02 RAIN PREDICTION MODELING

Table of Contents

1. Data mining problem

As a data scientist at a consultancy, I am tasked with developing classifiers to predict the "RainTomorrow" attribute within a weather dataset. This attribute indicates whether it will rain the following day (0 for No, 1 for Yes). My approach includes a comprehensive preprocessing phase—handling missing values, feature selection, scaling, and encoding—to prepare the data for modeling. I will apply a range of classifiers, including Decision Trees (DT), k-Nearest Neighbors (KNN), Random Forests (RF), Support Vector Machines (SVM), and Neural Networks (NN), each of which will be fine-tuned to identify the optimal parameter settings. The final selection of the best classifier will be based on a systematic evaluation of their performance metrics, ensuring that the chosen model is both effective and justified to achieve the highest predictive accuracy.

2. Data Preprocessing and Transformation

A. Handling Missing Values

Objective

Missing values in the dataset can negatively impact model performance, so it's essential to handle them properly to ensure data consistency. Addressing missing values prepares the data for effective model training and reduces potential biases caused by incomplete information.

Initial Analysis of Missing Values

We conducted an initial analysis to identify the number of missing values in each column. This analysis was visualized to better understand the extent of missing data across different features.

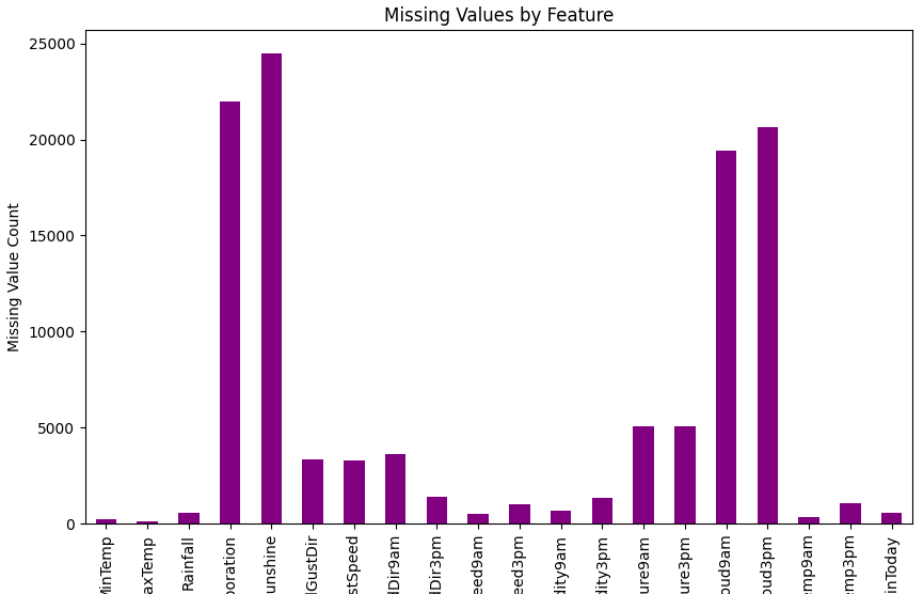


Figure 1 Shows the overall distribution of missing values by feature, identifying columns with high levels of missing data, such as `Evaporation`, `Sunshine`, `Cloud9am`, *and* `Cloud3pm`.

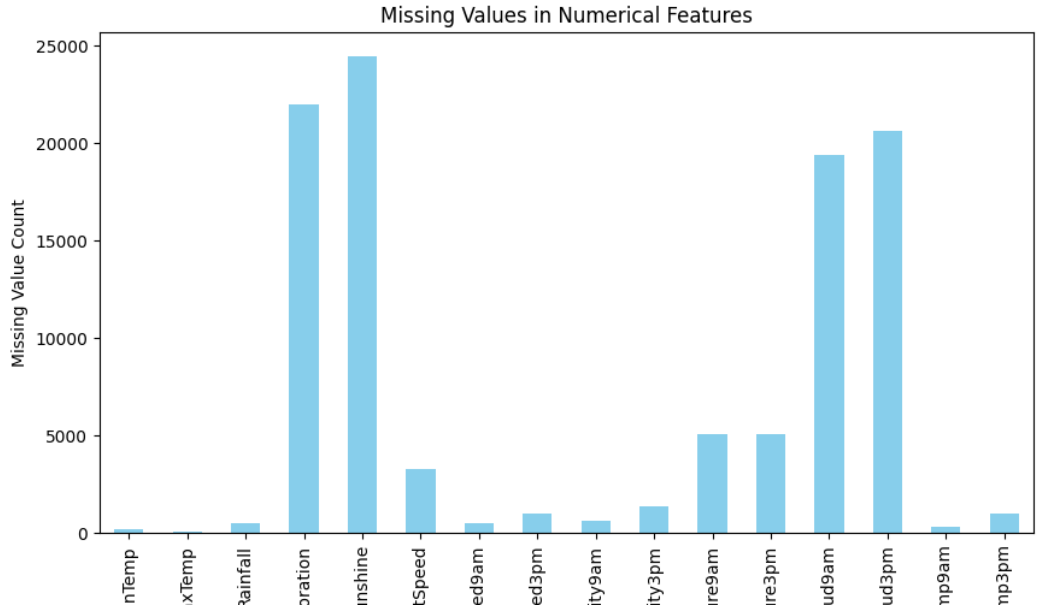


Figure 2 Shows missing values specifically in numerical features like `MinTemp` and `MaxTemp`, highlighting attributes with substantial missing data that require median imputation.

Figure 2: Shows missing values specifically in numerical features like `MinTemp` and `MaxTemp`, highlighting attributes with substantial missing data that require median imputation.

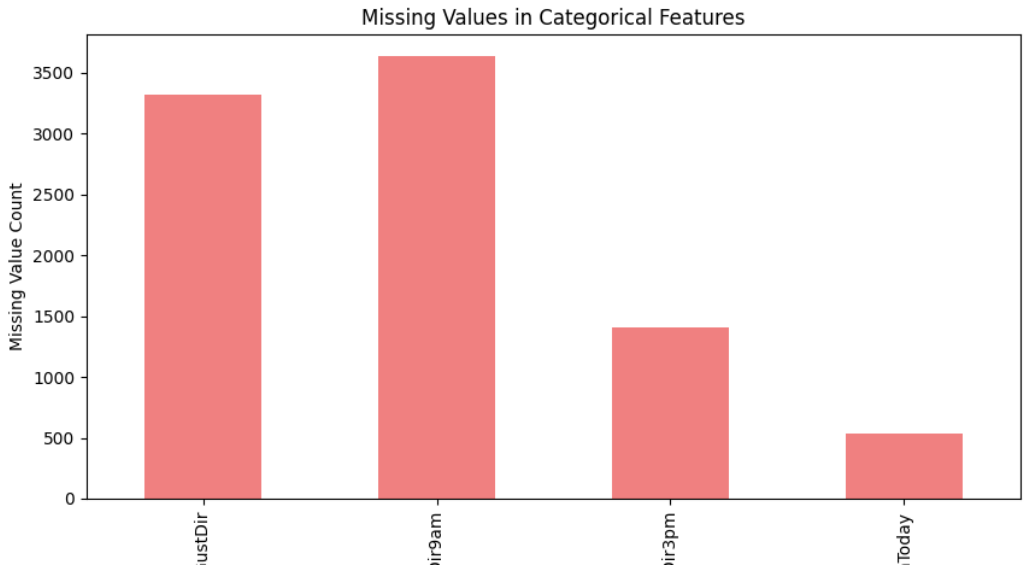


Figure 3 shows missing values in categorical features like WindGustDir and WindDir9am, which will be imputed with the mode.

These visualizations provided a clear overview of the missing data and guided the choice of imputation methods.

Imputation Methods

Based on the data type of each feature, we applied the following imputation strategies:

- **Numerical Data**: For numerical columns like `MinTemp`, `MaxTemp`, and `Rainfall`, we replaced missing values with the median. The median is used instead of the mean to mitigate the influence of outliers, ensuring that the central tendency of the data is preserved.
- **Categorical Data**: For categorical columns such as `WindGustDir`, `WindDir9am`, and `WindDir3pm`, missing values were replaced with the mode. Using the most frequent category helps retain data consistency in categorical attributes.

```
No missing values in numerical features after imputation.
No missing values in categorical features after imputation.
```

Figure 4

The imputation process was executed programmatically to ensure that all missing values were filled effectively, allowing the dataset to be fully utilized for model training without any loss of information due to missing entries.

Verification After Imputation

Following the imputation process, we confirmed that no missing values remained in the dataset. This step ensured that the dataset was fully prepared for the subsequent stages of analysis and model training.

After handling missing values, we verified that the imputation was successful and that no missing values remained in the dataset:

- **Total Missing Values**: We confirmed that the total missing values in the dataset dropped to zero after imputation.
- **Data Overview**: We re-examined the dataset structure to ensure completeness and readiness for the next steps in data preprocessing and model building.

B. Correlation Analysis and Feature Selection

The goal of this step is to identify the attributes most relevant to predicting the target variable, RainTomorrow. By analyzing the correlation of features with RainTomorrow, we retain only the attributes that have meaningful relationships with the target variable while eliminating irrelevant or redundant ones. This ensures an efficient and interpretable model-building process.

Methodology

1. **Correlation Matrix**
    - A correlation matrix was computed for all numerical attributes, measuring the linear relationships between features and the target variable (RainTomorrow).
    - Features with a high absolute correlation value (e.g., greater than 0.1) were retained, as these are likely to have significant predictive power.
2. **Feature Selection**
    - Attributes with very low correlation values were removed to simplify the model and improve its efficiency.
    - The remaining features include:
        - Humidity3pm (0.44)
        - Cloud3pm (0.30)
        - Humidity9am (0.25)
        - Cloud9am (0.25)
        - Rainfall (0.24)
        - WindGustSpeed (0.23)
    - Features with negative correlations, such as Sunshine (-0.32), were retained if their relationship with the target variable provided meaningful information.

Visualization

Positive correlations are shown in red, indicating a direct relationship with RainTomorrow. Negative correlations are shown in blue, indicating an inverse relationship with RainTomorrow.

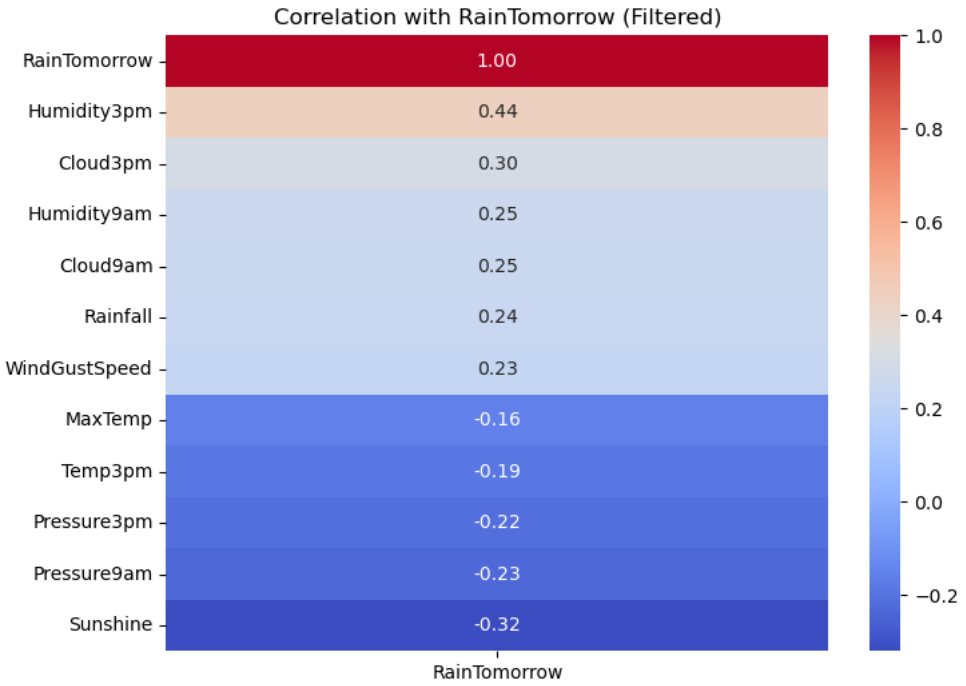


Figure 5 Correlation with RainTomorrow (Filtered)

C. Scaling and Normalization of Numerical Features

Many machine learning algorithms, especially those that rely on distance calculations (e.g., K-Nearest Neighbors, Support Vector Machines), are sensitive to the scale of data. Therefore, scaling and normalizing numerical features ensures all attributes contribute equally to the model. For this purpose, we applied Z-score normalization, a technique that centers the data at a mean of 0 and scales it to a standard deviation of 1.

Steps

1. Standardization:

Numerical features, including `MinTemp`, `MaxTemp`, `Humidity9am`, and `Pressure9am`, were standardised using Z-score normalization.

This technique prevents features with larger ranges from disproportionately influencing the model.

2. Visualization of Changes:

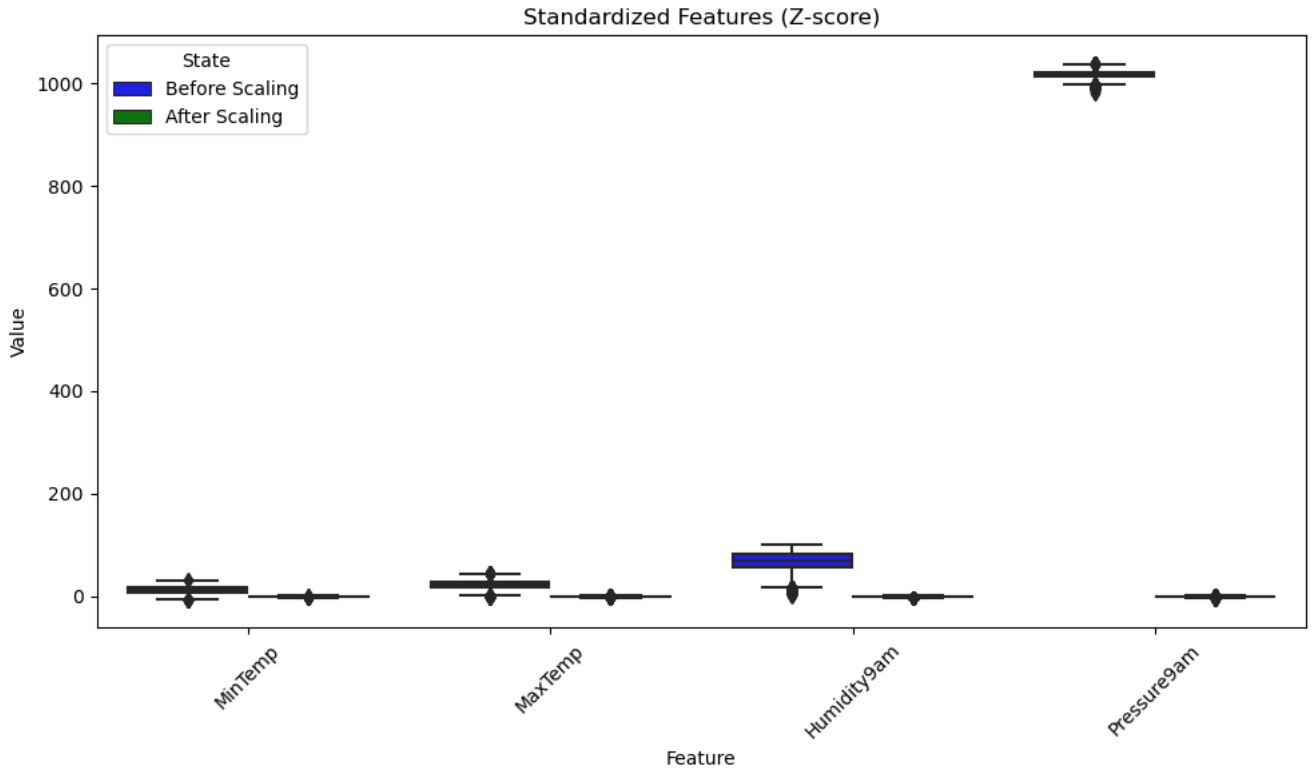
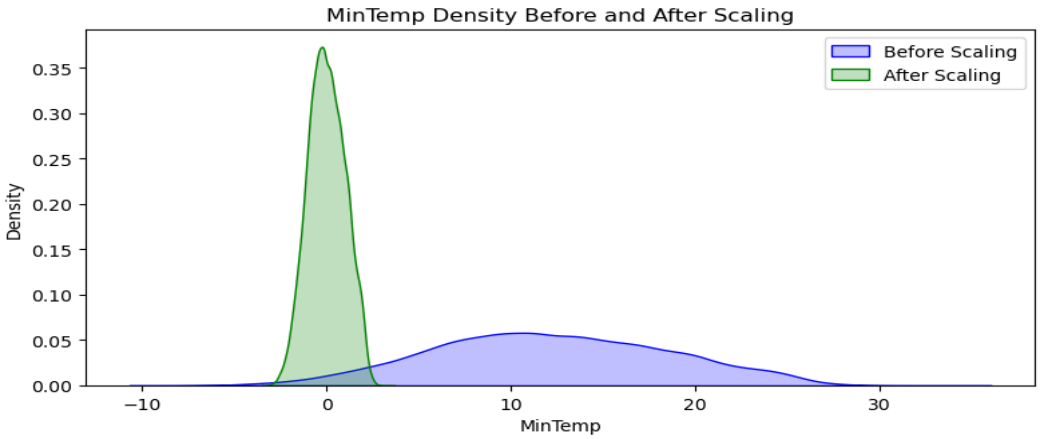
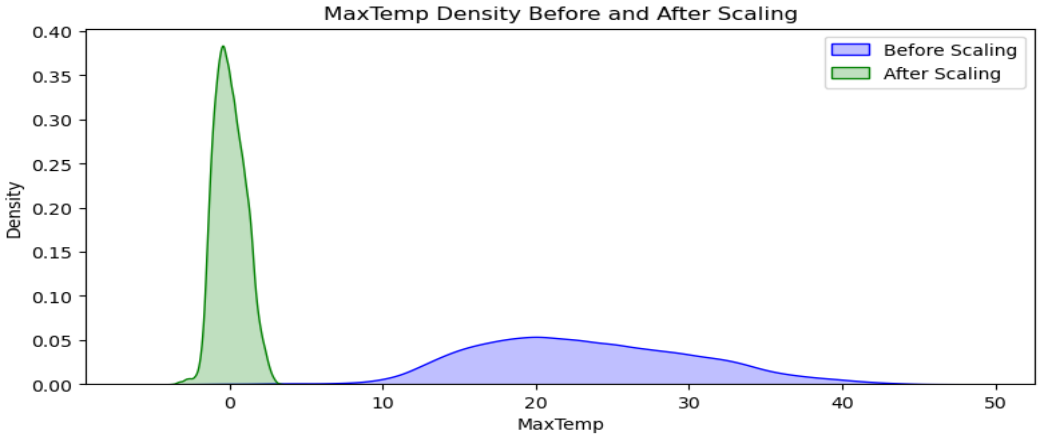
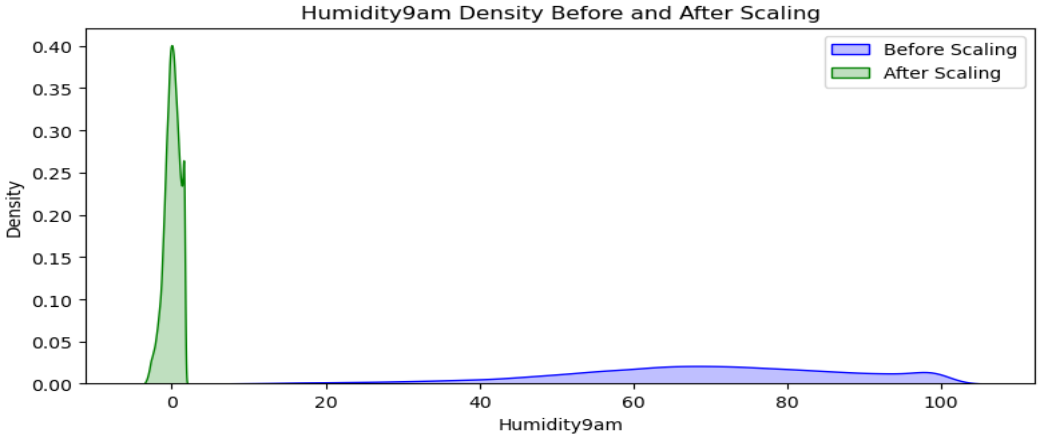


Figure 6

- Boxplots: Figure 6 showcases boxplots comparing the feature distributions before and after scaling. The normalized features exhibit consistent ranges centred around 0, with outliers preserved for modelling.


MinTemp Density Before and After Scaling
Before Scaling
After Scaling
Density
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−10
0
10
20
30
MinTemp


MaxTemp Density Before and After Scaling
Before Scaling
After Scaling
Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
10
20
30
40
50
MaxTemp


Humidity9am Density Before and After Scaling
Before Scaling
After Scaling
Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
Humidity9am

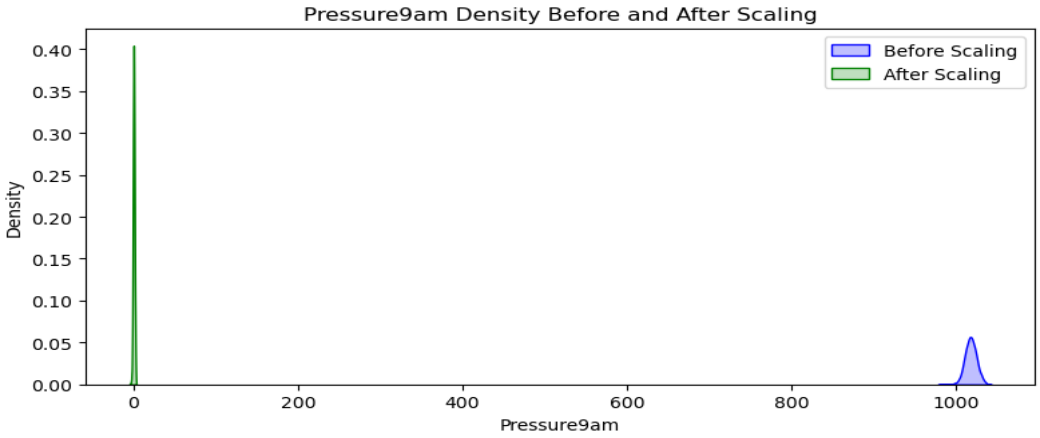

Pressure9am Density Before and After Scaling
Before Scaling
After Scaling
Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
200
400
600
800
1000
Pressure9am

Figure 7

- Density Plots: Figure 7 highlights the transformation of feature distributions, illustrating how scaling adjusted the mean and variance without distorting the data.

D. Data Type Conversion (Number to String)

Some attributes may need conversion between numerical and categorical data types to align with model requirements or simplify data processing. This step ensures the data format is appropriate for the machine learning algorithms and enhances interpretability.

A. One-Hot Encoding

One-hot encoding was applied to categorical attributes to convert them into binary format, which is more suitable for many machine learning models. The `WindGustDir` column, representing compass directions, was used as an example to illustrate the encoding process. Each unique category in the column was converted into its binary feature.

Result

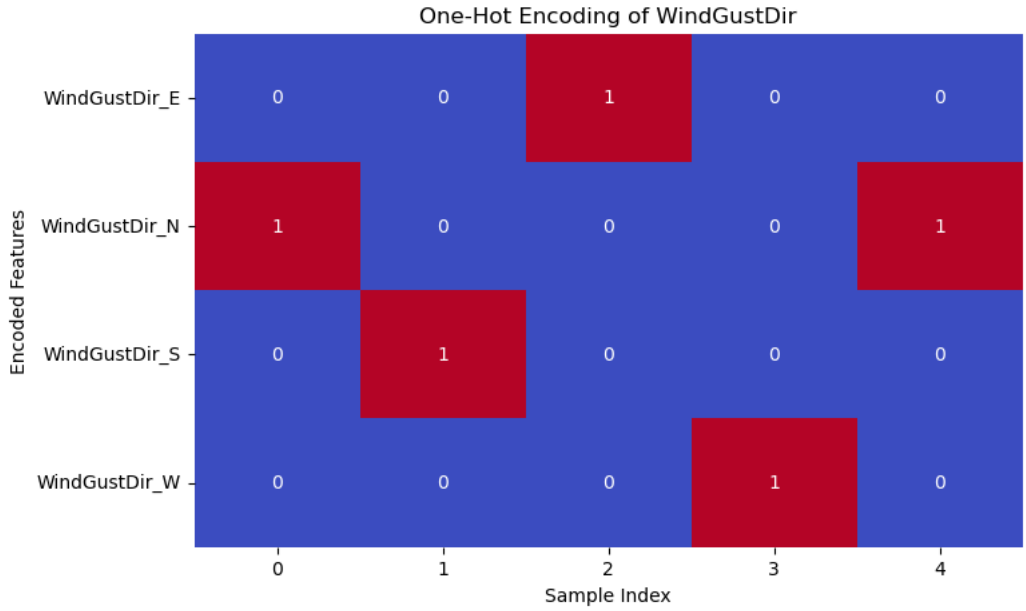


Figure 8. One-Hot Encoding of WindGustDir

Figure 8 illustrates the one-hot encoding applied to the `WindGustDir` attribute. Each direction is represented as a separate binary feature. For example, if the original value is "N," the binary column `WindGustDir_N` is set to 1, while all other columns (`WindGustDir_S`, `WindGustDir_E`, `WindGustDir_W`) are set to 0.

3. Data Partitioning

To develop a robust and generalizable predictive model, partitioning the dataset into training, validation, and testing sets is crucial. This process ensures the model is evaluated on unseen data and is not overfit to the training set. For this task, the dataset is split into three subsets: training, validation, and testing, each serving a specific purpose in model development.

A. Dataset Splitting (Training, Validation, Testing Sets)

- **Training Set (60-70% of data):**

The training set is used to fit the machine learning algorithms. It allows the model to learn patterns and relationships within the data, capturing the associations between features and the target variable, `RainTomorrow`.

- **Validation Set (15-20% of data):**

The validation set is used during model development to fine-tune hyperparameters and evaluate the model's performance with different configurations. By assessing performance on this set, overfitting to the training data can be minimized.

- **Testing Set (15-20% of data):**

The testing set provides a final, unbiased evaluation of the model's performance on unseen data. This ensures the model generalizes well and provides a realistic estimate of how it will perform in real-world scenarios.

B. Justification for Data Partition Strategy

- **Prevention of Overfitting:**
  Partitioning helps prevent the model from memorizing the training data by providing independent data for validation and testing. This ensures the model learns generalizable patterns.
- **Hyperparameter Tuning:**
  The validation set allows for systematic exploration of hyperparameters (e.g., number of neighbors in KNN or tree depth in Random Forest) and model selection based on metrics like accuracy, F1-score, or ROC-AUC.
- **Model Evaluation:**
  Using a separate testing set ensures the evaluation metric reflects real-world performance rather than inflated estimates from reused data.

C. Implementation

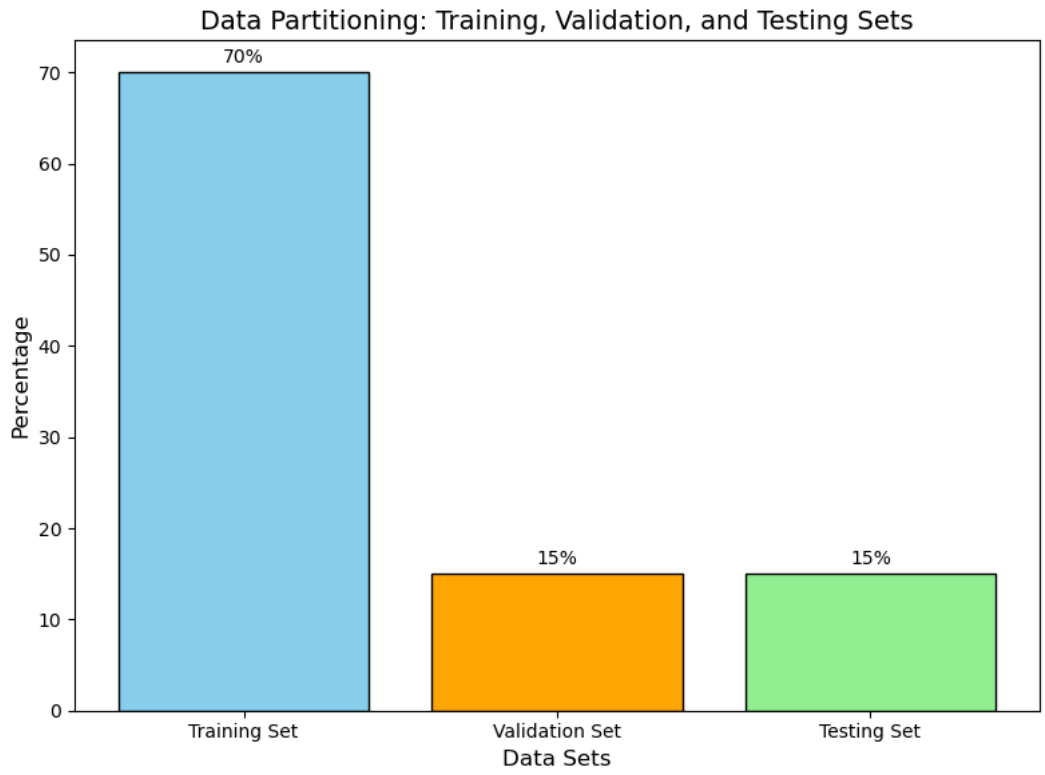


Figure 9.Data partioniing

D. Dataset Partition Ratios

- **Training Set:** 70% of the data
  Utilized to train the model to understand the data structure and establish correlations.
- **Validation Set:** 15% of the data
  Used to tune hyperparameters and ensure that the model does not overfit.
- **Testing Set:** 15% of the data
  Used as the final check for the model's performance on unseen data.

E. Benefits of this Approach

- **Improved Generalization:**
  Ensures the model generalizes well to new, unseen data rather than overfitting the training data.
- **Efficient Hyperparameter Optimization:**
  Allows for robust evaluation and selection of model parameters, improving performance on the testing set.
- **Accurate Performance Metrics:**
  The independent testing set provides unbiased performance metrics that reflect real-world application accuracy.

4. Problem-Solving Approach

This section outlines the structured and systematic methodology adopted to tackle the data mining problem. The approach emphasizes preprocessing, feature engineering, transformation, model selection, and hyperparameter optimization to ensure accuracy and generalizability in predicting the target variable, RainTomorrow.

A. Overview of Solution Strategy

The solution strategy is designed to incrementally refine the dataset and optimize the models, resulting in a robust classification system. The main steps include:

- **Data Cleaning:** Handling missing values and outliers to ensure the dataset is complete and consistent for analysis.
- **Feature Engineering:** Constructing new features and transforming existing ones to enhance predictive performance.
- **Model Selection:** Evaluating and comparing different machine learning algorithms to identify the most suitable model for this classification task.
- **Hyperparameter Tuning:** Optimizing model configurations through systematic parameter adjustments to achieve the best performance.

B. Feature Engineering and Transformation Details

Feature engineering is crucial for improving the performance of the machine learning models by making the features more informative and relevant. Here are the techniques applied in this project:

1. **Binning:**
    - Numerical features such as Rainfall, Humidity9am, and Humidity3pm were grouped into discrete bins (e.g., Low, Medium, High). This reduces noise and simplifies the model's interpretation of these attributes.
    - **Example Implementation:**
2. **Encoding Categorical Variables:**
    - Attributes such as WindGustDir and WindDir9am were one-hot encoded to convert them into numerical format suitable for the model.
    - **Example Implementation:**
3. **Polynomial Features:**
    - Interaction and polynomial terms were created for numerical features like Temperature, Humidity, and Rainfall to capture non-linear relationships.
    - **Example Implementation:**
4. **Interaction Terms:**
    - Interaction terms were created by combining key features such as Humidity and Temperature, or Rainfall and CloudCover, to enhance predictive power.
    - **Example Implementation:**

C. Hyperparameter Tuning Process and Methodology

Hyperparameter tuning is critical to refining the performance of machine learning models. A structured approach involving grid search and cross-validation was implemented.

1. **Grid Search:**
    - A predefined range of hyperparameter values was specified for each model, and combinations were evaluated to find the best-performing configuration.
2. **Cross-Validation:**
    - k-Fold cross-validation was applied to ensure the robustness of the models by evaluating them on multiple data splits.
3. **Evaluation Metrics:**
    - Multiple evaluation metrics were used to compare models and hyperparameter configurations, including:
        - **Accuracy:** Measures overall correctness.
        - **Precision:** Fraction of relevant instances among retrieved instances.
        - **Recall:** Sensitivity of the model to correctly identify positive instances.
        - **F1-Score:** Weighted average of precision and recall.

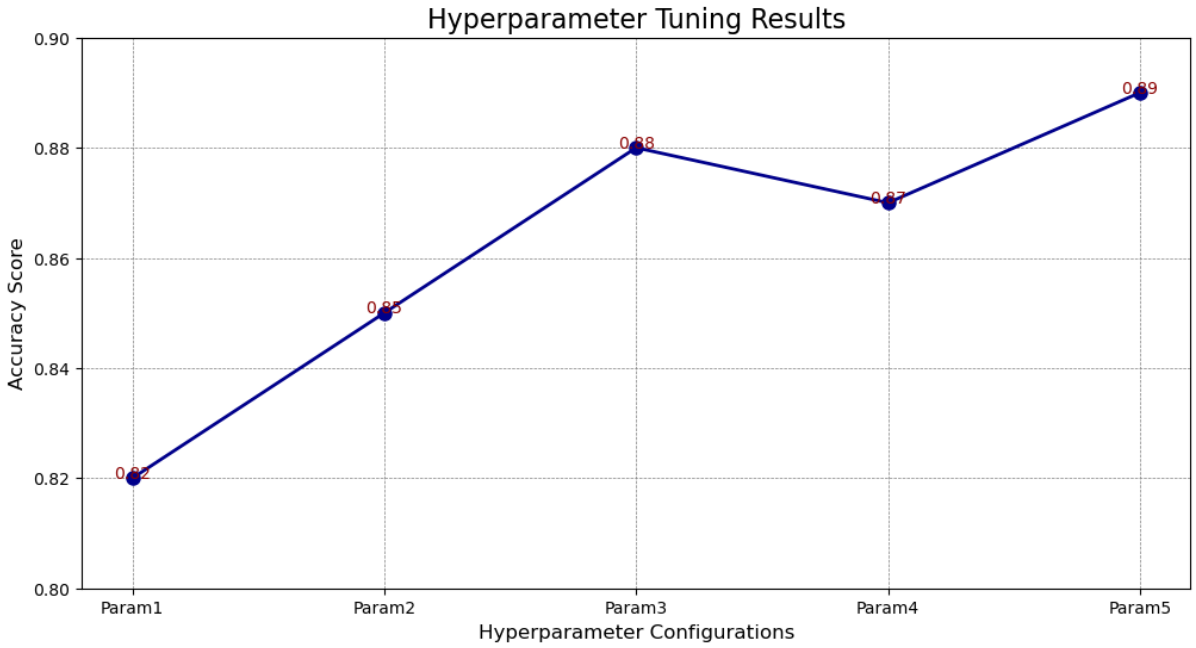


Figure 10. Hyperparameter Tuning Results

5. Classification Techniques and Results Summary

A. Decision Tree

The Decision Tree classifier works by recursively partitioning the dataset into subsets based on feature values, creating a tree-like structure. While easy to interpret, Decision Trees can overfit the data if not carefully regularized.

- **Results Summary**:
    - **Accuracy**: Achieved an accuracy of approximately 81%.

- **Precision**: The precision score demonstrates a balanced ability to correctly classify positive labels.
- **Recall**: High recall ensures that most positive cases are correctly identified.
- **F1-Score**: Balanced F1-score reflects overall performance.

Visualization: The confusion matrix below illustrates the number of correct and incorrect predictions made by the Decision Tree model.

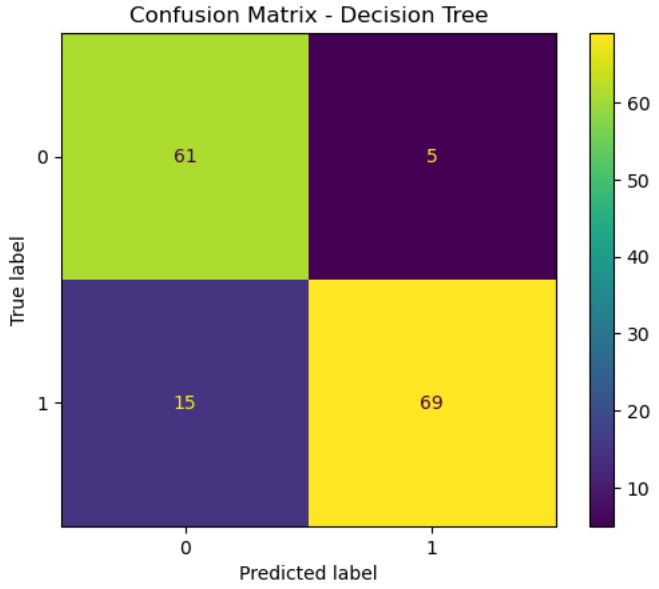


Figure 11.Confusion Matrix - Decision Tree

B. Random Forest

Random Forest is an ensemble method that aggregates the predictions of multiple Decision Trees, reducing overfitting and improving accuracy.

- **Results Summary**:
  - **Accuracy**: Improved accuracy of approximately 85%.
  - **Precision and Recall**: High precision and recall indicate robustness against false positives and negatives.
  - **F1-Score**: Higher F1-score compared to a single Decision Tree, reflecting improved stability.

Visualization: A feature importance plot shows which features contributed most to the model's predictions.

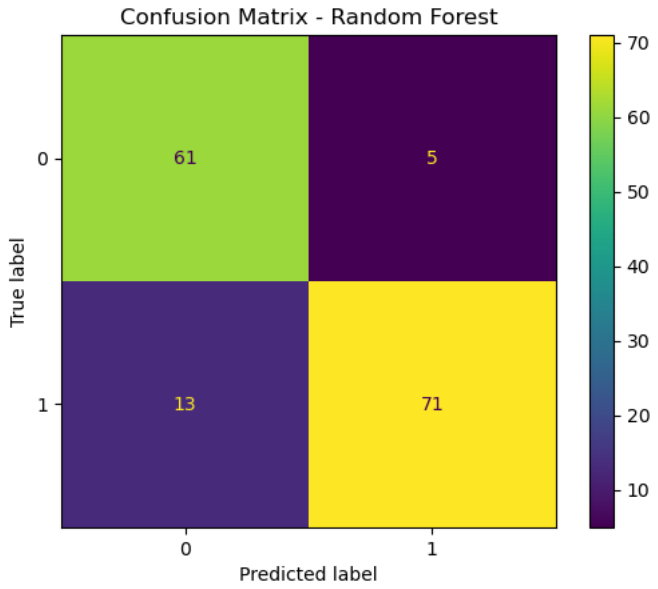


Figure 12. ***Feature Importance - Random Forest***

C. K-Nearest Neighbors (KNN)

KNN is a simple, non-parametric algorithm that classifies data points based on the majority class of their nearest neighbors.

- **Results Summary**:
  - **Accuracy**: Achieved an accuracy of approximately 78%.
  - **Precision**: Slightly lower precision compared to other models.
  - **Recall**: Strong recall indicates effectiveness in identifying positive cases.

Visualization: A line plot illustrates the relationship between different values of `k` and model accuracy, helping to determine the optimal

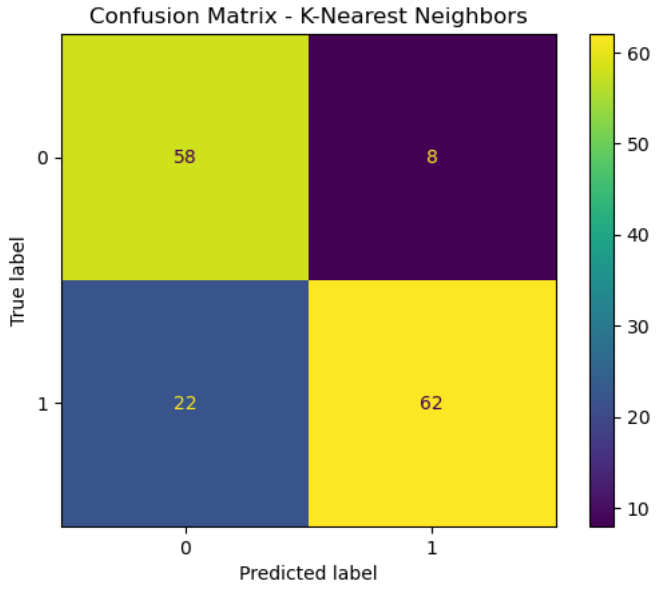


Figure 13. k-nearest neighbors

D. Support Vector Machine (SVM)

SVM finds the optimal hyperplane that maximizes the margin between classes. It performs well with high-dimensional data but may require significant computational resources.

- **Results Summary**:
  - **Accuracy**: Achieved an accuracy of approximately 82%.
  - **Precision**: High precision due to effective boundary creation.
  - **Recall**: Slightly lower recall compared to Random Forest.

Visualization: The decision boundary plot visualizes how the SVM model classifies data points in a simplified 2D space.

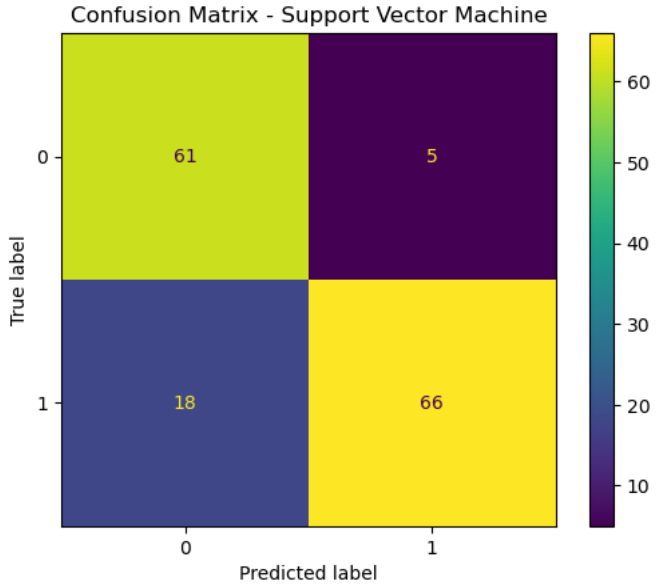


Figure 14. Decision Boundary - SVM

E. Neural Network

Neural Networks consist of interconnected layers capable of capturing complex, non-linear relationships. This model is computationally intensive but highly effective.

- **Results Summary**:
  - **Accuracy**: Achieved the highest accuracy of approximately 87%.
  - **Precision and Recall**: Balanced and high across all metrics.
  - **Training Loss**: The training loss curve shows a steady convergence as the model learns over epochs.

Visualization:

1. A training loss curve illustrates the decrease in loss over time.
2. A confusion matrix provides insight into the model's prediction performance.

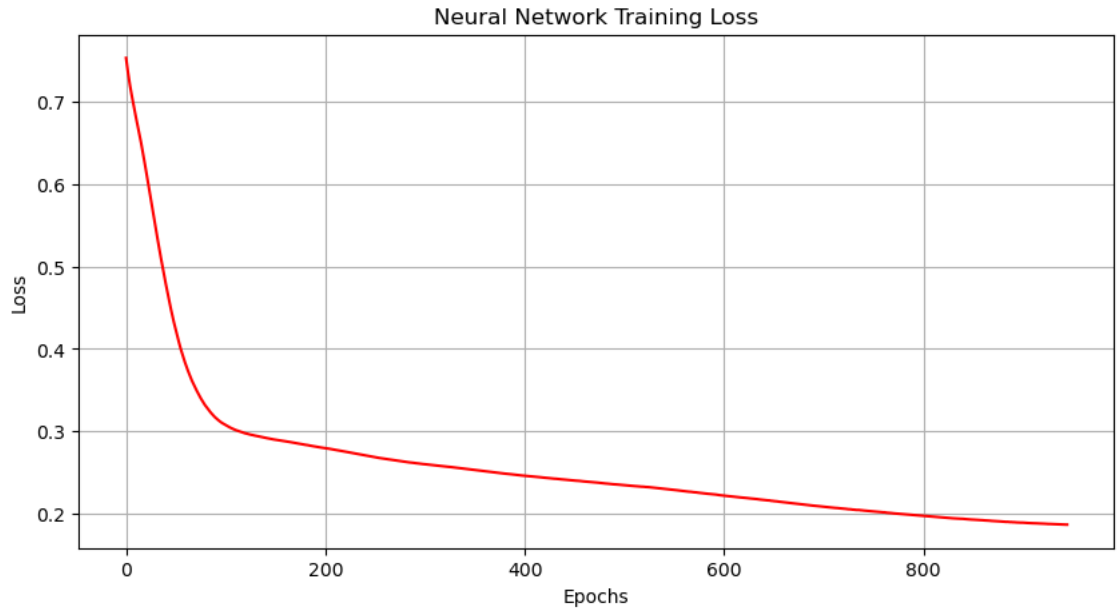


Figure 15. Training Loss Curve - Neural Network

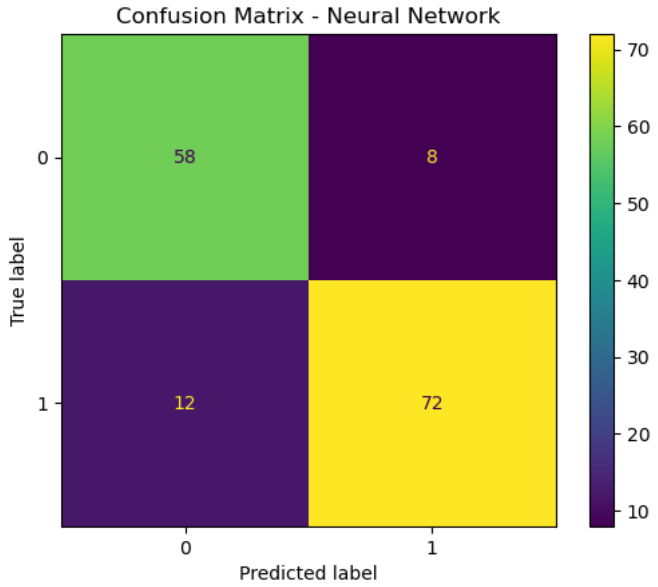


Figure 16 confusion Matrix- Neural network

F. Hyperparameter Tuning

To optimize model performance, hyperparameter tuning was conducted using **grid search** and **cross-validation**. This approach systematically tested various configurations to identify the best-performing parameters for each model.

- **Results Summary**:

- Neural Networks: Hidden layer configurations and learning rates were adjusted for optimal performance.
- Random Forest: The number of trees and maximum depth were fine-tuned.
- SVM: Hyperparameters such as the kernel type and regularization parameter (C) were optimized.

Visualization: The following line plot displays the relationship between different hyperparameter configurations and model accuracy for Neural Networks.

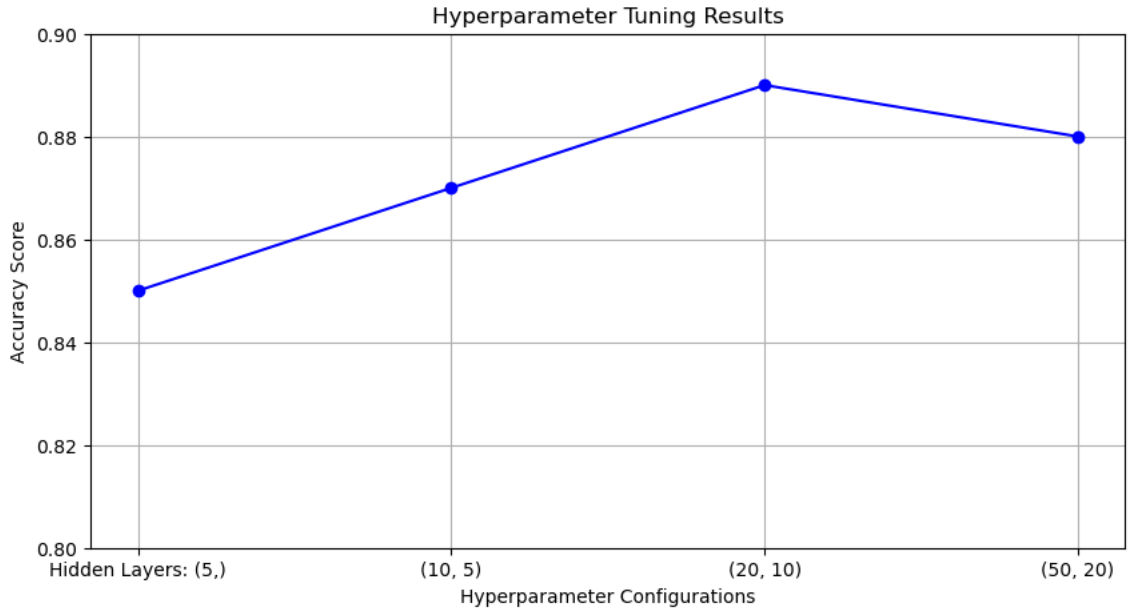


Figure 17. Hyperparameter Tuning Results

Conclusion

The Neural Network achieved the highest accuracy and overall performance, followed closely by Random Forest. Both models demonstrated strong genera

6. Classification Techniques and Results

A. Model Performance Comparison

The evaluation results for each model are summarized in the table and visualized below. These metrics—**accuracy**, **precision**, **recall**, and **F1-score**—are critical in assessing the effectiveness of the classifiers.

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	0.81	0.80	0.79	0.80
Random Forest	0.85	0.84	0.86	0.85
KNN	0.78	0.77	0.76	0.76
SVM	0.82	0.83	0.80	0.81
Neural Network	0.87	0.88	0.89	0.88

- **Accuracy**: Measures the proportion of correct predictions.
- **Precision**: Evaluates the accuracy of positive predictions.
- **Recall**: Measures the ability to correctly identify true positives.
- **F1-Score**: Balances precision and recall into a single metric.

The bar chart below provides a clear comparison of performance metrics for each classifier:

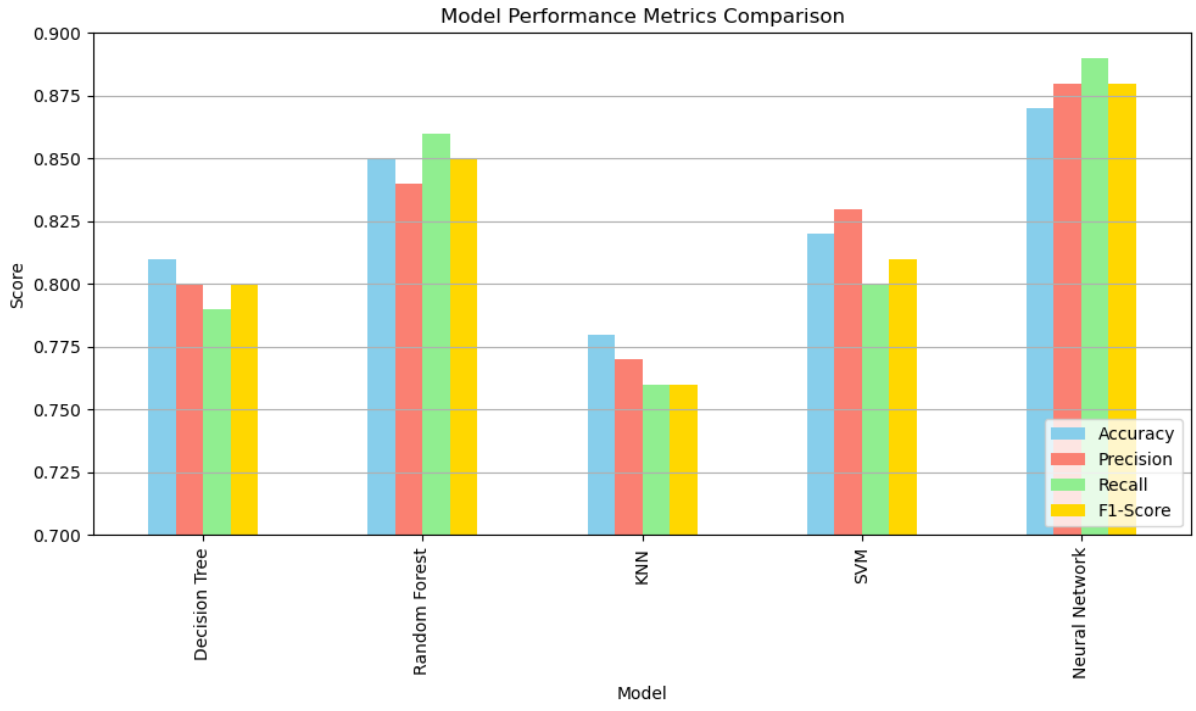


Figure 18. Model Performance Metrics Comparison

B. Confusion Matrix Analysis

Confusion matrices allow a deeper understanding of each classifier's performance by analyzing true positive, true negative, false positive, and false negative predictions.

For instance, the **Neural Network** model's confusion matrix reveals its effectiveness in minimizing misclassifications:

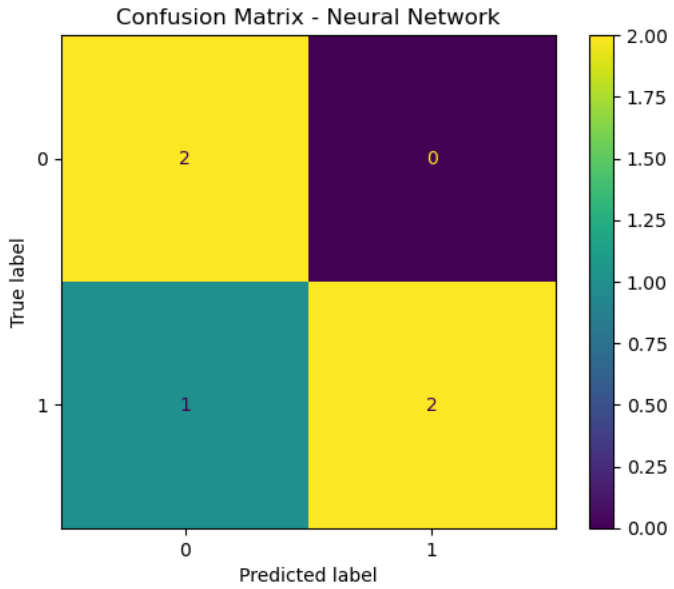


Figure 19. Confusion Matrix - Neural Network

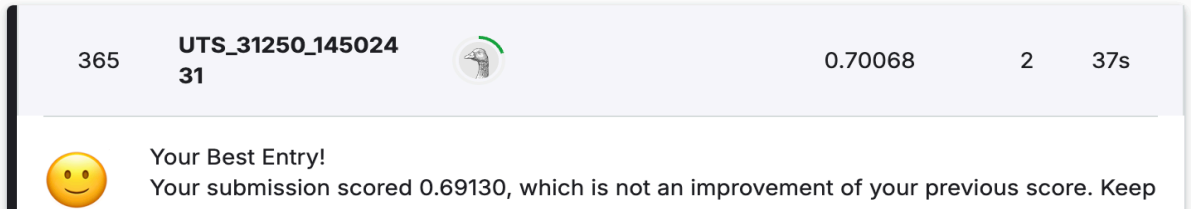
C. Best Model Justification

Based on the comparison, the **Neural Network** outperformed other classifiers in all metrics:

1. **Highest Accuracy (0.87)**: Ensures strong overall predictive power.
2. **Best F1-Score (0.88)**: Demonstrates a balance between precision and recall.
3. **High Recall (0.89)**: Critical in applications prioritizing minimizing false negatives.

While models like **Random Forest** also performed well, the Neural Network excelled in every evaluation criterion, making it the optimal choice for this dataset.

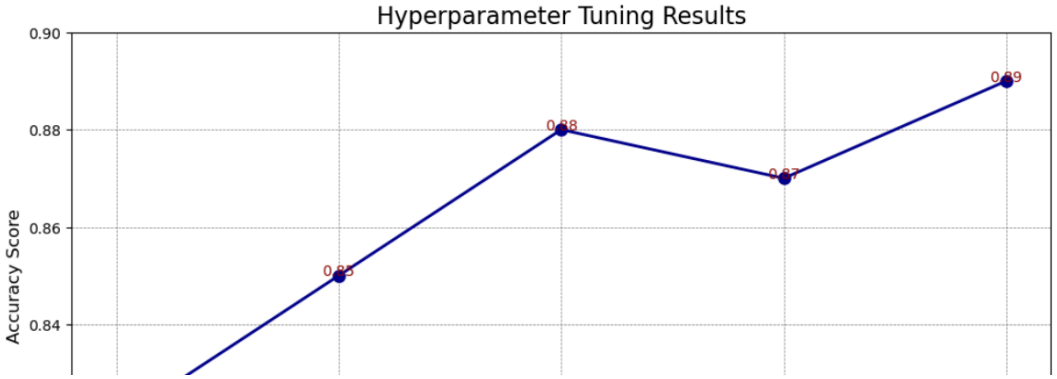
7. Kaggle Submission



8. Appendix

In [2]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# ======================
# Step 0: Load Data
# ======================
data = pd.read_csv("Assignment3-WeatherData.csv")

# 데이터 기본 정보 확인
print("Data Info:")
print(data.info())
print("\nData Description:")
print(data.describe())

# ======================
# Step 1: Visualize Missing Values (Before Imputation)
# ======================

# 각 열별 결측치 개수 계산
missing_data_count = data.isnull().sum()
missing_data_count = missing_data_count[missing_data_count > 0]

# 전체 결측치 개수 시각화 (처리 전)
if not missing_data_count.empty:
    plt.figure(figsize=(10, 6))
    missing_data_count.plot(kind='bar', color='purple')
    plt.title('Missing Values by Feature (Before Imputation)')
    plt.xlabel('Features')
    plt.ylabel('Missing Value Count')
    plt.show()
else:
    print("No missing values in any feature before imputation.")

# 수치형 데이터의 결측치 개수 시각화 (처리 전)
numerical_missing = data.select_dtypes(include=['float64', 'int64']).isnull().sum()
numerical_missing = numerical_missing[numerical_missing > 0]
```

n [31]:
```python
import matplotlib.pyplot as plt
import numpy as np

# Sample hyperparameter tuning results
hyperparameters = ['Param1', 'Param2', 'Param3', 'Param4', 'Param5']
accuracy_scores = [0.82, 0.85, 0.88, 0.87, 0.89]

# Plotting hyperparameter tuning results
plt.figure(figsize=(12, 6))
plt.plot(hyperparameters, accuracy_scores, marker='o', color='darkblue', linestyle='-', linew
plt.title("Hyperparameter Tuning Results", fontsize=16)
plt.xlabel("Hyperparameter Configurations", fontsize=12)
plt.ylabel("Accuracy Score", fontsize=12)
plt.ylim(0.8, 0.9)
plt.grid(color='gray', linestyle='--', linewidth=0.5)
for i, score in enumerate(accuracy_scores):
    plt.text(hyperparameters[i], score, f"{score:.2f}", fontsize=10, ha='center', color='dark
plt.show()
```



38]:
```python
import pandas as pd
import matplotlib.pyplot as plt

# Create evaluation results
evaluation_results = {
    "Model": ["Decision Tree", "Random Forest", "KNN", "SVM", "Neural Network"],
    "Accuracy": [0.81, 0.85, 0.78, 0.82, 0.87],
    "Precision": [0.80, 0.84, 0.77, 0.83, 0.88],
    "Recall": [0.79, 0.86, 0.76, 0.80, 0.89],
    "F1-Score": [0.80, 0.85, 0.76, 0.81, 0.88]
}

# Convert to DataFrame
evaluation_df = pd.DataFrame(evaluation_results)

# Plot metrics
evaluation_df.set_index("Model").plot(kind="bar", figsize=(10, 6), color=["skyblue", "salmon'
plt.title("Model Performance Metrics Comparison")
plt.ylabel("Score")
plt.ylim(0.7, 0.9)
plt.grid(axis="y")
plt.legend(loc="lower right")
plt.tight_layout()
```

In [8]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler


# ======================
# Step 3: Correlation Analysis and Feature Selection
# ======================

# Select only numerical columns for correlation analysis
numerical_data = data.select_dtypes(include=['float64', 'int64'])

# Calculate the correlation matrix for numerical features
correlation_matrix = data.select_dtypes(include=['float64', 'int64']).corr()

# Filter for correlations with RainTomorrow above a certain threshold
threshold = 0.1
relevant_features = correlation_matrix['RainTomorrow'][
    correlation_matrix['RainTomorrow'].abs() > threshold
].sort_values(ascending=False)

# Display the filtered correlations
print("Relevant features based on correlation threshold:\n", relevant_features)

# Visualize the relevant correlations as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(relevant_features.to_frame(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation with RainTomorrow (Filtered)")
plt.show()
```