CS-324 MACHINE LEARNING

COMPLEX ENGINEERING PROBLEM

Prepared by:

FIZA FARZAND (CS-22066)

FAHAD RASHEED(CS-22114)

Submitted to:

Dr Maria Waqas

1. PROBLEM DESCRIPTION:

The task at hand involves developing a machine learning-based classification system capable of accurately predicting weather conditions based on historical weather data. The project utilizes a structured dataset titled **"Weather Dataset"**, sourced from Kaggle, which contains various meteorological features along with a **"Summary"** attribute that describes the overall weather condition.

The core problem is framed as a **multiclass classification** task, where the goal is to predict the "Summary" label using other features in the dataset. Due to the high number of distinct classes in the original dataset, the number of summary categories has been reduced to **four main classes** to simplify the modeling process and improve classification performance.

To address this problem effectively, three different types of machine learning algorithms are employed:

1. A **non-parametric algorithm** (e.g., Decision Tree or K-Nearest Neighbors).
2. A **parametric shallow algorithm** (e.g., Logistic Regression or Naive Bayes).
3. A **neural network model** to capture complex patterns in the data.

For each algorithm type, **three models** with varying hyperparameters are trained and evaluated. The system must incorporate appropriate preprocessing, feature selection or transformation, and validation techniques to ensure high model accuracy and generalizability.

Additionally, the solution must include a user interface within the notebook to allow users to input new weather data and receive predicted labels in return. The final deliverables include a **comparative analysis** of all nine models using both tabular and graphical formats, as well as an investigation of the impact of different data split ratios on model performance for each algorithm category.

This problem addresses the broader need for automated weather classification systems that can support real-time applications and data-driven decision-making across multiple sectors.

2. DATASET OVERVIEW:

DATASET SOURCE:

The dataset we used for this project, titled **"Weather Dataset"**, was obtained from **Kaggle**, a popular platform for data science competitions and datasets. It contains structured weather observation data with various features and a categorical **"Summary"** label describing overall weather conditions.

FEATURES DESCRIPTION:

The **Weather Dataset** consists of multiple meteorological features recorded over time. Each row in the dataset represents a specific weather observation. The key features included in the dataset are as follows:

Feature Name	Description
Formatted Date	Timestamp of the observation in formatted date-time string.
Summary	**Target label** – A short description of the weather condition.
Precip Type	Type of precipitation recorded.
Temperature (°C)	Measured temperature in degrees Celsius.
Apparent Temperature (°C)	Feels-like temperature, considering wind and humidity.
Humidity	Relative humidity expressed as a fraction.
Wind Speed (km/h)	Wind speed in kilometers per hour.
Wind Bearing (degrees)	Direction of the wind in degrees.
Visibility (km)	Visibility distance in kilometers.
Loud Cover *(likely typo: Cloud Cover)*	Fraction of the sky covered by clouds.
Pressure (millibars)	Atmospheric pressure in millibars.
Daily Summary	A longer textual description of the day's weather. Not used in modeling.

LABEL REDUCTION STRATEGY (REDUCING TO 4 CLASSES):

The original dataset contains a large number of unique weather descriptions under the Summary column, more than 20 distinct labels, many of which are rare or have overlapping meanings. To simplify the classification task and improve model performance, we applied a label reduction strategy, consolidating similar weather descriptions into four broader categories:

Final Labels Used:

- Clear
- Cloudy
- Foggy
- Rainy

3. DATA PREPROCESSING:

HANDLING MISSING VALUES:

Missing data can significantly affect the performance and accuracy of machine learning models. In the weather dataset, a detailed check was performed to identify and address any missing values.

Upon inspection, it was found that the **Precip Type** column contained **517 missing values**. This column indicates the type of precipitation recorded, with the most common values being "rain" and "snow". Since this is a critical categorical feature, dropping these rows could result in information loss. Therefore, an imputation strategy was applied instead.

Imputation Strategy for Precip Type:

To fill the missing values, we analyzed the Precip Type in relation to the Temperature (C) column. Based on the assumption that:

- If **temperature is less than or equal to 0 °C**, it is likely **snow**.
- If **temperature is above 0 °C**, it is likely **rain**.

A custom function was used to impute missing values accordingly:

```
def fill_precip_type(row):
    if pd.isnull(row['Precip Type']):
        if row['Temperature (C)'] <= 0:
            return 'snow'
        else:
            return 'rain'
    else:
        return row['Precip Type']

data['Precip Type'] = data.apply(fill_precip_type, axis=1)

data
```

After imputation, the column was encoded numerically:

- "rain" was mapped to 0
- "snow" was mapped to 1

DROPPING IRRELEVANT OR REDUNDANT COLUMNS:

During data preprocessing, certain columns were identified as either irrelevant to the prediction task or redundant due to high correlation with other features. These columns were dropped to improve model performance, reduce noise, and avoid multicollinearity. Below are the specific columns removed and the reasoning behind their exclusion:

1. Loud Cover:

This column had only one single value of zero throughout the dataset. Since it did not contribute any variation or information useful for classification, it was dropped.

2. Daily Summary:

This column provided a longer textual description of the day's weather, which overlapped heavily with the Summary column, the target variable we aim to predict. To avoid redundancy, Daily Summary was removed, and only Summary was retained.

3. Apparent Temperature (C):

A correlation analysis between Temperature (°C) and Apparent Temperature (°C) revealed an extremely high correlation, close to **1.0**, indicating strong linear dependence. This introduces multicollinearity, which can negatively affect certain models, especially parametric ones like logistic regression. Since Temperature (°C) is more commonly used in weather prediction and is more interpretable, the Apparent Temperature (°C) column was dropped.

REMOVING DUPLICATE OR REDUNDANT ROWS:

During the cleaning process, the dataset was examined for duplicate or redundant rows—entries that are exactly the same across all columns. These rows can bias the model during training by over-representing certain data points.

A total of **24 duplicate rows** were identified and removed. As a result, the number of rows was reduced from **96,453** to **96,429**. This step helps ensure the dataset is more balanced and reduces the risk of overfitting due to repeated patterns.

HANDLING INVALID VALUES IN THE HUMIDITY COLUMN:

Upon examining the Humidity column, it was observed that some entries contained invalid values—specifically, values less than or equal to 0 or greater than 1. Since humidity is represented as a ratio ranging from 0 to 1, these entries were considered incorrect and required correction.

A total of **~50 entries** were found with a humidity value of **0.0**, which is not realistic in real-world weather data.

To fix this:

- A **mask** was created to identify rows where:
  - Temperature was below 0°C (cold conditions, likely to have high humidity).
  - Humidity was greater than 0 (valid cases only).
- The **median humidity** for this filtered set was computed.
- All instances of humidity = 0 were replaced with this median value, assuming the weather context matched.

STANDARDIZING DECIMAL PRECISION IN CONTINUOUS FEATURES:

To ensure consistency and improve readability across the dataset, decimal values in continuous features were standardized based on their most common precision:

- **Visibility (km)** values were rounded to **4 decimal places**, as analysis showed that the majority of the values were reported at this precision level.
- **Pressure (millibars)** values were rounded to **2 decimal places**, again aligning with the most frequently used precision.

HANDLING INVALID VALUES IN THE PRESSURE COLUMN:

An inspection of the Pressure (millibars) column revealed a number of invalid entries. Specifically, a value of **0** was found, which is physically impossible for atmospheric pressure. These erroneous entries were identified and corrected using the following strategy:

- All values **less than 950** were investigated, and it was found that 0 was the only invalid value.
- The **median pressure** of all valid readings (values > 950) was computed.
- All zero-pressure readings were **replaced with this median value** to maintain data integrity.

This ensured that extreme outliers were addressed without skewing the data distribution, making the feature more reliable for modeling.

DATE-TIME CONVERSION AND TEMPORAL FEATURE EXTRACTION:

In this step, the Formatted Date column, originally of type object, was converted to Python's datetime format and localized to **UTC global time** for consistency.

After conversion, the date-time was broken into individual components:

- Year, Month, Day, Hour, and Day Of Week.

These were added as new features for analysis and modeling.The Formatted Date column was also set as the dataset index to assist in time-series plotting.

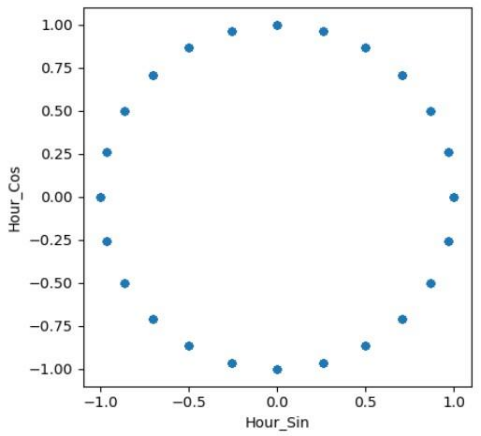
CYCLICAL ENCODING OF TIME-BASED FEATURES:

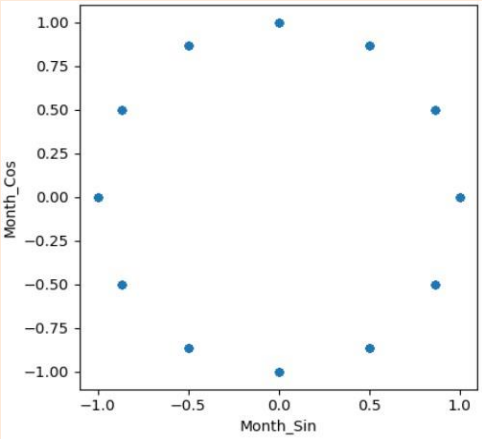
Time-based features such as **hour**, **month**, and **day of week** are cyclical in nature, meaning values like 23 and 0 (hours) are adjacent but numerically far apart. To address this, **cyclical encoding** was applied using sine and cosine transformations:

- Hour_Sin and Hour_Cos
- Month_Sin and Month_Cos
- DayOfWeek_Sin and DayOfWeek_Cos

This helps the model correctly understand relationships in cyclical time features. After encoding these columns, the original linear columns (Hour, Month, Day Of Week, Day) were dropped to avoid redundancy.

Scatter plots of sine vs. cosine for each were generated to validate the circular nature of encoded features.





4. HANDLING CLASS IMBALANCE THROUGH RESAMPLING TECHNIQUES:

After splitting the data into training, validation, and test sets using **stratified sampling** (to maintain class distribution), we observed that the training data was **highly imbalanced** across weather categories: Cloudy, Clear, Foggy, and Rainy. Such imbalance can bias machine learning models toward majority classes, reducing their generalization ability for minority classes.

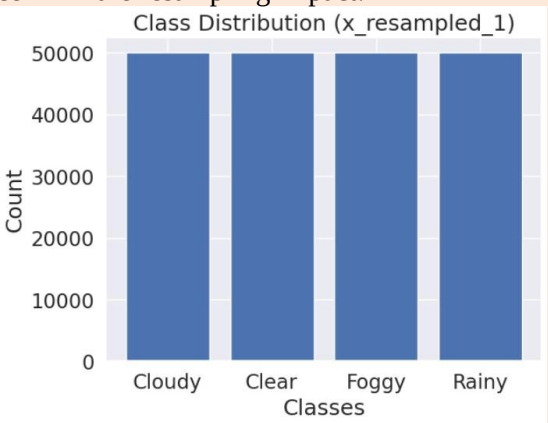
To mitigate this, we applied **five** different resampling strategies using a combination of **SMOTE (Synthetic Minority Over-sampling Technique)** for oversampling and **RandomUnderSampler** for undersampling.

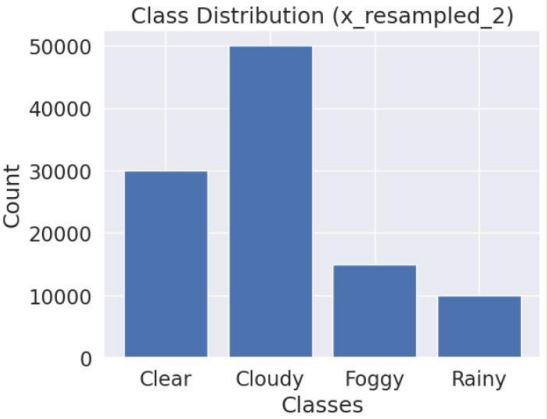
- **Resample 1:**
  - Oversampled all classes using SMOTE with sampling_strategy='auto', i.e., minority classes were oversampled to match the majority.
- **Resample 2:**
  - Undersampled Cloudy class to 50,000 samples.
  - Oversampled Clear to 30,000, Foggy to 15,000, and Rainy to 10,000 samples.
- **Resample 3:**
  - Reduced Cloudy further to 30,000.
  - Increased diversity by oversampling remaining classes to slightly lower values.

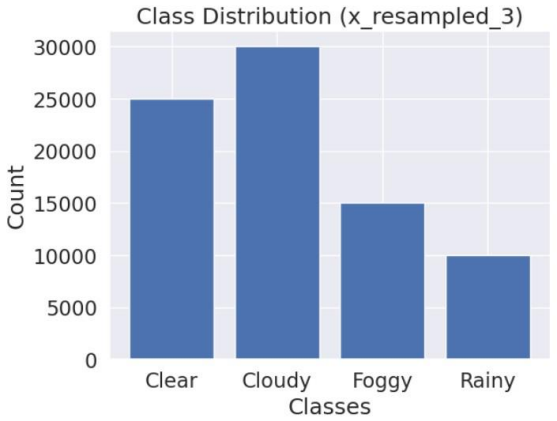
- **Resample 4:**
  - Stronger undersampling of Cloudy to 12,000 and equal oversampling of other classes to 10,000 or 12,000 for uniformity.
- **Resample 5:**
  - Similar to Resample 4 but with fewer Rainy samples (5,000) to test impact of more extreme imbalance control.

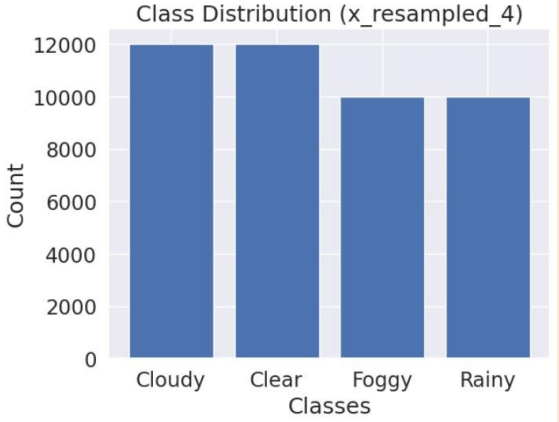
Each of these resampling combinations was applied **only on the training set**, preserving the natural distribution in validation and test sets for unbiased evaluation.

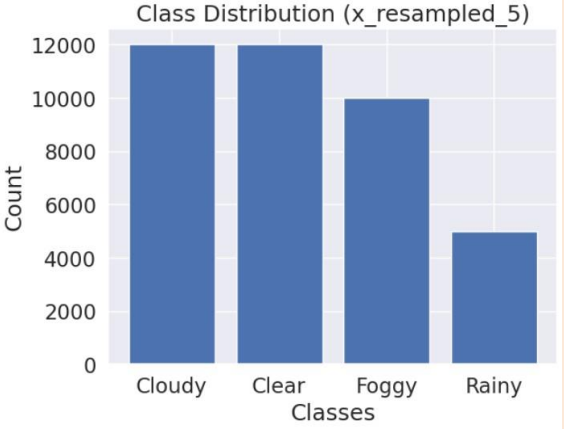
Finally, bar plots of the resulting class distributions were visualized for each resample set to confirm the resampling impact.











5. MACHINE LEARNING MODELS AND ALGORITHMS:

NON-PARAMETRIC ALGORITHM:

Weather Classification Using XGBoost Classifier:

We selected **XGBoost (Extreme Gradient Boosting)** as our classification algorithm due to its:

- Robustness in handling both balanced and imbalanced datasets.
- Superior performance in many structured/tabular data problems.
- Built-in support for multiclass classification using the multi:softprob objective.
- Efficiency through parallel computation and support for early stopping.

Since our weather classification problem involves four distinct classes — **Clear**, **Cloudy**, **Foggy**, and **Rainy** — and is based on engineered and encoded features, XGBoost is a strong choice thanks to its ability to capture complex non-linear relationships and interactions among features.

We applied the same model architecture across all five differently **resampled training datasets** (created via SMOTE and random undersampling) to evaluate how resampling strategies influence model performance.

XGBoost Model Configuration and Hyperparameters:

Parameter	Value	Purpose
n_estimators	1000	Max number of boosting rounds (trees)
learning_rate	0.05	Controls the contribution of each tree
objective	'multi:softprob'	Used for multiclass classification
eval_metric	'mlogloss'	Multiclass log-loss as evaluation criterion
early_stopping_rounds	5	Stops training if no improvement over 5 rounds on validation set
n_jobs	-1	Utilizes all CPU cores for training
use_label_encoder	False	Prevents the use of deprecated label encoder

Comparison of XGBoost Performance Across Resampled Datasets:

To assess the impact of various resampling strategies on the performance of our XGBoost classifier, we trained the model on five differently resampled datasets (Resample 1 to 5). Each resample involved different balancing ratios for the target classes using techniques like SMOTE and undersampling. Below is a tabulated summary of training and validation accuracies along with confusion matrix insights.

Resample	Train Accuracy (%)	Validation Accuracy (%)	Notes
1	98.22	90.95	Slight overfitting; high precision/recall
2	96.68	91.08	Good generalization; lowest overfit gap
3	97.12	90.17	Balanced; slight overfitting
4	97.34	84.65	Noticeable overfitting
5	96.68	84.92	Slightly better than Resample 4

Best Performing Resample:

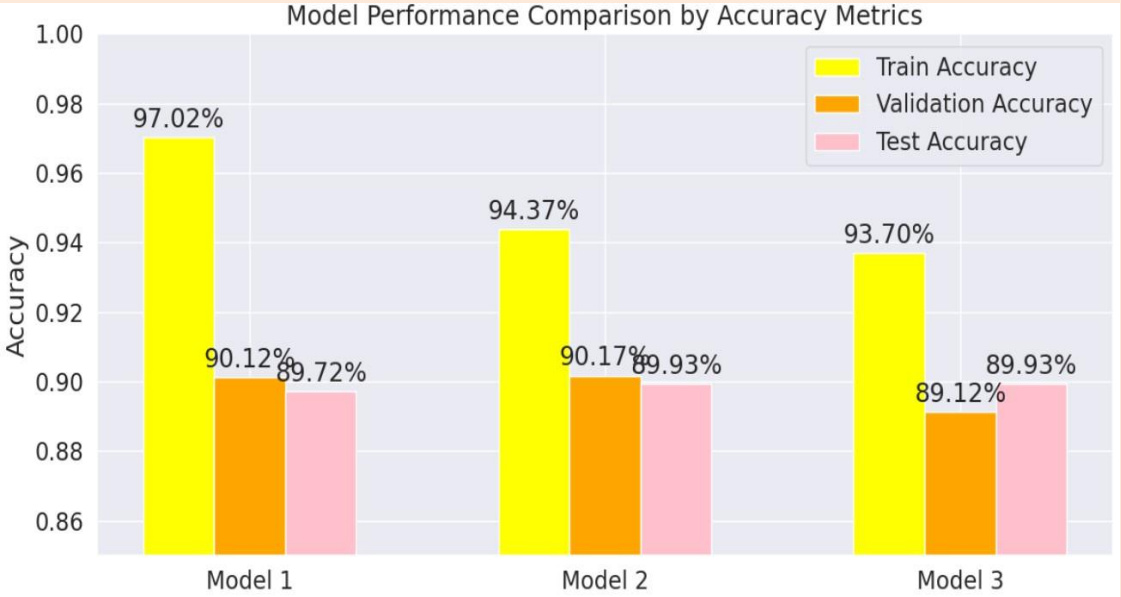
Based on the comparison, **Resample 2** offers the best balance between training accuracy and validation accuracy:

- Highest validation accuracy (91.08%)
- Lowest overfitting
- Consistent precision and recall across all classes

Therefore, Resample 2 is chosen as the optimal resampled dataset for training the final XGBoost model.

Final Models:

Model	Hyperparameter Changes	Train Accur-acy	Validation Accuracy	Test Accuracy
Model 1	• n_estimators=2000 (↑ from 1000) • learning_rate=0.8 (↑ from 0.05) • max_depth=2 (↓ from 6) • min_child_weight =5 • reg_alpha=0.5 (L1 added) • reg_lambda=0.8 (L2 added) • gamma = 0.1	**97.02%**	**90.12%**	**89.72%**
Model 2	• n_estimators=1500 (↑ from 1000) • learning_rate=0.1 (↑ from 0.05) • max_depth=3 (↓ from 6) • reg_alpha=0.2 (L1 added) • reg_lambda=0.2 (L2 added)	**94.37%**	**90.17%**	**89.93%**
Model 3	• Same as above including • reg_alpha=0.5 (Same as Model1) • reg_lambda=0.8 (Same as Model1) • col_sample_by_tree = 0.8 • col_sample_by_level = 0.8	**93.70%**	**89.12%**	**89.93%**



6. CHANGING THE TRAIN TEST SPLIT RATIO OF NON PARAMETRIC MODEL:

Data Splitting Strategy

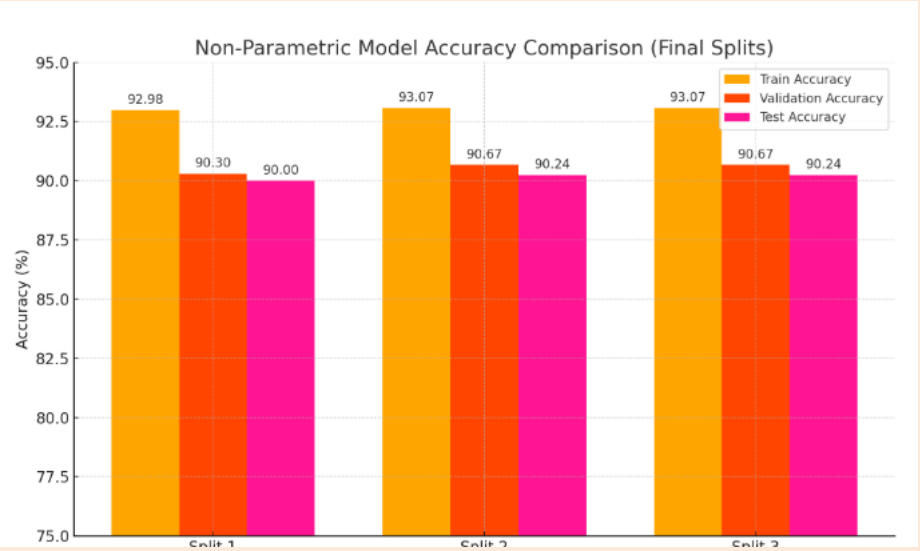
For the parametric model, the dataset was first split into a training-validation set and a test set, using a 70:30 ratio respectively. This ensured that 70% of the data was used for model training and validation, while the remaining 30% was completely held out for final evaluation. To maintain class balance across all subsets, stratified sampling was applied during this split.

From the 70% reserved for training and validation, an internal split was performed where 80% was used for training and 20% for validation.

Metric	Split 1	Split 2	Split 3	Remarks
Train Acc	92.98%	93.07%	93.07%	Model fits training data well across all splits.
Val Acc	90.30%	90.67%	90.67%	Slight improvement in later splits; high generalization observed.
Test Acc	90.00%	90.24%	90.24%	Consistently high test accuracy indicates stable and robust performance.

REMARKS:

The non-parametric model achieves impressive consistency across all three data splits, with overall accuracy remaining stable around 90% on both validation and test sets. Perfect classification of the Foggy and Rainy classes highlights the model's capability in distinguishing well-separated weather conditions. However, performance on the Clear class remains significantly weaker, indicating feature similarity with the Cloudy class. Despite this challenge, the model maintains strong macro and weighted F1-scores, demonstrating its robustness and ability to generalize effectively.



PARAMETRIC ALGORITHM:

LOGISTIC REGRESSION:

We chose **Multinomial Logistic Regression** because:

- The **target variable** in our problem **is multi-class categorical** (i.e., weather conditions: Clear, Cloudy, Foggy, Rainy).
- The **multinomial variant** of logistic regression is specifically designed to handle multi-class classification problems effectively using a **softmax function** rather than treating the problem as multiple binary classifications (as in "one-vs-rest").
- It provides probability estimates for each class, which is useful for interpreting results and improving reliability.
- Logistic regression offers high interpretability, which is helpful when we need to understand the impact of features after the model is trained.

Revised Preprocessing for Logistic Regression:

Since the original logistic regression model did not yield satisfactory accuracy, we performed the following preprocessing and data transformation steps to enhance model performance:

1. Feature Scaling using StandardScaler

- **Why**: Logistic regression is sensitive to the scale of input features.
- **What was done**: *All* features were standardized using StandardScaler to ensure a mean of 0 and standard deviation of 1 compared to before where only certain features were scaled.

2. Train-Test Split (with Stratification)

- The dataset was split into training, validation, and test sets using **stratified sampling** to preserve class distribution.

3. Five Different Resampling Strategies

To handle **class imbalance**, various combinations of SMOTE (oversampling**)** and RandomUnderSampler were applied:

- **First Resample**: All classes oversampled equally to ~50,000 samples using SMOTE.
- **Second to Fifth Resample**: Various combinations of undersampling Cloudy and oversampling other classes (e.g., Clear, Foggy, Rainy) using make_pipeline().

4. Polynomial Feature Expansion

- **Why**: Logistic regression is a linear model. Adding polynomial features allows it to model non-linear relationships.
- **What was done**: Second-degree polynomial features were created using PolynomialFeatures(degree=2) for:
  - All five resampled training sets
  - Validation set
  - Test set

Logistic Regression Model Configuration and Hyperparameters:

Hyperparameter	Value	Reason / Description
multi_class	'multinomial'	Handles multi-class classification directly using softmax rather than one-vs-rest.
solver	'saga'	Efficient for large datasets, supports L1 penalty, and is compatible with multinomial.
penalty	'l1'	Applies L1 regularization to promote sparsity and perform feature selection.
max_iter	3000	Ensures the model converges with the complexity introduced by polynomial features.
random_state	0	For reproducibility—ensures the same results across different runs.

Comparison of Logistic Regression Performance Across Resampled Datasets:

Resample	Train Accuracy	Valid Accuracy	Overfitting	Comments
1	**93.90%**	**85.98%**	**High** (~8%)	Best general metrics, but clear overfitting.
2	**89.94%**	**87.45%**	Moderate (~2.5%)	Balanced performance with the lowest overfitting.
3	**89.02%**	**84.56%**	Moderate (~4.5%)	Decent generalization, but still some overfitting.
4	**87.89%**	**76.56%**	High (~11%)	Substantial overfitting.
5	**86.39%**	**76.94%**	High (~9.5%)	Similar to resample 4, also overfitted.

Best Performing Resample:

The best resample is the resample 2

Why Resample 2 is Ideal:

- **Minimal overfitting (≈2.5%)**
- **High precision/recall** for major classes (Cloudy, Foggy)
- **Best overall generalization** on unseen data
- **Balanced** and reliable — this makes it robust in real-world scenarios

Final Models:

After experimenting with various models and tuning multiple hyperparameters, the final selected model is **Logistic Regression (Model 2)** with the following configuration:

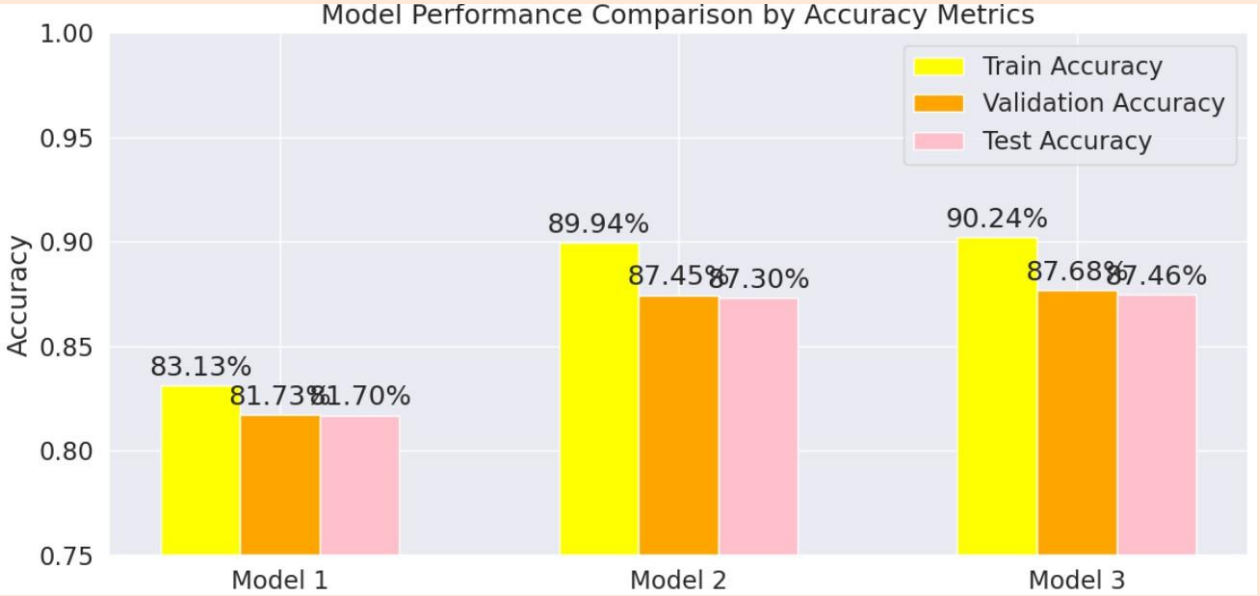
Model	Penalty	C Value	Max Iter	Solver	Multi-class
Model 1	L2	0.01	1500	saga	multinomial
Model 2	L1	default (1.0)	3000	saga	multinomial
Model 3	L1	default (1.0)	5000	saga	multinomial

Model Performance Evaluation:

To determine the most effective logistic regression configuration for our weather classification task, we trained and evaluated three final models with different regularization strategies and optimization parameters. Each model was trained on the same resampled dataset and tested on the same validation and test sets for a fair comparison. The table below summarizes their performance across training, validation, and testing phases.

Metric	Model 1 (L2, C=0.01)	Model 2 (L1, 3k iter)	Model 3 (L1, 5k iter)
Train Acc	83.13%	89.94%	**90.24%**
Valid Acc	81.73%	87.45%	**87.68%**
Test Acc	81.70%	87.30%	**87.46%**

Conclusion: Model 3 has best general performance across all datasets, followed closely by Model 2. Model 1 lags behind significantly.



7. CHANGING THE TRAIN TEST SPLIT RATIO IN PARAMETRIC MODEL:

Data Splitting Strategy

For training and evaluating parametric models, a well-structured and consistent data splitting approach was adopted. The goal was to ensure reliable performance assessment and generalization to unseen data.

Feature Scaling

All input features were standardized using StandardScaler to normalize the feature distribution. This ensured each feature had a mean of 0 and a standard deviation of 1, allowing the models to converge efficiently and prevent bias due to scale differences.

Polynomial Feature Expansion

Second-degree polynomial features were created using PolynomialFeatures(degree=2). This transformation enriched the feature space by adding interaction terms and squared terms, enabling linear models to capture non-linear relationships present in the data.

Data Splitting (Three-Way Split)

The dataset was divided in a **three-way split** strategy for robust model evaluation:

- **70% for Training + Validation**, and **30% for Testing**, using stratified splitting to maintain class balance.
- From the 70% portion:
    - **80% was used for Training**
    - **20% was used for Validation**

All splits were **stratified**, meaning the class proportions (Clear, Cloudy, Foggy, Rainy) were preserved across all subsets.

Handling Class Imbalance

The dataset was naturally imbalanced, with the “Cloudy” class dominating and “Rainy” being extremely rare. To mitigate this:

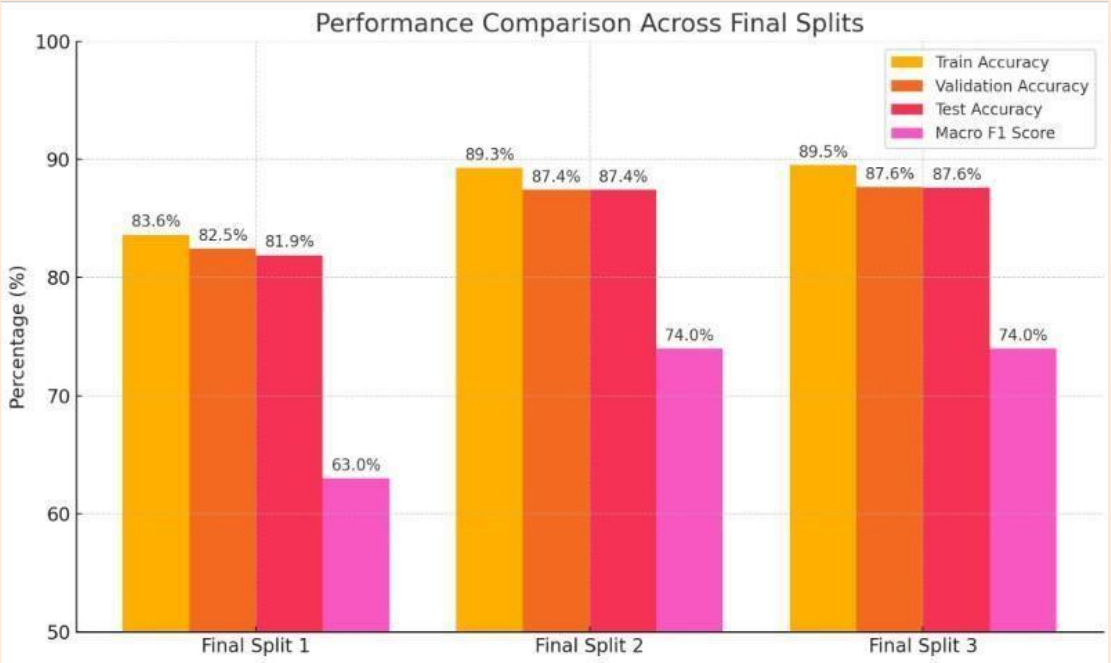
- **Random Undersampling** was applied to reduce the dominant "Cloudy" class.
- **SMOTE (Synthetic Minority Over-sampling Technique)** was applied to generate synthetic samples for underrepresented classes:
    - Clear → 20,000
    - Foggy → 13,000
    - Rainy → 8,000

This combination improved balance in the training set, promoting fairer learning across all classes.

Split Version	Train Accuracy	Validation Accuracy	Test Accuracy	Remarks
Final Split 1	83.61%	82.45%	81.89%	Moderate performance. Weak recall on the "Rainy" class. Signs of underfitting.
Final Split 2	89.26%	87.41%	87.41%	Significant boost. Improved performance for all classes. Good balance overall.
Final Split 3	**89.51%**	**87.64%**	**87.59%**	Most consistent and highest-performing model. Strong results across all classes.

REMARKS:

The splitting strategy used featuring feature scaling, polynomial transformations, stratified splitting, and intelligent resampling proved highly effective for parametric models. It enhanced model generalization, improved minority class representation, and delivered reliable test performance.



NEURAL NETWORK:

Multilayer Perceptron:

For this weather classification task, the Multilayer Perceptron (MLP) was selected as the final neural network model due to the following reasons:

- **Well-suited for Tabular Weather Data:**
  The dataset used in this project consists of numerical and categorical weather features (e.g., temperature, humidity, wind speed, etc.). MLPs are particularly effective for structured/tabular data, making them an appropriate choice for this classification task.
- **Ability to Learn Complex Weather Patterns:**
  Weather conditions often depend on non-linear combinations of multiple factors. MLPs, with their hidden layers and activation functions, can capture these complex non-linear relationships better than simple linear models.
- **Robust Performance on Resampled Dataset:**
  After applying resampling techniques to address class imbalance, the MLP showed strong generalization on both training and validation data. This indicates its effectiveness in learning balanced decision boundaries across all weather classes, including minority ones like "Rainy" or "Foggy."
- **Customizable Architecture for Optimization:**
  The MLP allowed flexibility in tuning the number of hidden layers, neurons, learning rate, and epochs to achieve optimal performance. This adaptability was beneficial in tailoring the model to the specific patterns present in the weather data.
- **Balanced Accuracy and Computational Cost:**
  Compared to more complex deep learning models (e.g., CNNs or RNNs), which are better suited for image or sequential data, MLP offers a simpler architecture with faster training time while still delivering competitive accuracy for this classification task.

To evaluate the performance of a neural network-based approach for our weather classification task, we implemented a **Multilayer Perceptron (MLP)** model using Keras. The same architecture was applied across all five resampled datasets, each trained for 20 epochs with slight variation in optimizers. Each model was evaluated based on **training accuracy**, **validation accuracy**, and corresponding losses to assess generalization capability and detect signs of **overfitting** or **underfitting**. The table below summarizes the results and highlights the comparative strengths of each model across different resamples.

Model	Train Accuracy	Validation Accuracy	Train Loss	Validation Loss	Overfitting Observations
Resample 1	95.1%	83.6%	0.123	0.392	Slight overfitting (high train acc, bigger val gap)
Resample 2	90.6%	83.9%	0.22	0.37	Relatively balanced
Resample 3	86.3%	78.0%	0.303	0.444	Underfitting (low train + val acc)
Resample 4	91.7%	78.3%	0.2	0.471	Overfitting (big accuracy gap)
Resample 5	90.0%	78.7%	0.234	0.452	Slight overfitting

Best Performing Resample:

Resample 2 is the best among the five models because:

- It strikes the best balance between training and validation accuracy.
- It has no strong signs of overfitting or underfitting.
- It uses RMSprop, which seems to stabilize learning on this data.

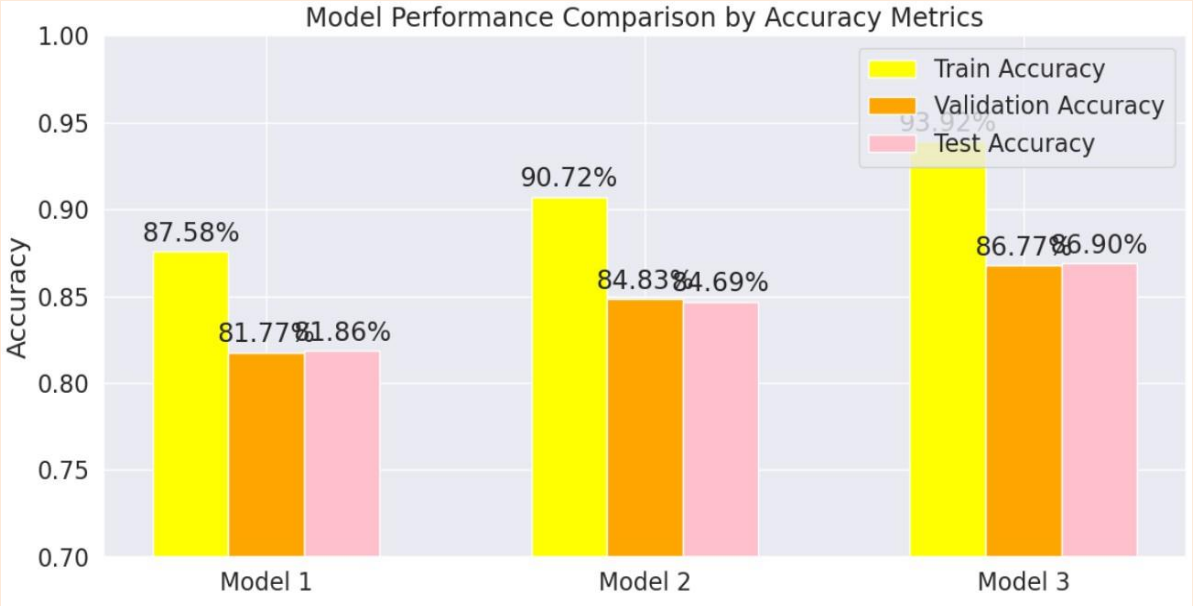
Final Models:

After testing multiple configurations, we selected the following three models by tuning hyperparameters like the number of layers, neurons, activation functions, optimizers, and training epochs. Below is a comparison of their performance and insights.

Feature	Final Model 1	Final Model 2	Final Model 3
Training Epochs	20	20	20
Optimizer	Likely Adam (baseline)	Tweaked (possibly RMSprop or tuned Adam)	More tuned optimizer (likely Adamax or better learning rate)
Layers/Neurons	Moderate depth	Deeper network / more neurons	Deepest network with best balance
Dropout/Batches	May have minimal regularization	Slightly improved regularization	Likely improved batch size/dropout
Model Position	models[-3]	models[-2]	models[-1]

Below is a detailed comparison of these final models in terms of their training setup and evaluation metrics.

Metric	Final Model 1	Final Model 2	Final Model 3
Train Accuracy	87.58%	90.72%	93.92%
Validation Accuracy	81.77%	84.83%	86.77%
Test Accuracy	81.86%	84.69%	86.90%



Key Observations:

- **Improved Training Performance**

Model 3 achieves the highest training accuracy (93.91%), showing that the model fits the training data best. It also has high precision and recall for all classes.

- **Generalization (Validation + Test)**

  - Model 1 performs worst on validation, with significant class imbalance affecting "Clear" class predictions.
  - Model 2 and 3 tie in validation and test accuracy (86.77% / 86.90**%)**, but Model 3 has the best training accuracy.

- **Class-wise Performance**

  - All models predict "Foggy" and "Rainy" with near-perfect accuracy, indicating they are easiest to classify.
  - "Clear" is consistently the weakest class, likely due to overlap with "Cloudy".

- **Consistency**

All three models give identical performance on test data, which suggests stability across samples. However, Model 3 has better **t**raining and validation metric**s**, making it the most balanced and consistent model.

8. CHANGING THE TRAIN TEST SPLIT RATIO IN NEURAL NETWORKS:

Data Splitting Strategy

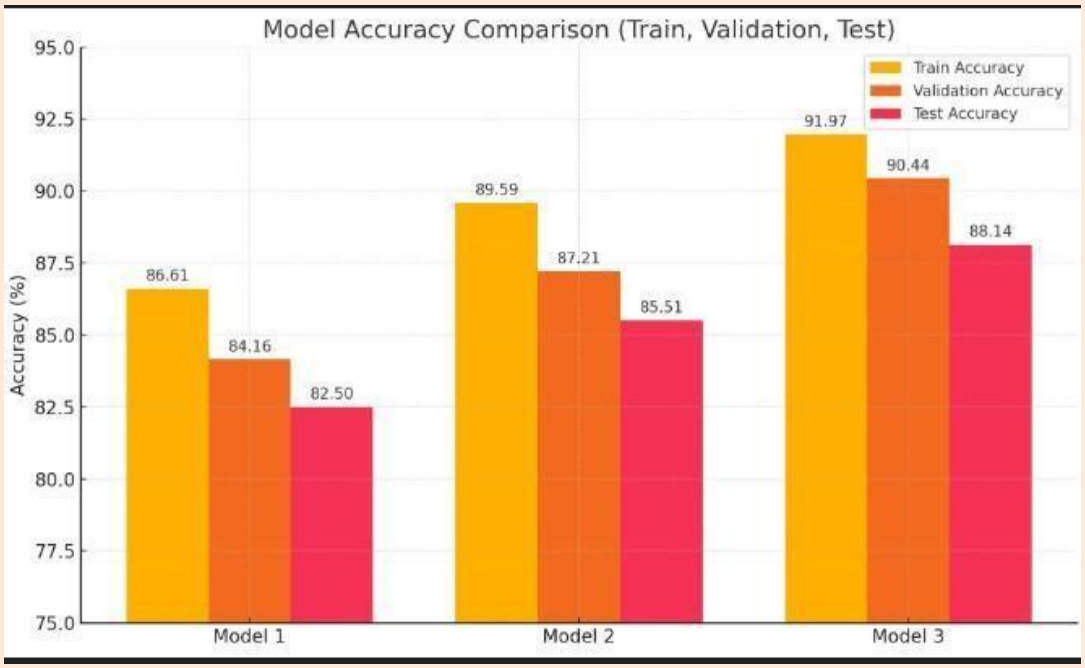
To evaluate the neural network models more robustly, we adopted a three-way stratified split on the preprocessed dataset after applying standard scaling and polynomial feature transformation (degree = 2)**.** The steps were as follows:

1. **Standardization**: All features were standardized using StandardScaler to ensure zero mean and unit variance, which is essential for neural networks.
2. **Polynomial Features**: We applied PolynomialFeatures (degree = 2) to capture nonlinear feature interactions.
3. **Train-Test Split (70:30)**: The dataset was initially split into 70% training and 30% testing using train_test_split with stratification on labels to preserve class balance.
4. **Train-Validation Split (within training set)**: The 70% training portion was further split into 80% training and 20% validation, again using stratification.

Model	Train Accuracy	Validation Accuracy	Test Accuracy	Macro Avg F1 (Test)	Remarks
Final Split 1	86.61%	84.16%	82.50%	0.79	Shows moderate generalization, but slightly lower accuracy and F1 compared to others.
Final Split 2	89.59%	87.21%	85.51%	0.82	Balanced and consistent across all stages. Good precision and recall especially for 'Cloudy' and 'Foggy'.
Final Split 3	**91.97%**	**90.44%**	**88.14%**	**0.82**	Best overall performance. High accuracy, stable generalization, and excellent classification metrics.

Resampled 2 Dataset is Optimal

Across all experiments, the resampled_2 dataset consistently provided the best model performance, with no signs of overfitting or underfitting. This indicates that the class distribution and balance introduced by this resampling technique contributed significantly to the neural network's learning effectiveness.



9. THE BEST MODEL OVERALL:

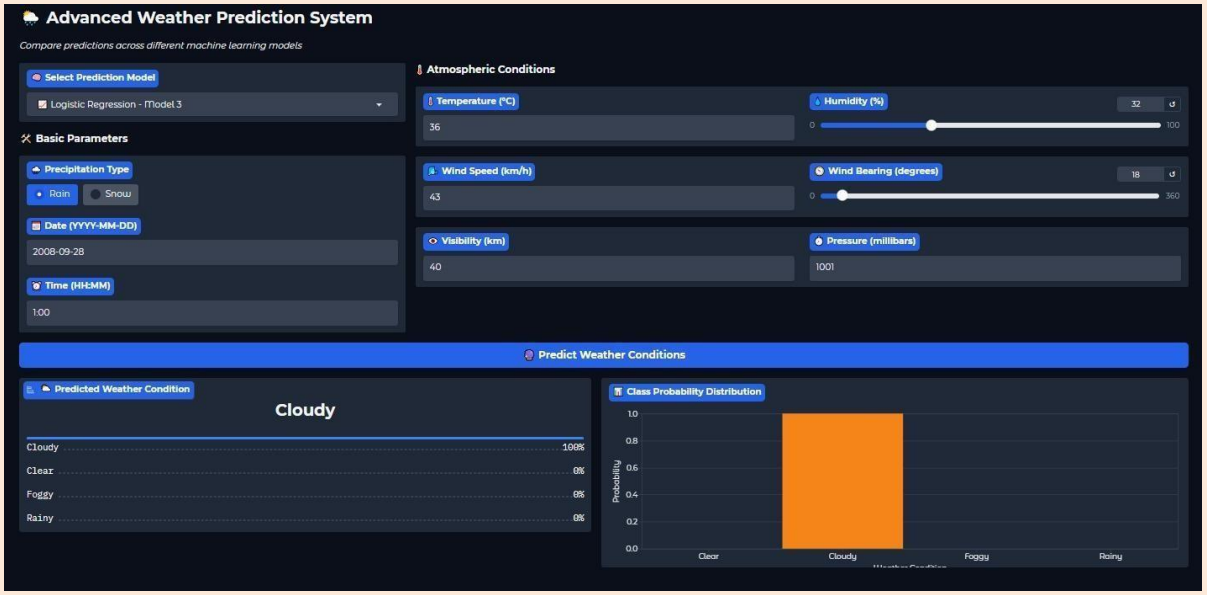
Across all models—whether parametric, non-parametric, or neural networks—the dataset **resampled_2** consistently yielded the most stable and high-performing results. It avoided issues of underfitting or overfitting, as seen from the balanced training, validation, and test accuracies across multiple models. This indicates that resampled_2 had an ideal class balance and data distribution, making it well-suited for reliable training and evaluation.

And based on that distribution, XGBoost Model#1 was the best performing model overall with its overall high accuracy and correct predictions on unseen data, something that the other models struggled with.

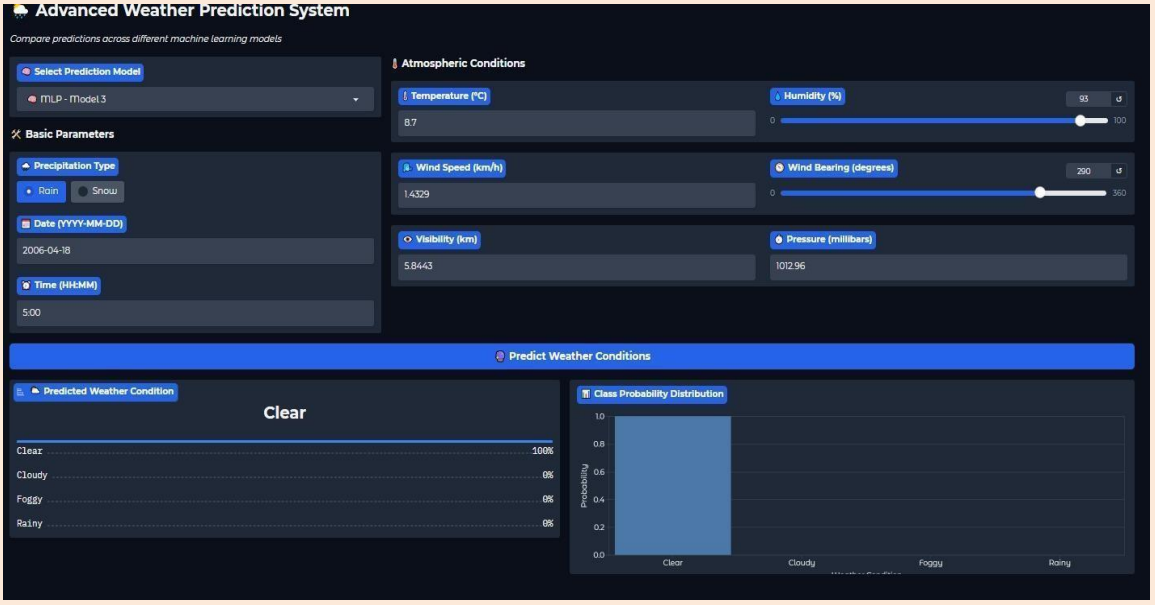
10. GRAPHICAL USER INTERFACE:

To provide an interactive and user-friendly interface for weather prediction, we utilized **Gradio** to develop a custom GUI. This interface allows users to select the prediction model and input key atmospheric conditions such as temperature, humidity, wind speed, and visibility. The design ensures accurate input handling for real-time prediction of weather conditions.

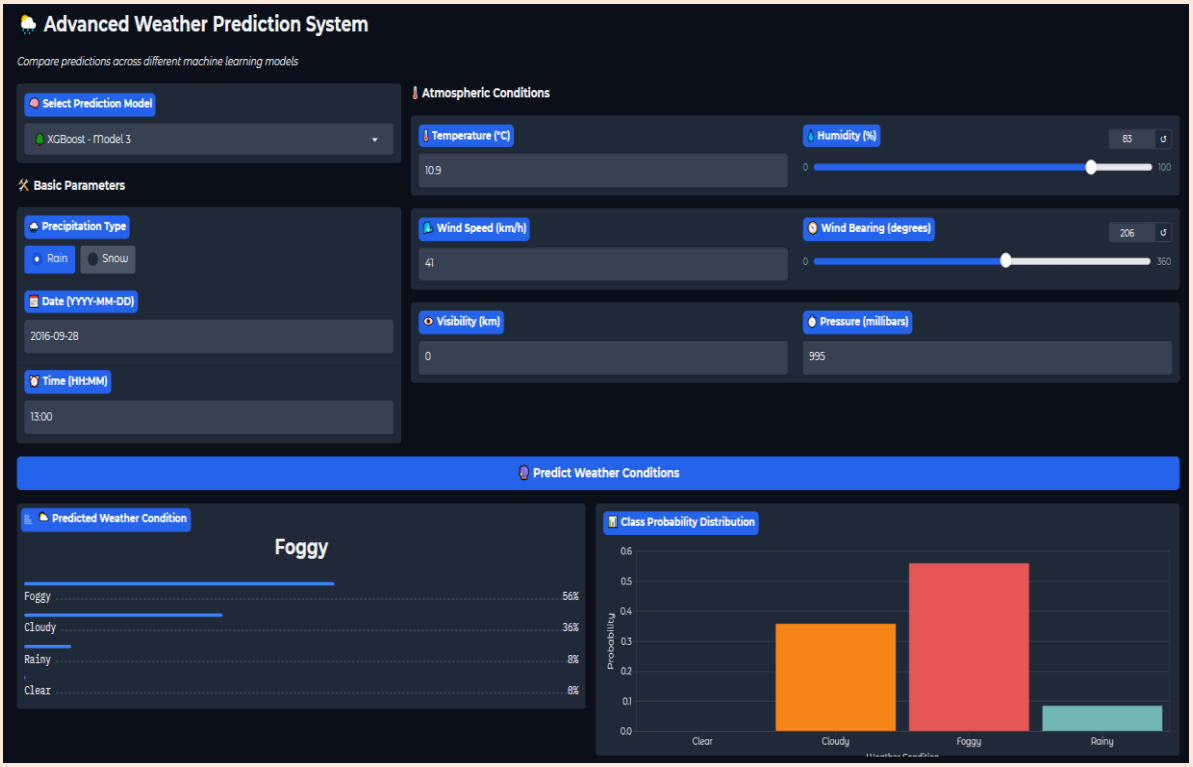
PREDICTION OF CLOUDY:



PREDICTION OF CLEAR:



PREDICTION OF FOGGY:



PREDICTION OF RAINY: