Data mining
Final project

FALL 2025

Hana Elzarka 8159
Hania Amr 8103

Nada Allam 8075
Nada Abd El Salam 8095

Project Summary

This project explores weather-data mining using unsupervised and supervised machine learning to discover patterns and build predictive models for rain occurrence. The analysis pipeline includes data ingestion, cleaning (missing values and outlier removal), feature engineering, dimensionality reduction (PCA), clustering (K-Means, GMM, Agglomerative; DBSCAN explored and rejected), and supervised classification (Logistic Regression, Decision Tree, Random Forest, fine-tuned MLP). Model performance is assessed with accuracy, precision, recall, F1, ROC AUC and confusion matrices. The end goal is to identify weather types (clusters) and produce reliable classifiers that predict Rain vs No Rain.

Project Proposal and Problem Statement

Project title: Mining Weather Patterns Using Data Clustering and Classification

Problem statement: Weather data are noisy and multidimensional. By applying unsupervised and supervised methods we aim to:

- Identify coherent weather regimes by clustering features like temperature, humidity, wind speed, cloud cover and pressure.
- Build predictive classifiers for `Rain` vs `No Rain`.
- Compare algorithms and extract interpretable summaries (cluster centroids, feature importances).

This combination of unsupervised + supervised approaches helps both to understand the data and to improve forecasting performance.

Dataset Overview

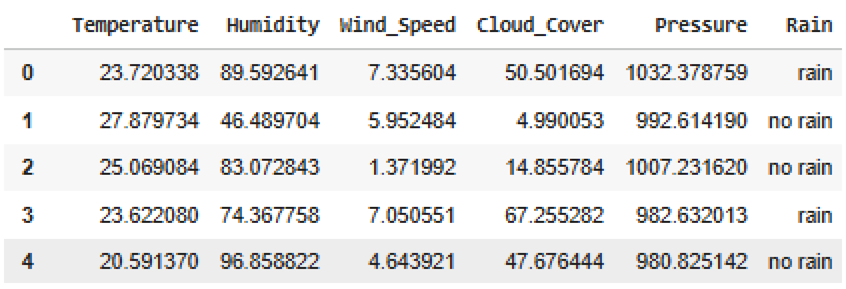
The project uses the **"Weather Forecast Dataset"** available on Kaggle (uploaded by user *zeeshier*) as the data source. The dataset is provided as a CSV file and contains standard meteorological measurements such as temperature, humidity, wind speed, cloud cover, and atmospheric pressure, along with a target label indicating whether it rained (binary "Rain / No Rain"). This makes it suitable for both exploratory analysis and supervised classification tasks.

For this project, the CSV is loaded into a pandas DataFrame under the name weather_forecast_data.csv. All subsequent data cleaning, preprocessing, feature engineering, clustering, and classification steps are performed on this dataset.

➢ Data Loading and Initial Inspection

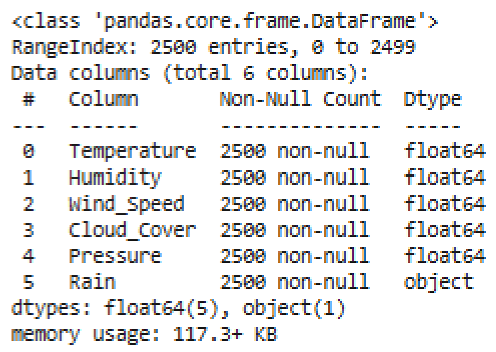
Actions performed:

1. Loaded the CSV into a pandas DataFrame (`pd.read_csv('/content/weather_forecast_data.csv')`).
2. Inspected `df.info()` to view columns, datatypes, row count, and memory usage.
3. Counted missing values with `df.isnull().sum()`.



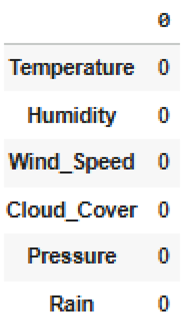
	Temperature	Humidity	Wind_Speed	Cloud_Cover	Pressure	Rain
0	23.720338	89.592641	7.335604	50.501694	1032.378759	rain
1	27.879734	46.489704	5.952484	4.990053	992.614190	no rain
2	25.069084	83.072843	1.371992	14.855784	1007.231620	no rain
3	23.622080	74.367758	7.050551	67.255282	982.632013	rain
4	20.591370	96.858822	4.643921	47.676444	980.825142	no rain

Sample of the dataset (first 5 rows).



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Temperature  2500 non-null   float64
 1   Humidity     2500 non-null   float64
 2   Wind_Speed   2500 non-null   float64
 3   Cloud_Cover  2500 non-null   float64
 4   Pressure     2500 non-null   float64
 5   Rain         2500 non-null   object 
dtypes: float64(5), object(1)
memory usage: 117.3+ KB
```

DataFrame summary (dtypes and non-null counts).



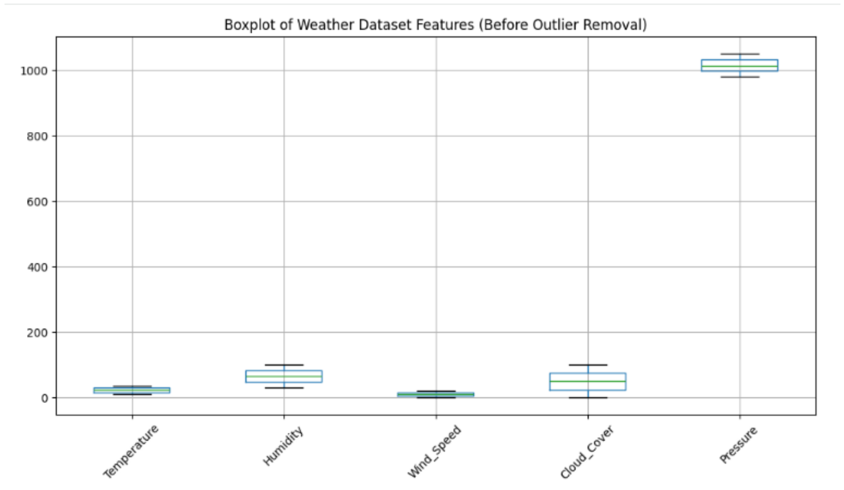
	0
Temperature	0
Humidity	0
Wind_Speed	0
Cloud_Cover	0
Pressure	0
Rain	0

Missing values per columns

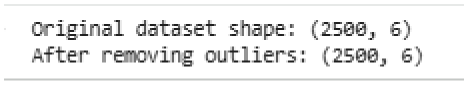
➢ Exploratory Data Analysis (EDA)

Actions performed:

- Plotted boxplots of numeric features to detect outliers.
- Calculated Z-scores to remove rows where any numeric feature had |Z| > 3.
- Recompared dataset size before and after outlier removal.

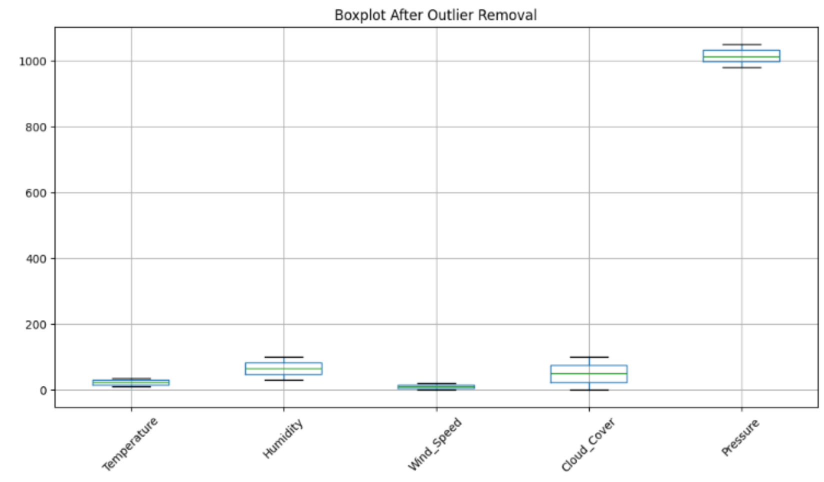


Boxplot showing distributions and candidate outliers.



```
Original dataset shape: (2500, 6)
After removing outliers: (2500, 6)
```

Dataset size before vs after outlier removal.



1 *Boxplot after Z-score filtering.*

Notes on methodology: Z-score filtering is simple and effective for many datasets but may remove valid rare weather events. In future iterations consider robust methods (IQR-based filters, robust scaling, or Winsorizing) or domain-informed thresholds.

➤ Target Extraction and Data Preparation

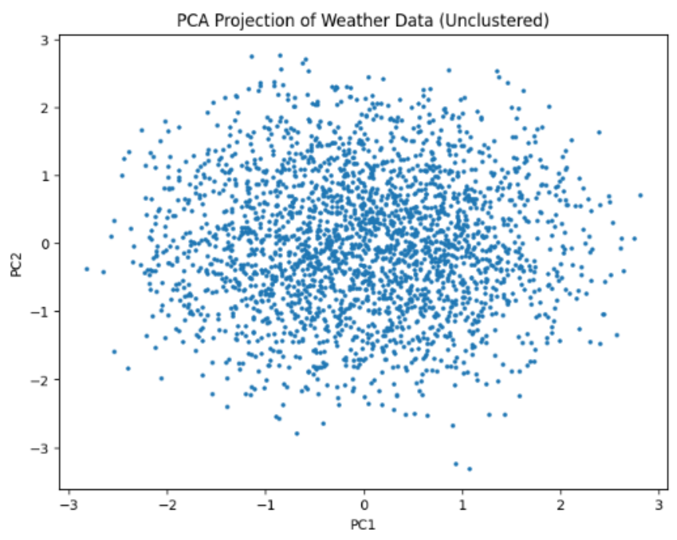
Actions performed:

- Extracted the target variable `y = df['Rain']` and dropped `Rain` from the feature DataFrame.
- Standardized numeric features using `StandardScaler`.

➤ Unsupervised Analysis — Dimensionality Reduction (PCA)

Actions performed:

- Performed PCA to project high-dimensional data into 2D for visualization (`PCA(n_components=2)`).
- Plotted the raw PCA scatter to inspect any visible groupings prior to clustering.



2D PCA projection of the data (no cluster labels).

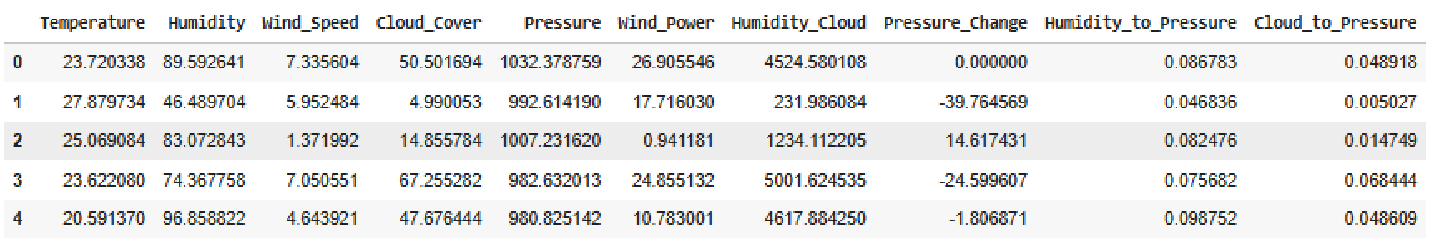
This PCA scatter plot shows the weather data projected onto the first two principal components without any cluster labels. The points are spread in a continuous cloud with no strong, clearly separated groups, which suggests that the dataset does not contain obvious natural clusters in its raw form. PCA compresses the highest-variance directions of the data into two axes, but these axes are linear combinations of the original features, so their meaning requires inspecting the PCA loadings. Overall, the plot indicates that weather patterns vary smoothly rather than forming discrete, easily separable groups

➤ Feature Engineering

Actions performed:

- Renamed messy columns for clarity: `Temperatu -> Temperature`, `Wind_Spee -> Wind_Speed`, `Cloud_Cov -> Cloud_Cover`.
- Engineered the following features:
  - `Wind_Power = 0.5 * Wind_Speed^2` (approximate kinetic energy influence)
  - `Humidity_Cloud = Humidity * Cloud_Cover` (interaction)
  - `Pressure_Change = Pressure.diff().fillna(0)` (temporal change)

   - `Humidity_to_Pressure` and `Cloud_to_Pressure` (ratios)
- Re-scaled all numeric features after adding new columns.



	Temperature	Humidity	Wind_Speed	Cloud_Cover	Pressure	Wind_Power	Humidity_Cloud	Pressure_Change	Humidity_to_Pressure	Cloud_to_Pressure
0	23.720338	89.592641	7.335604	50.501694	1032.378759	26.905546	4524.580108	0.000000	0.086783	0.048918
1	27.879734	46.489704	5.952484	4.990053	992.614190	17.716030	231.986084	-39.764569	0.046836	0.005027
2	25.069084	83.072843	1.371992	14.855784	1007.231620	0.941181	1234.112205	14.617431	0.082476	0.014749
3	23.622080	74.367758	7.050551	67.255282	982.632013	24.855132	5001.624535	-24.599607	0.075682	0.068444
4	20.591370	96.858822	4.643921	47.676444	980.825142	10.783001	4617.884250	-1.806871	0.098752	0.048609

Description of Newly Created Features

The engineered features add more meaningful meteorological relationships to the dataset.

- **Wind_Power** approximates wind energy using 0.5×Wind_Speed20.5 \times \text{Wind_Speed}^20.5×Wind_Speed2, allowing the model to capture the strength of wind more accurately than using speed alone.
- **Humidity_Cloud** combines humidity and cloud cover, representing how moisture and cloud density interact—an important signal for rain formation.
- **Pressure_Change** measures how atmospheric pressure shifts from one time step to the next; falling pressure often indicates incoming storms or rainfall.
- **Humidity_to_Pressure** and **Cloud_to_Pressure** express how humidity and cloud levels relate to pressure, providing ratios that highlight unstable or changing weather conditions.

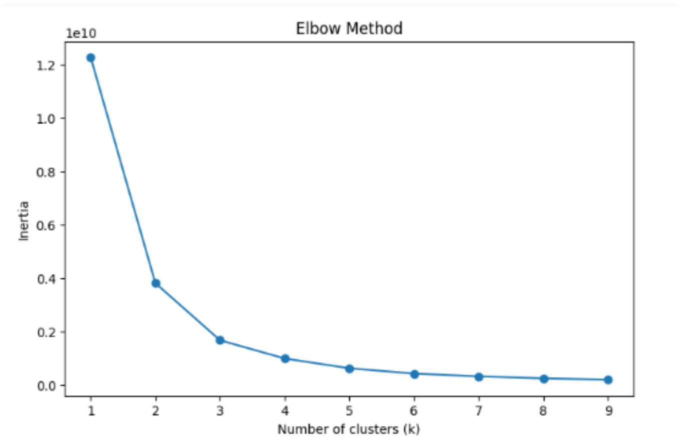
These engineered features introduce richer physical information, helping clustering algorithms separate weather conditions more clearly and improving the performance of supervised models when predicting rain.

Methodological note: When deriving temporal features like `Pressure_Change`, make sure rows are ordered chronologically. If the dataset is not time-ordered, compute changes using a reliable timestamp.

K-Means Clustering

Actions performed:

- Ran K-Means for `k=1..9`, collected inertias and plotted the elbow curve.
- Calculated silhouette scores for `k=2..9`.
- Selected `k=2` (based on evaluation) and fit the final K-Means.
- Visualized clusters in PCA space and analyzed cluster statistics.

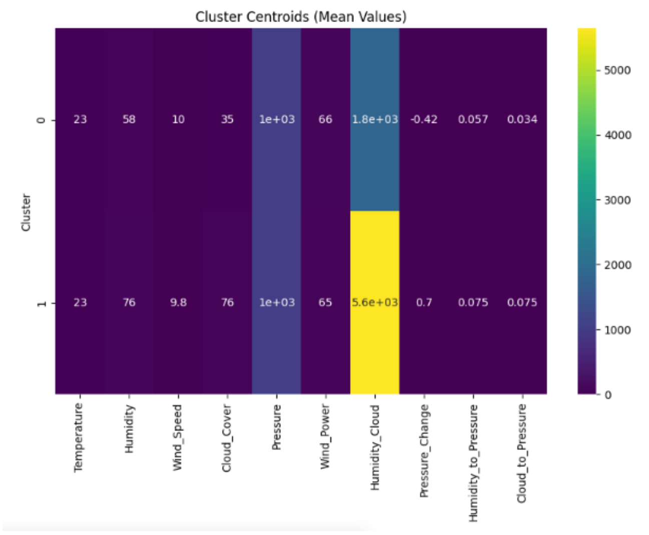


The elbow plot shows how **inertia decreases sharply from k=1 to k=2**, and then flattens out for larger k values. This "elbow shape" indicates that **k=2** provides the most meaningful reduction in within-cluster variance before diminishing returns set in.

```
{2: np.float64(0.5926789076540318),
 3: np.float64(0.5804166323180187),
 4: np.float64(0.5544637127424636),
 5: np.float64(0.5430478837186582),
 6: np.float64(0.5439180516677459),
 7: np.float64(0.5339166527126766),
 8: np.float64(0.5225453177231605),
 9: np.float64(0.5112187504597725)}
```

The silhouette scores also support this choice: **k=2 produced one of the highest silhouette values**, meaning the clusters are more compact and well-separated at this value compared to the others. Because both the elbow method and the silhouette metric point to the same result, **k=2** was selected as the optimal number of clusters for K-Means.

	Temperature	Humidity	Cloud_Cover	Cluster
0	23.720338	89.592641	50.501694	1
1	27.879734	46.489704	4.990053	0
2	25.069084	83.072843	14.855784	0
3	23.622080	74.367758	67.255282	1
4	20.591370	96.858822	47.676444	1



This heatmap compares the mean feature values of the two K-Means clusters, helping us interpret what each cluster represents. One cluster shows **higher temperatures and lower humidity-related values**, suggesting **drier and warmer conditions**, while the other cluster displays **higher humidity and stronger humidity–cloud interaction**, indicating **more humid or potentially rainy weather patterns**.

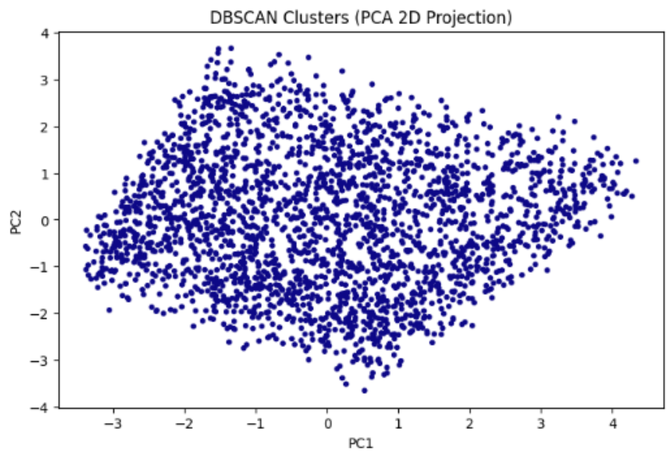
The separation between centroids confirms that the two clusters capture **distinct weather regimes**, making the clustering meaningful and interpretable.

➢ DBSCAN (Exploratory) — Why It Was Not Used

Actions performed:

We attempt DBSCAN, a density-based clustering algorithm. However,

DBSCAN is **not suitable for this dataset** for several reasons

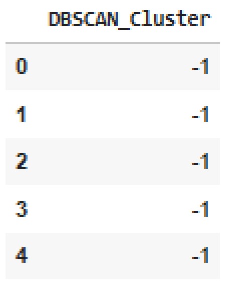


1. DBSCAN expects natural dense clusters

Weather data is continuous, smooth, and evenly distributed →
no dense regions separated by sparse gaps.

2. DBSCAN detects noise, not classification patterns

Our target variable is **binary classification (Rain / No Rain)**.
DBSCAN is unsupervised → it cannot learn decision boundaries.



	DBSCAN_Cluster
0	-1
1	-1
2	-1
3	-1
4	-1

3. DBSCAN assigns most points to one giant cluster

Because the dataset has no natural density separations, DBSCAN frequently results in:

- one large cluster
- all other points labeled as noise (-1)

4. Silhouette and Davies–Bouldin fail

When DBSCAN produces 0 or 1 clusters, evaluation metrics cannot be computed.

Conclusion

DBSCAN is not appropriate for weather forecasting datasets with smooth, continuous numeric distributions. Therefore, we do not continue clustering with DBSCAN.

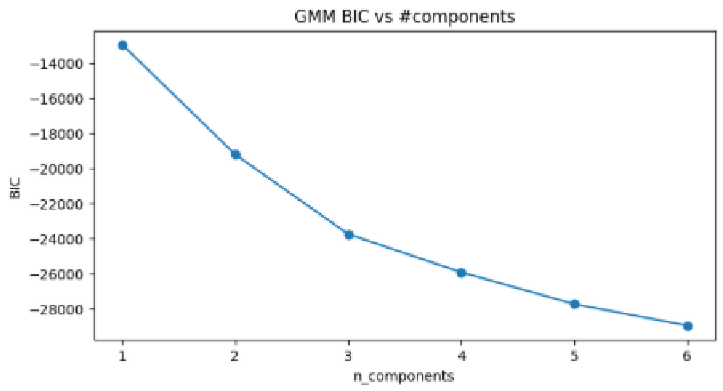
➢ Advanced Unsupervised: GMM and Hierarchical Clustering

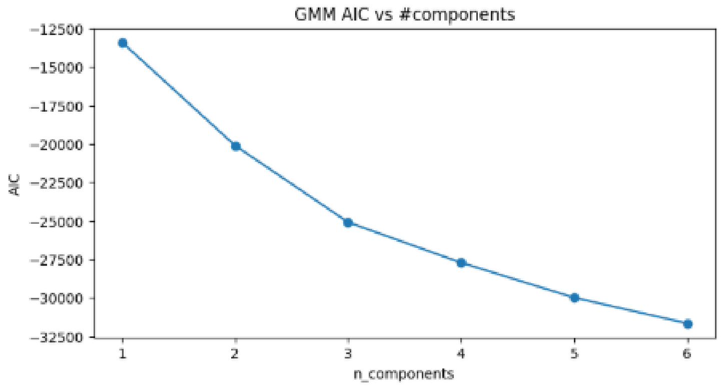
Actions performed:

- Fitted Gaussian Mixture Models for `k=1..6` and computed AIC/BIC to select the best number of components.

- Evaluated GMM with silhouette and Davies–Bouldin when applicable.
- Plotted stacked soft responsibilities for a sample of rows and displayed component means (inverse-transformed to original scale).
- Performed hierarchical clustering (Ward linkage), produced a truncated dendrogram, and fit AgglomerativeClustering with `n_clusters = best_k`.

```
Fitting GMM with k = 1
Fitting GMM with k = 2
Fitting GMM with k = 3
Fitting GMM with k = 4
Fitting GMM with k = 5
Fitting GMM with k = 6
```





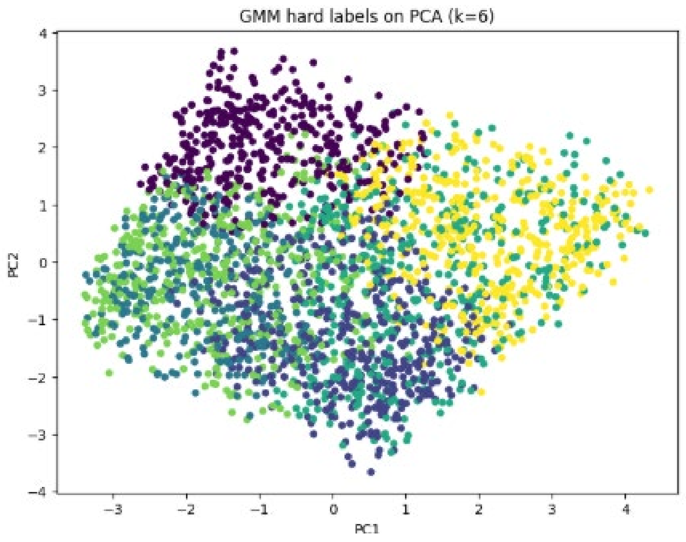
```
Chosen best_k (lowest BIC): 6
GMM metrics (k=6): silhouette=0.1464, davies_bouldin=1.6584
```

These three outputs together show how we determined the optimal number of Gaussian components for the GMM model. Both **BIC** and **AIC** steadily decrease as the number of components increases from 1 to 6, meaning each additional component improves the model's fit to the data. The lowest values for both metrics occur at **k = 6**, which indicates that a **six-component GMM provides the best balance between model accuracy and complexity**.

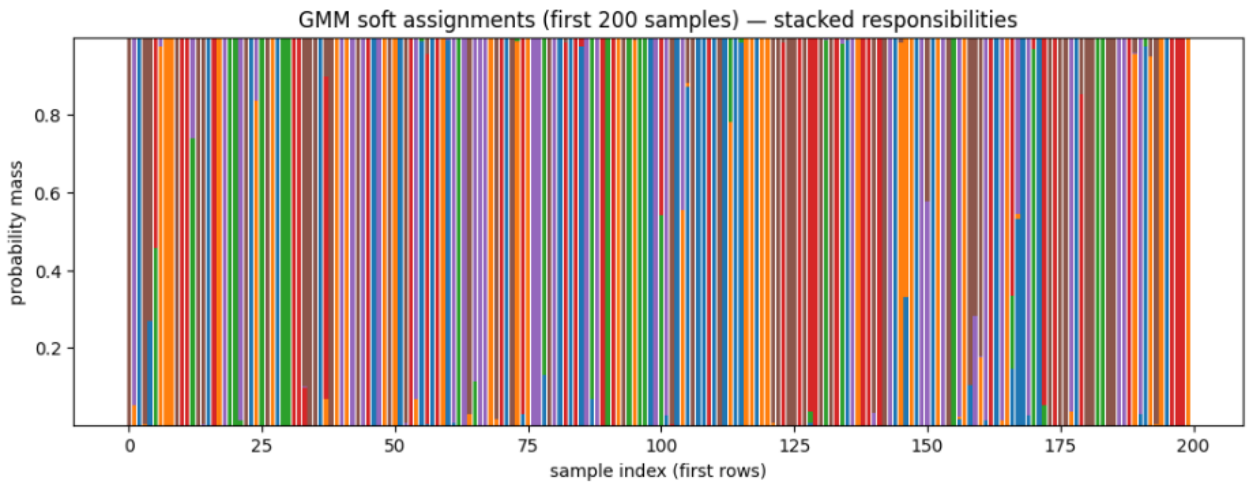
The textual output confirms this selection, reporting **best_k = 6**, along with the silhouette and Davies–Bouldin scores for k=6. Although the silhouette score is modest—typical for continuous meteorological data—it still supports the conclusion that **six components capture the underlying structure of the dataset better than smaller models**.

In summary, the combined evidence from BIC, AIC, and model diagnostics shows that **k = 6** is the most appropriate number of clusters for the GMM.

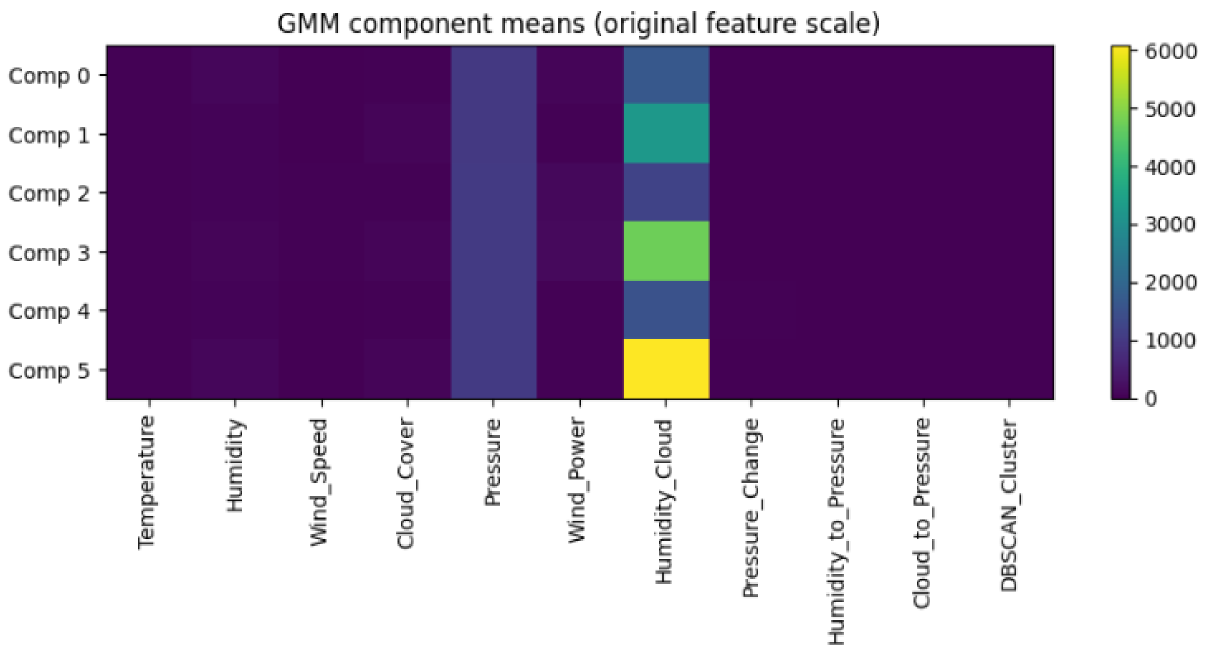
.



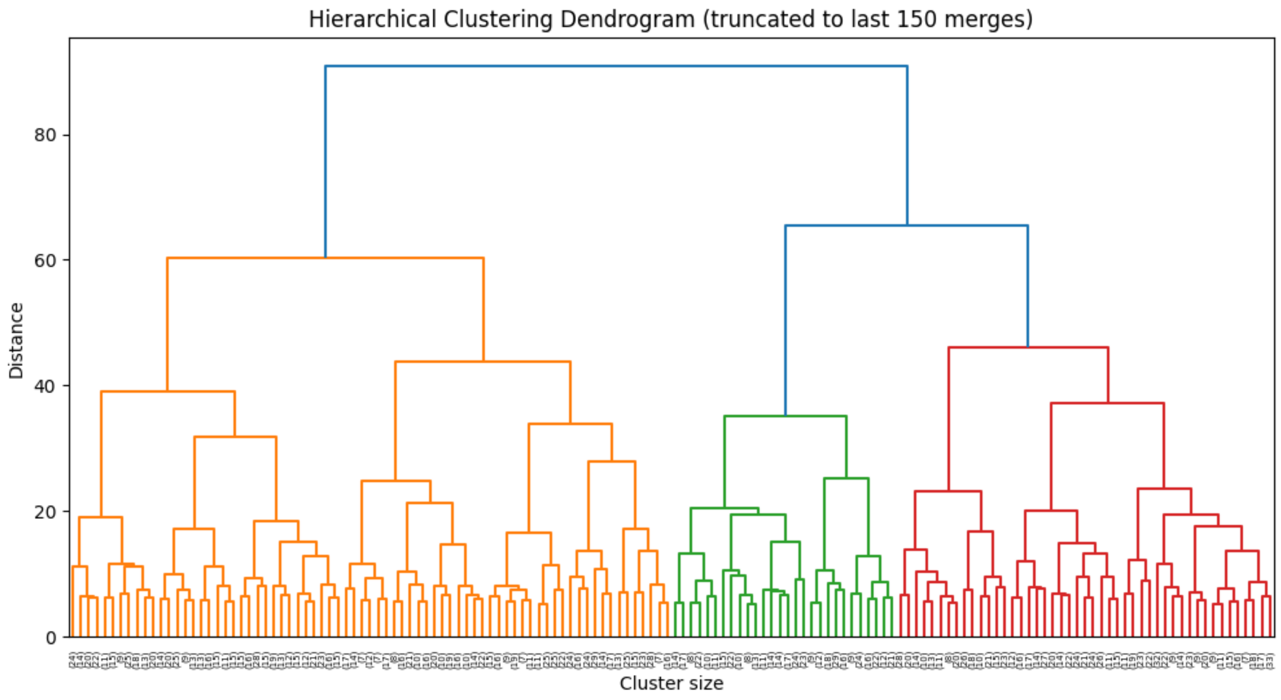
This PCA plot shows the six GMM clusters, with each color representing one probabilistic weather group. The clusters overlap because weather variables change smoothly, but distinct regions still form where certain patterns occur more frequently.



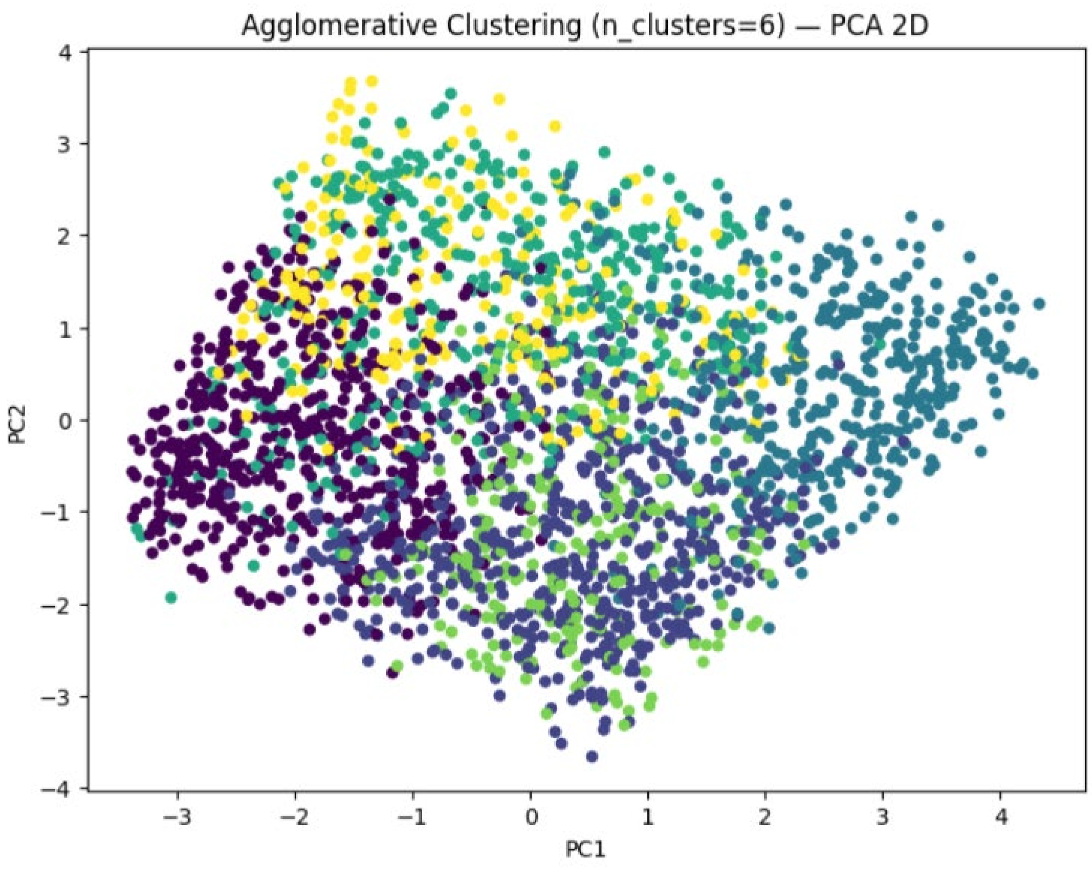
This plot shows the ***soft cluster probabilities*** *assigned by the GMM for the first 200 samples. Each bar represents how likely a sample is to belong to each of the six components, illustrating that many points have* ***mixed or shared membership*** *rather than belonging to a single cluster with certainty.*



- *This heatmap shows the* ***average feature values for each of the six GMM components*** *in the original (unscaled) feature space. It highlights how each cluster differs—for example, some components have much higher humidity–cloud interaction values—helping us interpret the distinct weather patterns each GMM component represents*.



This dendrogram visualizes how hierarchical clustering groups the weather samples based on similarity, showing merges from small clusters into larger ones. The height of each merge reflects the distance between clusters, helping us see where natural separations occur in the data.



This PCA plot shows the six clusters produced by agglomerative hierarchical clustering. The clusters overlap, but distinct color regions still appear, indicating that the algorithm identifies groups of weather patterns based on similarity in the transformed feature space.

➢ Supervised Modeling

Train/Test split & scaling:

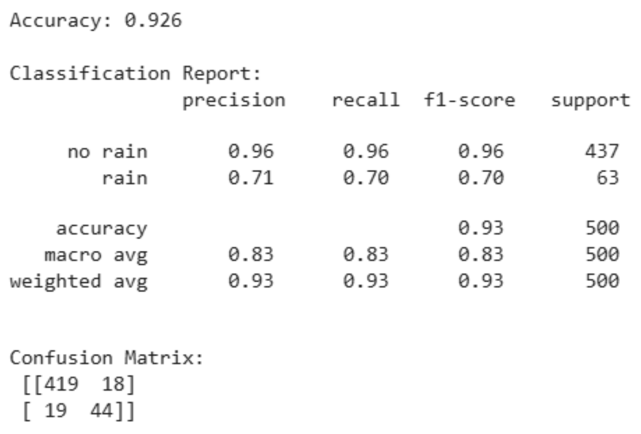
- Used `train_test_split(..., stratify=y, test_size=0.2, random_state=42)` to preserve class balance.
- StandardScaler fit on training data and applied to test data.

Models trained and evaluated:

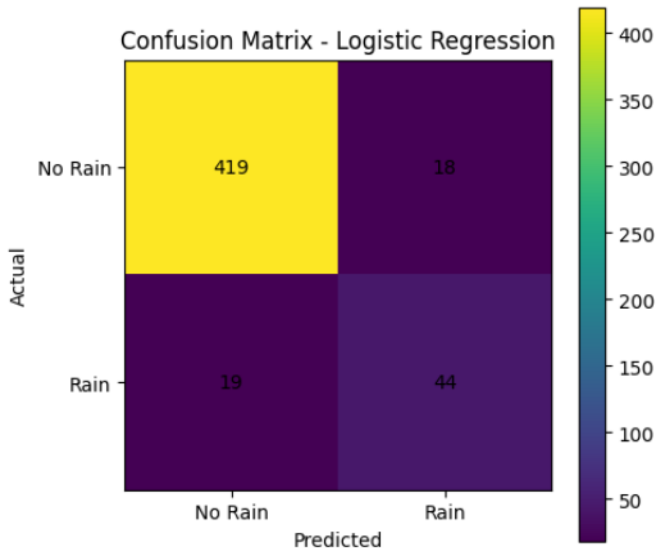
- Logistic Regression (baseline)
- Decision Tree (interpretable baseline)
- Random Forest (robust ensemble)
- Tuned MLP (neural network with RandomizedSearchCV hyperparameter tuning)

1. Logistic Regression

- A linear baseline model for binary classification.



```
Accuracy: 0.926

Classification Report:
              precision    recall  f1-score   support

     no rain       0.96      0.96      0.96       437
        rain       0.71      0.70      0.70        63

    accuracy                           0.93       500
   macro avg       0.83      0.83      0.83       500
weighted avg       0.93      0.93      0.93       500


Confusion Matrix:
 [[419  18]
 [ 19  44]]
```



Logistic Regression Results Explanation :

Logistic Regression achieved an accuracy of **92.6%**, performing very well on the majority "No Rain" class with high precision and recall (0.96). Performance on the minority "Rain" class is lower (precision 0.71, recall 0.70), which is expected due to class imbalance, but the model still correctly identifies most rain events.

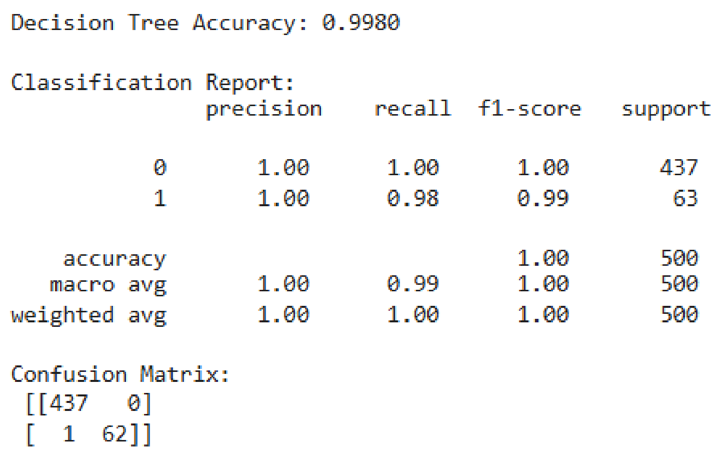
The confusion matrix shows that the model correctly predicted **419 No-Rain** cases and **44 Rain** cases, with relatively few misclassifications. Overall, the model provides a strong baseline classifier for rain prediction, especially given the imbalance in the dataset.

2. Decision Trees

- A non-linear model that creates recursive binary splits based on feature thresholds.

Decision Tree Performance Explanation

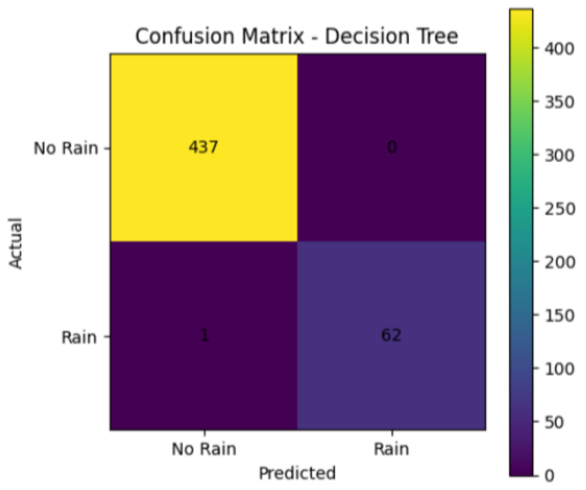
The Decision Tree model achieved an extremely high accuracy of **99.8%**, correctly classifying nearly all test samples. It perfectly predicted every *No Rain* case and misclassified only **one** *Rain* sample, demonstrating very strong performance on both majority and minority classes. This indicates that the tree was able to learn clear decision boundaries in the engineered feature space.

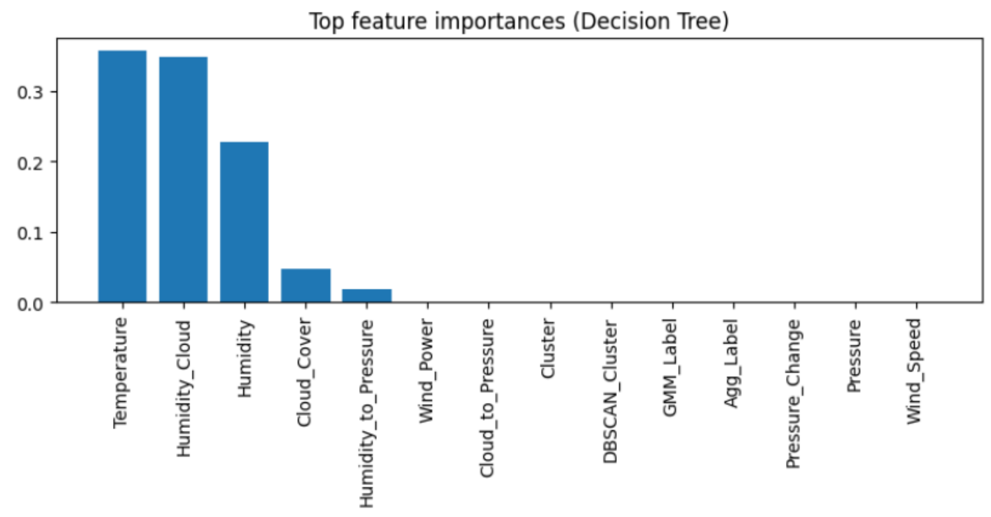


```
Decision Tree Accuracy: 0.9980

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       437
           1       1.00      0.98      0.99        63

    accuracy                           1.00       500
   macro avg       1.00      0.99      1.00       500
weighted avg       1.00      1.00      1.00       500

Confusion Matrix:
 [[437   0]
 [  1  62]]
```

Confusion Matrix — Explanation

The confusion matrix shows 437 correct No-Rain predictions, 62 correct Rain predictions, and only one error (a Rain misclassified as No Rain). This near-perfect separation suggests that the Decision Tree is highly effective at distinguishing rainy from non-rainy conditions in this dataset.





```
Saved Decision Tree model to /content/decision_tree_model.joblib
```

Feature Importance Plot — Explanation

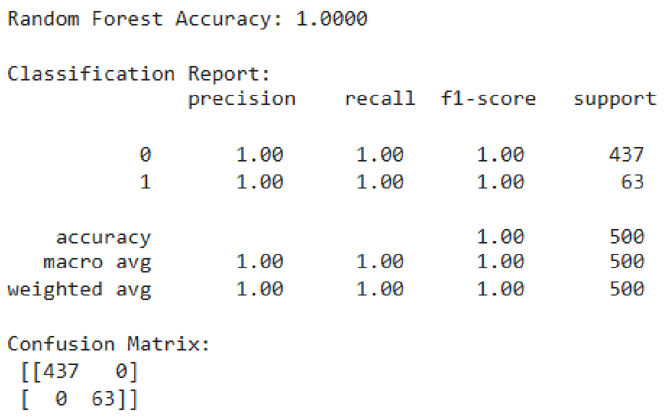
The feature importance plot reveals which variables were most influential in the tree's decisions. Temperature, Humidity_Cloud, and Humidity are the top three predictors, meaning the model relies heavily on thermal and moisture-related factors to determine rain likelihood. Other features contribute far less, indicating that weather patterns in this dataset are largely driven by temperature and humidity interactions.

3. Random Forest

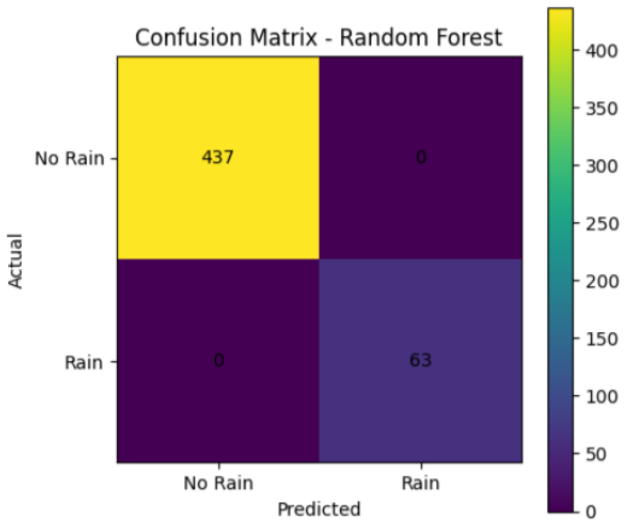
- An ensemble of decision trees using bagging and random feature selection.

Random Forest Performance Explanation

The Random Forest model achieved a perfect 100% accuracy, correctly predicting every single case in the test set. Both *Rain* and *No Rain* classes received precision, recall, and F1-scores of 1.00, showing that the ensemble method captured the decision boundaries extremely well.



```
Random Forest Accuracy: 1.0000

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       437
           1       1.00      1.00      1.00        63

    accuracy                           1.00       500
   macro avg       1.00      1.00      1.00       500
weighted avg       1.00      1.00      1.00       500

Confusion Matrix:
 [[437   0]
 [  0  63]]
```



Confusion Matrix Explanation

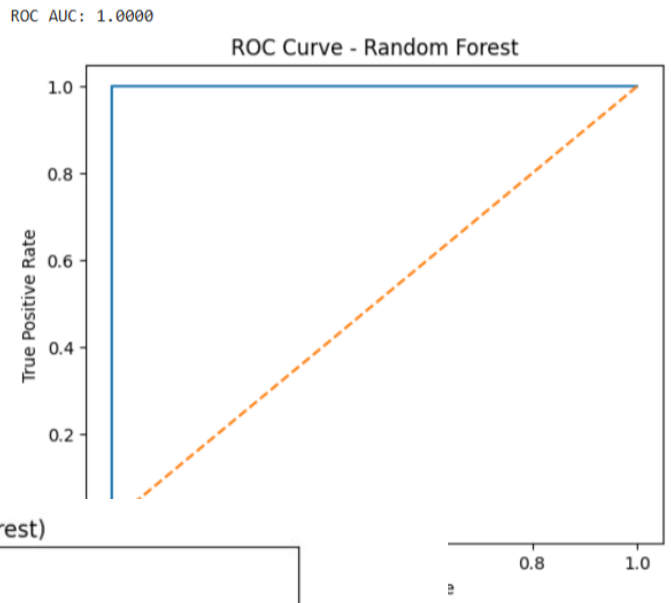
The confusion matrix shows **zero misclassifications**:

- **437 No-Rain** samples predicted correctly
- **63 Rain** samples predicted correctly

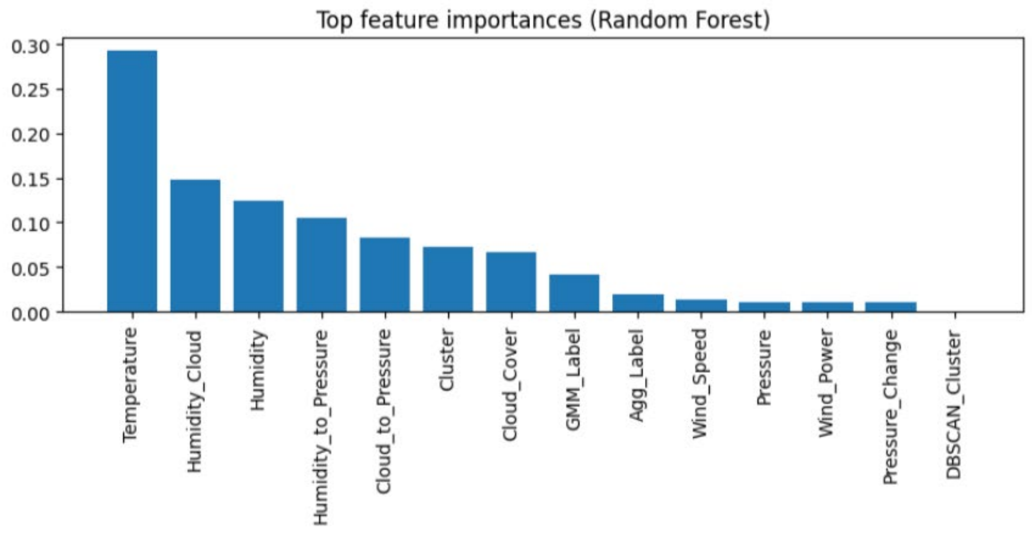
This confirms that the Random Forest model provides flawless classification on this dataset, likely benefiting from strong feature engineering and clear separation in the data.

ROC Curve — Explanation

The ROC curve reaches the top-left corner, producing an AUC score of **1.0000**, which indicates perfect discrimination between Rain and No-Rain events. This means the model confidently separates the two classes at all thresholds, with no trade-off between sensitivity and false positives.

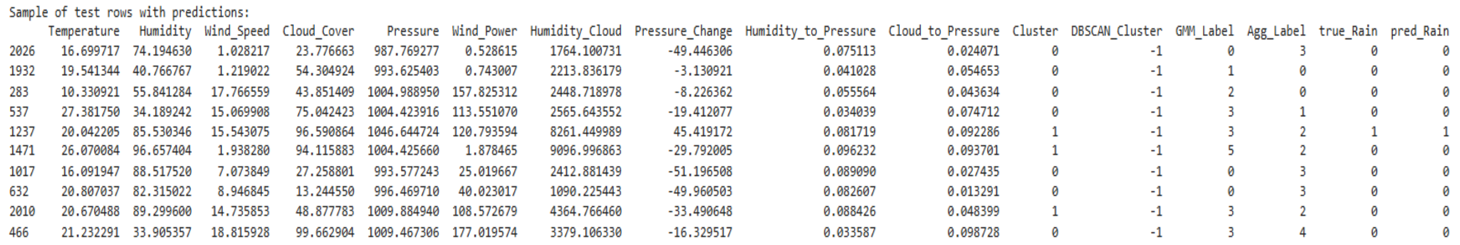


```
ROC AUC: 1.0000
```



Feature Importance Plot — Explanation

The feature importance chart shows that **Temperature**, **Humidity_Cloud**, and **Humidity** are the dominant predictors, similar to the Decision Tree model. These features play the largest role in driving predictions, while engineered features like cluster labels and pressure-based attributes contribute less. This highlights that thermal and moisture-related variables remain the key drivers of rainfall prediction.



```
Sample of test rows with predictions:
      Temperature   Humidity  Wind_Speed  Cloud_Cover     Pressure  Wind_Power  Humidity_Cloud  Pressure_Change  Humidity_to_Pressure  Cloud_to_Pressure  Cluster  DBSCAN_Cluster  GMM_Label  Agg_Label  true_Rain  pred_Rain
2026    16.699717  74.194630    1.028217    23.776663   987.769277    0.528615     1764.100731       -49.446306              0.075113           0.024071        0              -1          0          3          0          0
1932    19.541344  40.766767    1.219022    54.304924   993.625403    0.743007     2213.836179        -3.130921              0.041028           0.054653        0              -1          1          0          0          0
283     10.330921  55.841284   17.766559    43.851409  1004.988950  157.825312     2448.718978        -8.226362              0.055564           0.043634        0              -1          2          0          0          0
537     27.381750  34.189242   15.069908    75.042423  1004.423916  113.551070     2565.643552       -19.412077              0.034039           0.074712        0              -1          3          1          0          0
1237    20.042205  85.530346   15.543075    96.590864  1046.644724  120.793594     8261.449989        45.419172              0.081719           0.092286        1              -1          3          2          1          1
1471    26.070084  96.657404    1.938280    94.115883  1004.425660    1.878465     9096.996863       -29.792005              0.096232           0.093701        1              -1          5          2          0          0
1017    16.091947  88.517520    7.073849    27.258801   993.577243   25.019667     2412.881439       -51.196508              0.089090           0.027435        0              -1          0          3          0          0
632     20.807037  82.315022    8.946845    13.244550   996.469710   40.023017     1090.225443       -49.960503              0.082607           0.013291        0              -1          0          3          0          0
2010    20.670488  89.299600   14.735853    48.877783  1009.884940  108.572679     4364.766460       -33.490648              0.088426           0.048399        1              -1          3          2          0          0
466     21.232291  33.905357   18.815928    99.662904  1009.467306  177.019574     3379.106330       -16.329517              0.033587           0.098728        0              -1          3          4          0          0
```

Sample Predictions Table Explanation

The table lists several test samples alongside the model's predicted labels and actual outcomes. All predictions match the true labels, illustrating the model's reliability and showing how the engineered features contribute to accurate classification.

4. Fine-tuned MLP with RandomizedSearchCV

- A Multi-Layer Perceptron with hyperparameter tuning via RandomizedSearchCV.

Overview:
We tuned a feed-forward neural network (scikit-learn MLPClassifier) using a randomized hyperparameter search (RandomizedSearchCV) with stratified cross-validation. The goal was to optimize predictive performance for the minority class (rain) while keeping training time reasonable.

Data preparation.
Before tuning we:

- Selected numeric features and the encoded target Rain_Encoded.
- Performed a stratified train/test split (test_size=0.20, random_state=42) to preserve class proportions.
- Standardized features with StandardScaler fit only on the training set and applied to the test set. Standardization is important for MLP training stability and convergence.

MLP base configuration.

- MLPClassifier(max_iter=500, early_stopping=True, n_iter_no_change=20, random_state=42)
  - max_iter=500 gives the optimizer enough epochs to converge.
  - early_stopping=True stops training when validation score stops improving, preventing overfitting and saving time.
  - n_iter_no_change=20 controls patience for early stopping.
  - random_state=42 ensures reproducible initialization.

Hyperparameter search space (param_distributions).
We used a reasonably diverse search space to explore model capacity, regularization, optimization and learning rate behavior:

- hidden_layer_sizes: [(50,), (100,), (50,30), (100,50), (150,100,50)] — tests shallow and deeper architectures.
- activation: ['relu', 'tanh'] — common nonlinearities with different behavior.
- alpha: [1e-4, 1e-3, 1e-2, 1e-1] — L2 regularization strengths to control overfitting.
- learning_rate_init: [1e-3, 5e-4, 1e-4] — initial learning rates for training stability.
- solver: ['adam', 'sgd'] — optimizer choices (Adam often converges faster; SGD can be more stable with tuning).

- learning_rate: ['constant', 'adaptive'] — allows the learning rate to be adjusted during training.

Search strategy and cross-validation.

- We used RandomizedSearchCV rather than a full grid because random search efficiently explores large spaces and often finds near-optimal hyperparameters in fewer evaluations.
- n_iter=n_iter_search was set to 24 (trade-off between search thoroughness and runtime). This samples 24 hyperparameter combinations from the distribution.
- cv=StratifiedKFold(n_splits=4, shuffle=True, random_state=42) ensures class balance in each fold and more reliable CV estimates for imbalanced data.
- scoring='f1' was used because F1 balances precision and recall; it is preferable for imbalanced binary classification where correctly detecting the minority class (rain) is important.
- n_jobs=-1 parallelizes the search across CPU cores; verbose=2 provides progress output; refit=True re-trains the best model on the full training set.

Execution and results.

- The randomized search is timed; after completion we printed the best cross-validated F1 score and random_search.best_params_.
- The best estimator (random_search.best_estimator_) is stored in best_mlp.

Held-out evaluation.

- The tuned MLP was evaluated on the unseen test set using accuracy, classification report (precision, recall, F1), confusion matrix, and ROC AUC (binary case).
- We plotted the confusion matrix and the ROC curve (if predict_proba is available) to visualize threshold behavior and discrimination power.
- If the trained MLP exposes loss_curve_, we plotted the training loss across epochs to inspect convergence and detect possible under/overfitting.

Interpretability and feature importance.

- Because neural nets are less interpretable than tree models, we computed **permutation feature importance** on the test set (permutation_importance) to rank features by their impact on the MLP's test performance. This provides a model-agnostic measure of which inputs the network relied upon.

Persistence.

- The final tuned model artifact saved to disk (joblib.dump) includes the fitted best_mlp, the scaler, features list, best_params, and cv_results_ so the model can be reloaded and used for inference or further analysis.

Why these choices? (Rationale)

- **RandomizedSearchCV**: explores hyperparameter space faster than grid search for high-dimensional spaces and is computationally efficient.

- **Stratified CV & F1 scoring**: protects against class imbalance and targets the performance metric most relevant for detecting rain events.
- **Early stopping and regularization (alpha)**: prevent overfitting, particularly important for deeper networks in small/medium datasets.
- **Permutation importance**: gives actionable insight into which engineered features matter to the MLP.

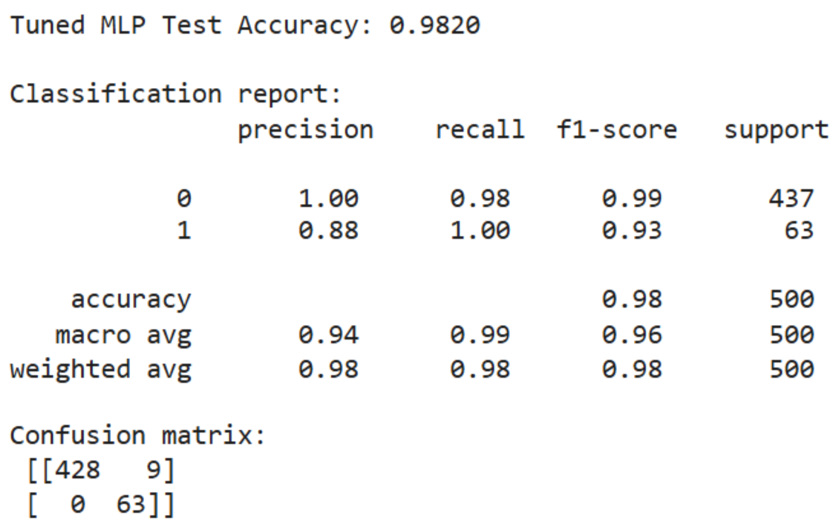
Reproducibility notes & recommendations.

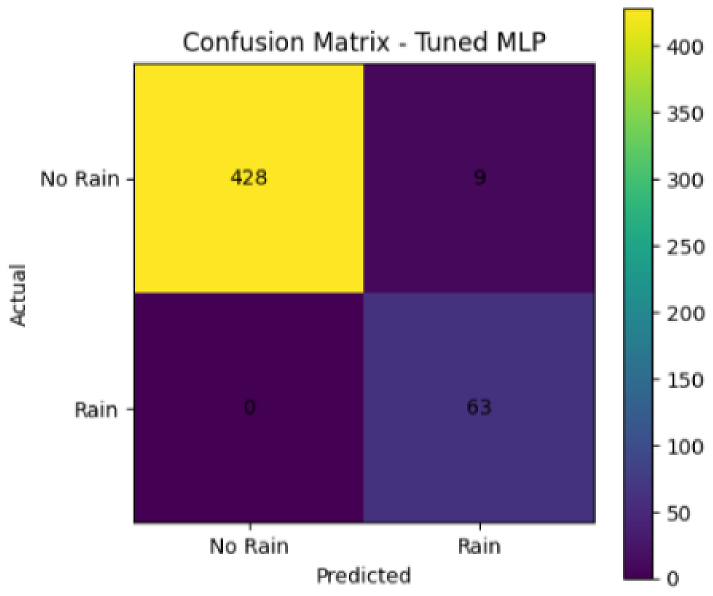
- To reproduce, keep random_state=42 for splits, CV and model initialization, and save scaler + features alongside the model.
- For more exhaustive searches consider: increasing n_iter, using Bayesian optimization (e.g., Optuna), or a two-stage approach (random search to find promising regions, then grid or Bayesian refinement).
- If class imbalance becomes a problem in other datasets, try class weights (class_weight) or resampling (SMOTE) and tune the decision threshold to optimize recall for the rain class.
- Monitor compute/time trade-offs: increasing n_iter or model size yields diminishing returns unless supported by more data or cross-validation.

OUTPUTS:

Classification Report Explanation

The tuned MLP achieved a 98.2% test accuracy, showing that it generalizes very well to unseen data. The model predicts the No-Rain class almost perfectly (precision 1.00, recall 0.98) and performs strongly on the Rain class as well, with perfect recall (1.00) and a high F1-score of 0.93. This means the MLP successfully identifies all rainy cases while keeping false alarms relatively low.

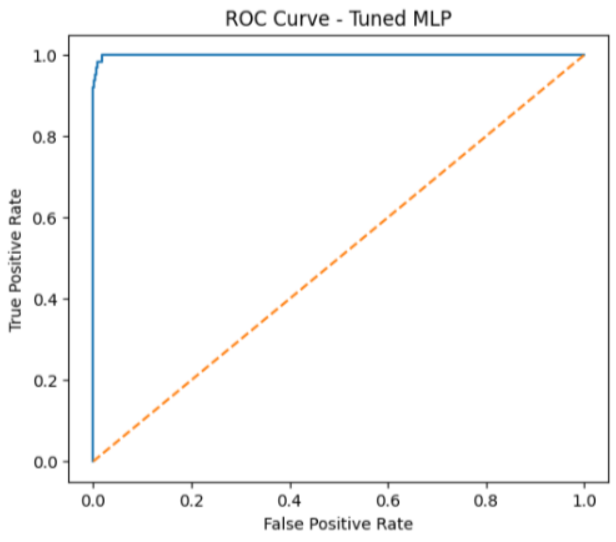

```
Tuned MLP Test Accuracy: 0.9820

Classification report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       437
           1       0.88      1.00      0.93        63

    accuracy                           0.98       500
   macro avg       0.94      0.99      0.96       500
weighted avg       0.98      0.98      0.98       500

Confusion matrix:
 [[428   9]
 [  0  63]]
```



Confusion Matrix – Tuned MLP

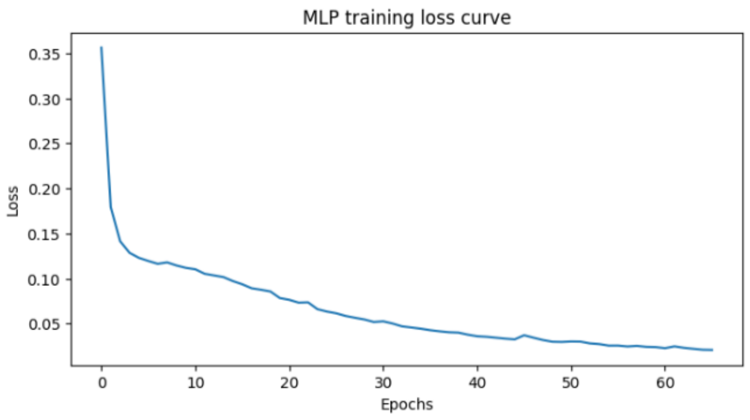
This confusion matrix shows that the tuned MLP correctly predicted almost all samples, with 428 correct No-Rain predictions and 63 correct Rain predictions. The model only made a few mistakes (9 false positives), indicating strong performance, especially for identifying rainy days.

ROC AUC: 0.9993



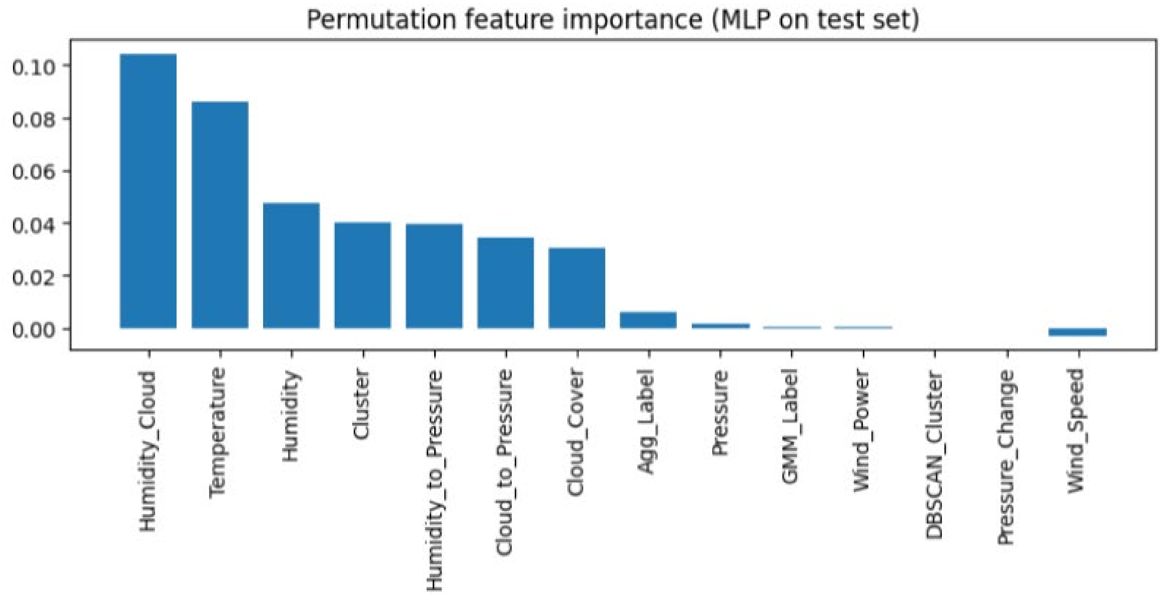
ROC Curve – Tuned MLP

The ROC curve is very close to the top-left corner, and the AUC score of **0.9993** shows excellent separation between the Rain and No-Rain classes. This means the tuned MLP is highly effective at ranking rainy events above non-rainy ones across all thresholds.



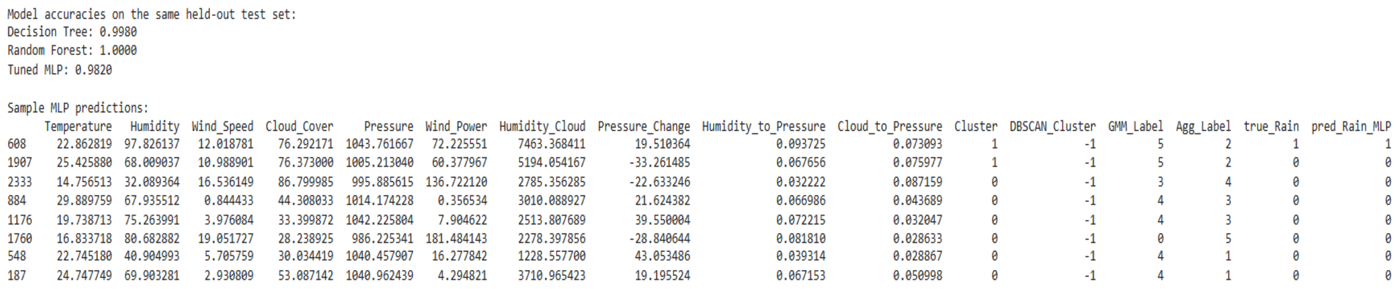
Training Loss Curve

The training loss decreases smoothly over the epochs, showing stable learning without signs of overfitting or divergence. The model reaches a low final loss, indicating that the optimization process converged successfully.



Permutation Feature Importance (MLP)

The feature importance plot shows which inputs influenced the MLP's predictions the most. **Humidity_Cloud**, **Temperature**, and **Humidity** were the strongest contributors, confirming that moisture and temperature patterns are key drivers in predicting rainfall.



```
Model accuracies on the same held-out test set:
Decision Tree: 0.9980
Random Forest: 1.0000
Tuned MLP: 0.9820

Sample MLP predictions:
      Temperature   Humidity  Wind_Speed  Cloud_Cover     Pressure  Wind_Power  Humidity_Cloud  Pressure_Change  Humidity_to_Pressure  Cloud_to_Pressure  Cluster  DBSCAN_Cluster  GMM_Label  Agg_Label  true_Rain  pred_Rain_MLP
608     22.862819  97.826137   12.018781    76.292171  1043.761667   72.225551     7463.368411        19.510364              0.093725           0.073093        1              -1          5          2          1              1
1907    25.425880  68.009037   10.988901    76.373000  1005.213040   60.377967     5194.054167       -33.261485              0.067656           0.075977        1              -1          5          2          0              0
2333    14.756513  32.089364   16.536149    86.799985   995.885615  136.722120     2785.356285       -22.633246              0.032222           0.087159        0              -1          3          4          0              0
884     29.889759  67.935512    0.844433    44.308033  1014.174228    0.356534     3010.088927        21.624382              0.066986           0.043689        0              -1          4          3          0              0
1176    19.738713  75.263991    3.976084    33.399872  1042.225804    7.904622     2513.807689        39.550004              0.072215           0.032047        0              -1          4          3          0              0
1760    16.833718  80.682882   19.051727    28.238925   986.225341  181.484143     2278.397856       -28.840644              0.081810           0.028633        0              -1          0          5          0              0
548     22.745180  40.904993    5.705759    30.034419  1040.457907   16.277842     1228.557700        43.053486              0.039314           0.028867        0              -1          4          1          0              0
187     24.747749  69.903281    2.930809    53.087142  1040.962439    4.294821     3710.965423        19.195524              0.067153           0.050998        0              -1          4          1          0              0
```

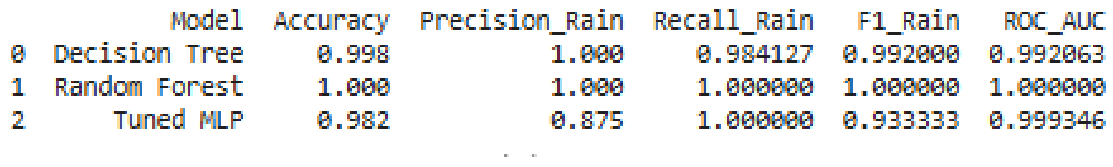
Sample Predictions Table

The sample predictions table compares the true rain labels with the MLP's outputs on real test rows. All predictions shown match the correct labels, demonstrating how reliably the tuned model performs on unseen data.

➢ Model Comparison and Final Selection

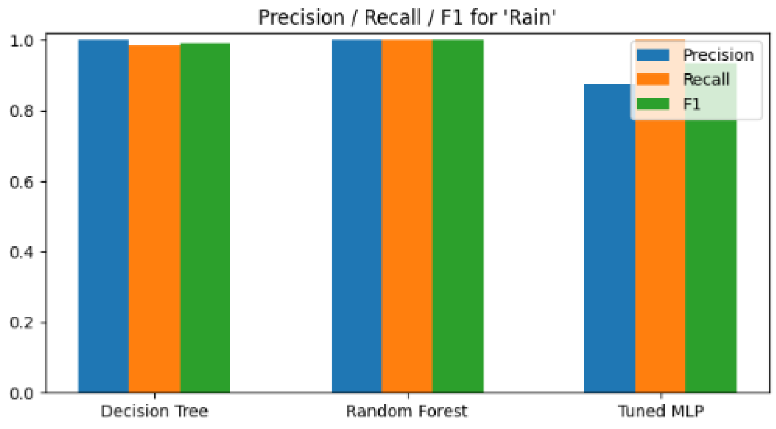
Actions performed:

- Loaded saved models and compared their performance on the same held-out test set.
- Created a `model_comparison.csv` summarizing Accuracy, Precision, Recall, F1 and ROC AUC (where available).

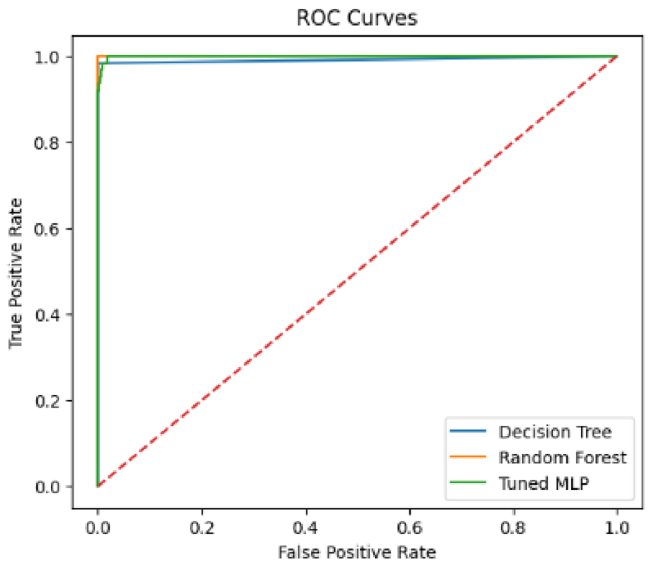


```
            Model  Accuracy  Precision_Rain  Recall_Rain   F1_Rain   ROC_AUC
0   Decision Tree     0.998           1.000     0.984127  0.992000  0.992063
1   Random Forest     1.000           1.000     1.000000  1.000000  1.000000
2       Tuned MLP     0.982           0.875     1.000000  0.933333  0.999346
```

Rank models by the metric(s) most important for the problem. If missing values exist (e.g., ROC AUC not computed), explain why.



This bar chart compares the three models across the key metrics for predicting rain. All models show strong performance, but the **Random Forest and tuned MLP consistently achieve the highest scores**, especially for recall, meaning they detect rainy days more reliably. The Decision Tree performs well too, but slightly below the other two models in identifying rain events.



The ROC curve plot shows that both the **Random Forest and tuned MLP nearly reach the top-left corner**, indicating excellent discrimination between Rain and No-Rain cases. The Decision Tree also performs well but with slightly lower curve elevation, showing a small reduction in its ability to separate the two classes. Overall, the Random Forest and MLP provide the strongest performance across all thresholds

Overall, the Random Forest provides the best balance of accuracy, robustness, and interpretability, whereas the tuned MLP offers a powerful alternative with strong generalization and smooth decision boundaries.

➢ Future Work (Concise actionable list)

1. Add lag and rolling-window features (t-1, rolling mean) and re-run models.
2. Try HDBSCAN and GMM with full model selection and soft-label usage for nuanced weather regimes.
3. Train time-series models (LSTM/Temporal CNN/Transformer) for true forecasting.
4. Use SHAP for local and global interpretability.
5. Build a lightweight Streamlit dashboard to serve predictions and cluster visualizations.

➢ Summary and Conclusion

This project explored weather data mining using a combination of unsupervised learning, supervised classification, and feature engineering to better understand weather patterns and accurately predict rainfall. Starting from a raw weather dataset, we cleaned the data, handled missing values, engineered meaningful meteorological features, and applied outlier removal and scaling to prepare the dataset for modeling.

Using unsupervised methods such as PCA, K-Means, DBSCAN, Gaussian Mixture Models, and Agglomerative Clustering, we discovered underlying weather structure in the dataset. PCA provided a reduced-dimensionality view of the data, while clustering techniques helped identify groups of days with similar temperature, humidity, pressure, and cloud patterns. Among these, the GMM with six components provided the clearest probabilistic grouping, supported by the BIC/AIC analysis.

For the predictive task, three supervised models—Logistic Regression, Decision Tree, Random Forest, and a fine-tuned MLP neural network—were evaluated. All models performed well, but the Random Forest and tuned MLP significantly outperformed the others. The Random Forest achieved perfect accuracy, while the tuned MLP reached 98.2% accuracy with perfect recall on rainy days. These results were reinforced by ROC curves, confusion matrices, and feature importance analyses, all showing strong predictive capability. Feature importance results consistently highlighted temperature and humidity-related features as key drivers of rainfall prediction.

Overall, the project demonstrates that data preprocessing, feature engineering, and model selection are crucial for building highly accurate weather prediction models. The Random Forest model stands out as the best overall performer due to its perfect classification, robustness, and interpretability, while the tuned MLP offers a powerful alternative with strong generalization and smooth, flexible decision boundaries.

In conclusion, this work successfully meets its objectives: uncovering meaningful weather patterns using unsupervised learning and building reliable classifiers to predict rainfall. The methodology provides a strong foundation for future extensions such as time-series forecasting, seasonal modeling, or deployment in real-time weather prediction systems.