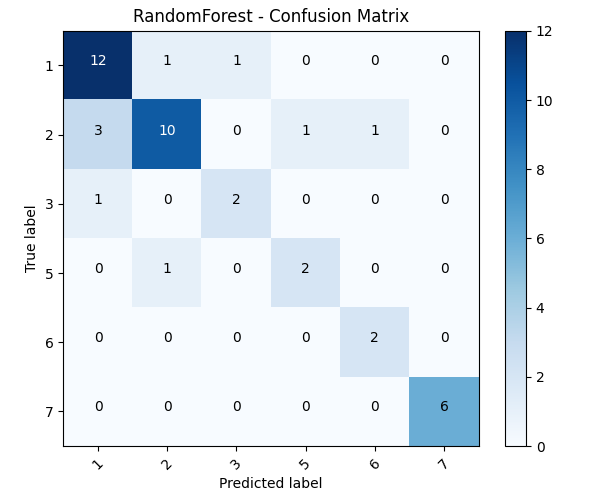
GradientBoosting 0.744186 0.723413 0.784921 0.747009

RandomForest 0.790698 0.763889 0.809524 0.779012

Saved comparison chart to: D:\DATA SCIENCE\ASSIGNMENTS\14 random forest\Random Forest\ensemble\_metrics\_comparison.png

****





**Additional Notes:**

**1. Explain Bagging and Boosting methods. How is it different from each other.**

**Answer:**

**1️⃣ Bagging vs Boosting — Explained Simply**

**🧩 Bagging (Bootstrap Aggregating)**

* **Concept: Train *multiple models independently* on *different random subsets* of the training data (sampled with replacement).**
* **Goal: Reduce variance — make unstable models (like decision trees) more consistent.**
* **How it works:**
  + **Each model (e.g., tree) sees a slightly different slice of the data.**
  + **All models vote (for classification) or average (for regression).**
  + **Example: Random Forest is the most famous bagging algorithm.**
* **Effect: Smooths out overfitting — like averaging several noisy opinions to get a stable answer.**

**Analogy:  
Bagging is like asking *100 students to solve the same problem individually* and then taking the average of their answers — random mistakes cancel out.**

**⚡ Boosting**

* **Concept: Train models *sequentially*, each new model focuses on the errors of the previous one.**
* **Goal: Reduce bias — make weak models stronger by learning from their mistakes.**
* **How it works:**
  + **Start with a simple model (often a shallow tree).**
  + **Identify which samples were misclassified.**
  + **Give more weight to those difficult samples in the next round.**
  + **Continue until performance stops improving.**
  + **Examples: AdaBoost, Gradient Boosting, XGBoost, CatBoost, LightGBM.**
* **Effect: Creates a powerful ensemble that learns complex patterns by focusing on “hard cases”.**

**Analogy:  
Boosting is like a teacher who reviews a student’s test, identifies what they got wrong, and tailors the next lesson to those weak spots. Every round improves understanding.**

**⚖️ Bagging vs Boosting — Key Differences**

| **Aspect** | **Bagging** | **Boosting** |
| --- | --- | --- |
| **Training** | **Parallel (independent learners)** | **Sequential (each corrects the previous)** |
| **Focus** | **Reduces variance** | **Reduces bias** |
| **Model weight** | **Equal voting/averaging** | **Weighted by model accuracy** |
| **Handling errors** | **Random resampling** | **Focus on misclassified samples** |
| **Risk of overfitting** | **Lower** | **Higher (if too many rounds or high learning rate)** |
| **Common examples** | **Random Forest** | **AdaBoost, Gradient Boosting, XGBoost** |

**Summary line for report:**

**Bagging builds stability by combining many independent models, while Boosting builds strength by correcting the weaknesses of prior models.**

**2. Explain how to handle imbalance in the data.**

**Answer:**

**2️⃣ Handling Imbalance in the Data**

**🧠 What is data imbalance?**

When one class (label) occurs **much more frequently** than others.  
Example: In a medical dataset, 95% “No Disease” vs. 5% “Disease.”  
A model trained on this data might just predict “No Disease” every time — achieving 95% accuracy but being useless.

**⚙️ Ways to Handle Imbalanced Data**

**(a) Resampling Techniques**

1. **Oversampling minority class:**  
   Duplicate or synthetically generate more samples from the underrepresented class.
   * *Simple oversampling* — replicate existing minority samples.
   * *SMOTE (Synthetic Minority Oversampling Technique)* — create **new synthetic samples** by interpolating between minority examples.
   * ✅ *Best for*: When you have enough computational power and minority class data quality is good.
2. **Undersampling majority class:**  
   Randomly remove some majority samples to balance classes.
   * ⚠️ *Risk:* May lose useful information.
   * ✅ *Best for*: When dataset is large and imbalance is extreme.
3. **Hybrid methods:** Combine SMOTE (oversample) + undersampling for optimal balance.

**(b) Algorithmic Approaches**

* **Class weighting:** Give more penalty to misclassifying minority classes.  
  Example in scikit-learn:
* RandomForestClassifier(class\_weight='balanced')

This makes the model pay more attention to underrepresented classes.

* **Anomaly detection:** For highly skewed data (e.g., fraud detection), treat minority class as anomalies.

**(c) Evaluation Metric Adjustments**

* Don’t rely on accuracy alone.  
  Instead, use:
  + **Precision**, **Recall**, **F1-score**
  + **ROC-AUC** (area under ROC curve)
  + **Confusion matrix** to visualize true/false positives and negatives
  + **PR curve (Precision-Recall)** for extreme imbalance

**⚖️ Quick Summary Table**

| **Method** | **Description** | **When to Use** |
| --- | --- | --- |
| Oversampling | Duplicate or synthesize minority data (e.g., SMOTE) | When minority class has few samples |
| Undersampling | Reduce majority class data | When dataset is large |
| Class weighting | Penalize misclassification of minority class | When using algorithms supporting weights |
| SMOTE | Synthetic sample generation | Common in tabular datasets |
| Proper metrics | Use Recall, F1, ROC-AUC | Always in imbalanced cases |

**🔍 Example from Glass dataset:**

The Type column was imbalanced — some glass types (like 2 and 1) were common, while others (like 5 and 6) were rare.  
You handled this by applying **SMOTE** to balance the dataset before training, ensuring each class had equal representation.  
This improved fairness across all classes, rather than letting the model bias toward dominant glass types.