CSE 454 PROJECT REPORT RAINFALL PREDICTION

JANUARY 26, 2024

ELIFNUR KABALCI
1801042617

Introduction

"Rainfall Prediction: An Analysis on Australian Weather Data" is the title of this project, which aims to develop a machine learning model that predicts the likelihood of rainfall using weather data collected from various regions of Australia. The dataset, named "weatherAus.csv," contains a wide range of meteorological variables, including key weather measurements such as temperature, humidity, wind speed, and direction.

The purpose of our project is to create a model that provides reliable rainfall forecasts using this rich dataset, allowing both individuals and businesses to better plan their daily activities. An effective rainfall prediction model has a broad range of applications, from agriculture to urban planning and disaster management.

Our model development process will begin with exploratory data analysis of the dataset, followed by data preprocessing steps. These will include filling missing values, encoding categorical variables, and normalizing the data. Subsequently, the most appropriate model will be selected using various classification algorithms and evaluated with metrics such as the ROC curve, Confusion Matrix, and Cohen's Kappa.

Finally, hyperparameter tuning and cross-validation techniques will be applied to improve the model's performance and ensure its generalizability. The project will conclude with an evaluation of the model's results and recommendations on how it can be used to predict future rainfall events.

Link of the Trailer:
https://gtu.sharepoint.com/sites/ben776/Shared%20Documents/General/Recordings/Meeting%20in%20_General_-20240125_235812-Meeting%20Recording.mp4?web=1

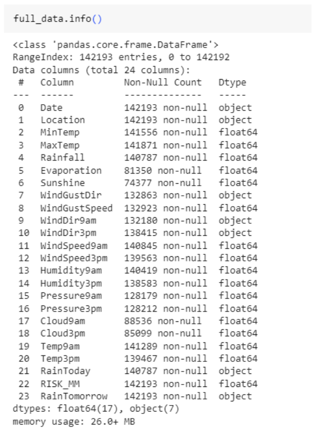
EDA

This image displays the general characteristics of a weather dataset encapsulated in a Pandas DataFrame's `info()` output. The DataFrame contains a series of features related to weather conditions, spanning a total of 142193 entries distributed across 24 columns. Each column represents different meteorological parameters. Looking at the data types, most columns contain `float64` numerical type data, and some columns are of the `object` type. `object` type data usually represents textual information or complex data structures.

Columns such as Date (`Date`), Location (`Location`), and Rain Status (`RainToday` and `RainTomorrow`) are of the `object` type. This suggests that these columns might contain date information, textual location information, and categorical data in the form of yes/no answers. Columns like `MinTemp`, `MaxTemp`, `Rainfall`, `Evaporation`, and `Sunshine` contain numerical values representing features such as minimum temperature, maximum temperature, the amount of rainfall, the amount of evaporation, and duration of sunshine, respectively.

The dataset also includes columns detailing atmospheric conditions at different times of the day, such as wind speed (`WindGustSpeed`, `WindSpeed9am`, `WindSpeed3pm`), wind direction (`WindGustDir`, `WindDir9am`, `WindDir3pm`), humidity (`Humidity9am`, `Humidity3pm`), pressure (`Pressure9am`, `Pressure3pm`), and cloud cover (`Cloud9am`, `Cloud3pm`). These data can be used for instantaneous and historical analyses of the weather conditions in a specific location.

The size and diversity of the dataset can be quite beneficial for developing machine learning models, making weather predictions, analyzing climate change, and other applications in environmental sciences. The memory usage is approximately 26.4+ MB, indicating that it is a moderately sized dataset that can be easily processed with a modern computer. The non-null counts suggest the presence of missing data in some columns, which should be considered during the data preprocessing stages. For instance, missing data in columns like `Sunshine`, `Evaporation`, `Cloud9am`, and `Cloud3pm` might indicate days when these features were not recorded or errors during data collection. These deficiencies are important issues that need to be addressed when working with the dataset, whether by filling in the missing values or excluding them from the analysis.



```
full_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142193 entries, 0 to 142192
Data columns (total 24 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           142193 non-null  object
 1   Location       142193 non-null  object
 2   MinTemp        141556 non-null  float64
 3   MaxTemp        141871 non-null  float64
 4   Rainfall       140787 non-null  float64
 5   Evaporation    81350 non-null   float64
 6   Sunshine       74377 non-null   float64
 7   WindGustDir    132863 non-null  object
 8   WindGustSpeed  132923 non-null  float64
 9   WindDir9am     132180 non-null  object
 10  WindDir3pm     138415 non-null  object
 11  WindSpeed9am   140845 non-null  float64
 12  WindSpeed3pm   139563 non-null  float64
 13  Humidity9am    140419 non-null  float64
 14  Humidity3pm    138583 non-null  float64
 15  Pressure9am    128179 non-null  float64
 16  Pressure3pm    128212 non-null  float64
 17  Cloud9am       88536 non-null   float64
 18  Cloud3pm       85099 non-null   float64
 19  Temp9am        141289 non-null  float64
 20  Temp3pm        139467 non-null  float64
 21  RainToday      140787 non-null  object
 22  RISK_MM        142193 non-null  float64
 23  RainTomorrow   142193 non-null  object
dtypes: float64(17), object(7)
memory usage: 26.0+ MB
```

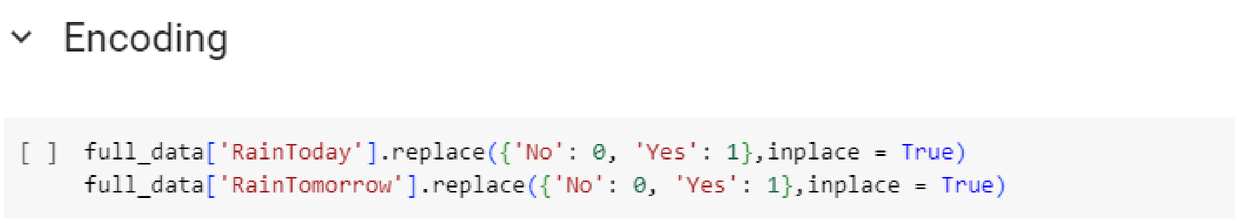
Encoding

In this visual, a data encoding process is demonstrated on two categorical data columns from the weather dataset, namely `RainToday` and `RainTomorrow`. Data encoding is typically used to convert categorical data into numerical values. This is necessary because most machine learning algorithms cannot directly work with textual or categorical data and require numerical inputs to perform better.

In this example, the 'No' and 'Yes' values in the `RainToday` and `RainTomorrow` columns are being replaced with 0 and 1, respectively. This transformation converts the data in these columns into binary formats, representing the presence or absence of these features in a numerical form. Binary encoding is a common method for two-valued categorical data such as yes/no, true/false.

The `inplace=True` argument ensures that the modification is performed directly on the existing DataFrame, meaning the original DataFrame is altered, and no new copy is created. This is important for memory efficiency as it prevents additional memory usage when working with large datasets.

This operation can be seen as a part of preparing the dataset for the modeling phase and is often one of the data preprocessing steps. After this encoding, the `RainToday` and `RainTomorrow` columns can be provided as inputs to machine learning models. These changes are crucial, especially for binary classification tasks like predicting rain.



Encoding

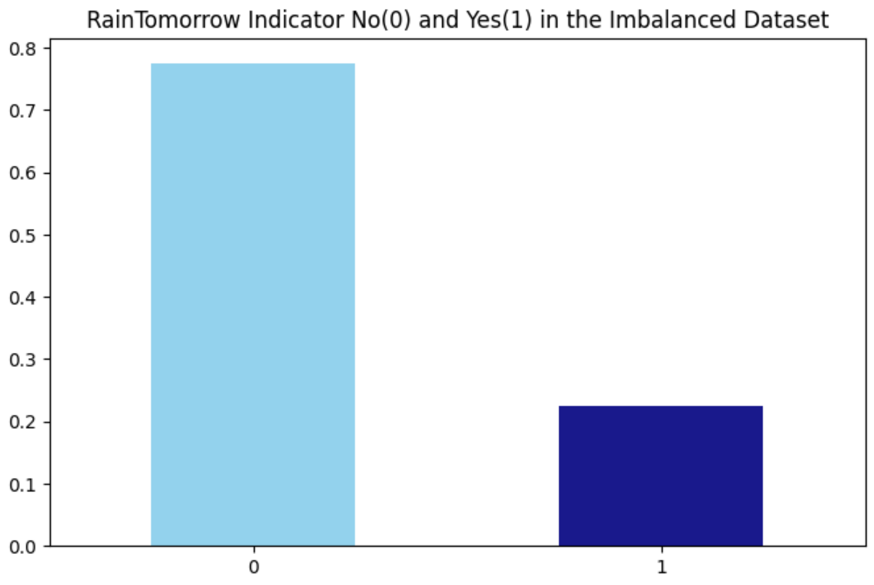
```
[ ]  full_data['RainToday'].replace({'No': 0, 'Yes': 1},inplace = True)
     full_data['RainTomorrow'].replace({'No': 0, 'Yes': 1},inplace = True)
```

Preprocessing – Balance Dataset

The Python code and its corresponding output seen in the visual contain a bar chart displaying the distribution of values in the `RainTomorrow` column of the weather dataset. This chart illustrates the distribution of classes (whether it will rain or not) in the dataset and has been used to analyze whether the dataset is balanced or not.

The bar chart has been created using the normalized value counts of the `RainTomorrow` column (`value_counts(normalize=True)`). The normalization process calculates the ratio of each class to the total number of entries and expresses the results as percentages. In the graph, the 'No' (0) class appears to be more than 70%, while the 'Yes' (1) class is less than 30%. This indicates an imbalance in the dataset, where the number of days indicating no rain significantly outweighs the number of days indicating rain.

Such an imbalance is a significant concern during the training of machine learning models because it can lead the model to favor the majority class and potentially result in misclassification of the minority class. Imbalanced datasets can negatively impact the performance of the model and lead to misleadingly high accuracy rates. Therefore, when working with imbalanced datasets, it is important to implement strategies to address the imbalance, such as resampling techniques or different performance metrics. This graph suggests that the dataset is a candidate for such adjustments, and analysts or data scientists should take this imbalance into consideration during the modeling phase.

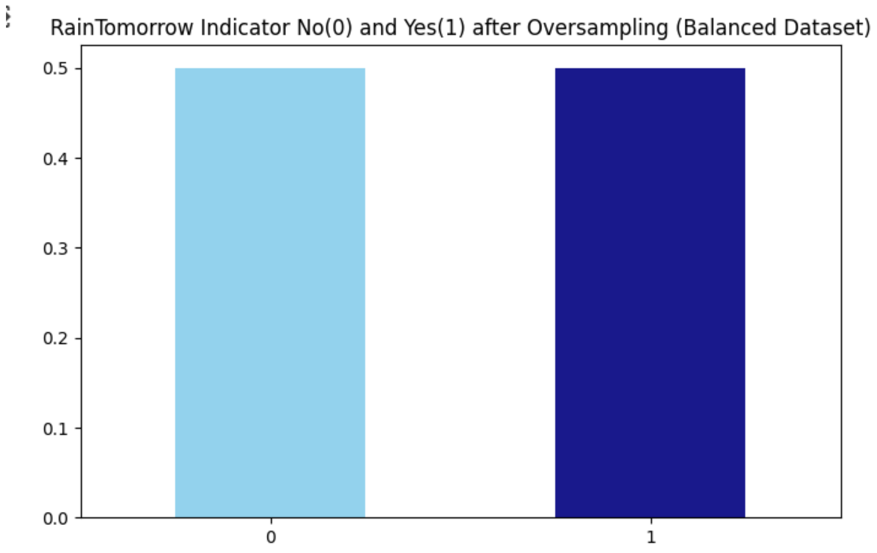


It shown in the visual, along with the associated output, demonstrates the 'oversampling' technique used to address the imbalance issue in a dataset. This technique aims to achieve a balanced distribution between classes by increasing the number of samples of the less represented class (in this case, 'Yes,' representing the occurrence of rain with a value of 1).

The code first uses the `resample` function from the `sklearn.utils` module to increase the number of samples in the 'Yes' class to be equal to the number of samples in the 'No' class. The `replace=True` argument allows the same samples to be selected multiple times, thereby increasing the number of observations in the minority class. `n_samples=len(no)` specifies that the number of samples in the minority class should be equal to the number of samples in the majority class. The `random_state` argument ensures the process is reproducible.

After the resampling process, a new combined dataset (`oversampled`) containing both 'No' and 'Yes' classes is created. Then, the distribution of values in the `RainTomorrow` column of this balanced dataset is plotted as a bar chart. In the graph, both classes have approximately equal proportions, around 50%, indicating that the oversampling process has successfully created a balanced dataset.

This balanced dataset can enable machine learning models to make fairer and more consistent predictions. The model is more likely to learn from and correctly classify samples from the minority class. However, oversampling has its own challenges and drawbacks. For instance, because oversampling artificially replicates copies of the same samples, it can lead to overfitting of the model. Therefore, when using a balanced dataset, careful validation and testing processes should be applied to assess the model's generalization.

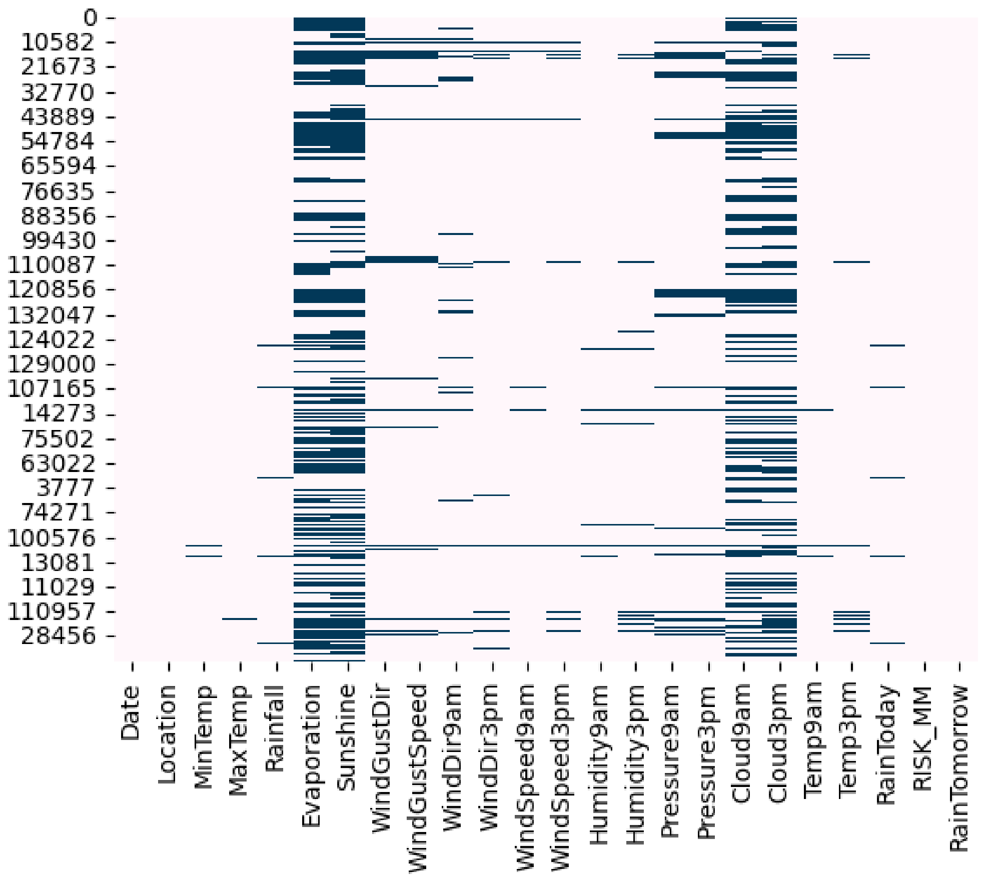


The Python code and their outputs shown in the two visuals demonstrate methods used to analyze missing data patterns in a dataset.

In the first visual, a heatmap is drawn using the Seaborn library. This heatmap displays missing data points (`NaN` values) in each column of the `oversampled` DataFrame. Black areas represent where data is present, while blue lines represent missing data points. The graph reveals significant amounts of missing data in some columns, especially in the `Sunshine`, `Evaporation`, `Cloud3pm`, and `Cloud9am` columns. It provides a quick and visual understanding of which columns have missing data and the distribution of these missing values in the dataset.

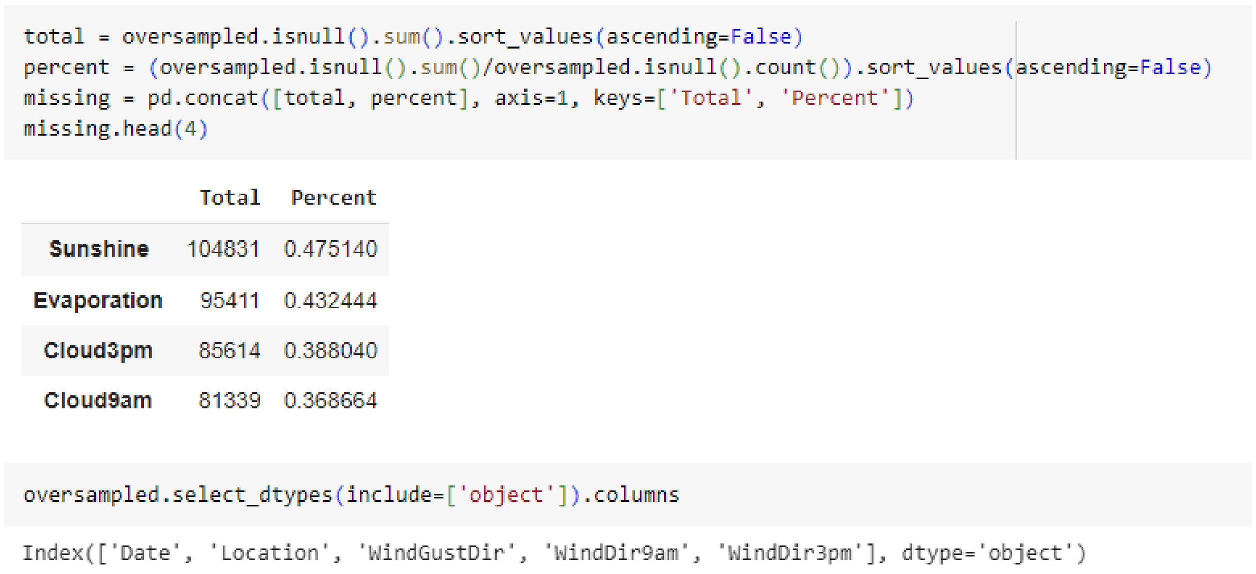
```
# Missing Data Pattern in Training Data
import seaborn as sns
sns.heatmap(oversampled.isnull(), cbar=False, cmap='PuBu')
```

```
<Axes: >
```



The code in the second visual uses functions from the Pandas library to calculate the number and percentage of missing values. By combining the `isnull()` and `sum()` functions, it calculates the total number of missing values in each column, which is then sorted by columns. The result is a DataFrame containing the counts of missing values and the corresponding percentage rates, obtained by dividing the number of missing values by the total number of entries in the dataset. The first four columns with the highest missing value percentages are displayed in this DataFrame. Additionally, the `select_dtypes(include=['object'])` method is used to list columns with a data type of `object`, which may contain textual or categorical data.

The combination of these two visuals helps data scientists identify missing data problems, determine how seriously they should be taken, and develop strategies to address missing data issues. Missing data can pose a significant challenge in data analysis and machine learning models, and it's crucial to have a good understanding of the structure and extent of missing data before attempting to address it. For example, if missing data is randomly distributed, it may be possible to impute or remove them to clean the dataset. However, if missingness follows a specific pattern or occurs under certain conditions, it may require more complex analyses and handling techniques.



```
total = oversampled.isnull().sum().sort_values(ascending=False)
percent = (oversampled.isnull().sum()/oversampled.isnull().count()).sort_values(ascending=False)
missing = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing.head(4)
```

```
             Total   Percent
Sunshine     104831  0.475140
Evaporation   95411  0.432444
Cloud3pm      85614  0.388040
Cloud9am      81339  0.368664
```

```
oversampled.select_dtypes(include=['object']).columns
```

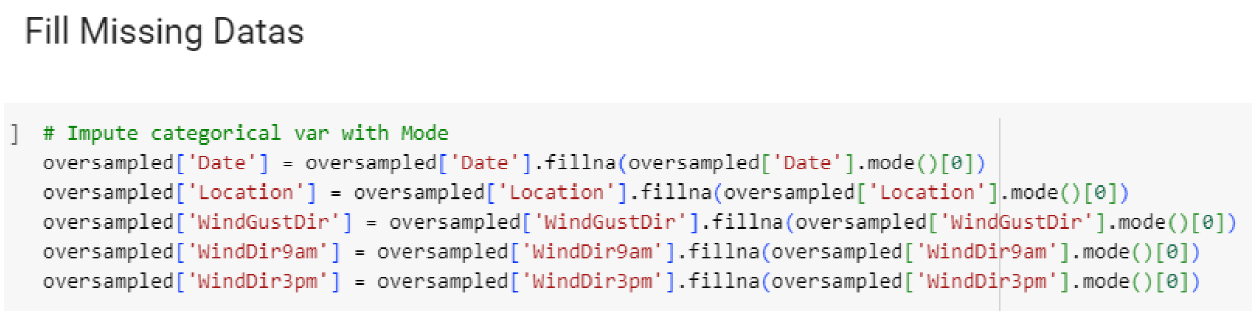
```
Index(['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm'], dtype='object')
```

Fill Missing Data’s

The Python code shown in the visual demonstrates a method used to fill in missing data in the dataset. This method involves filling in missing values for categorical variables using the mode (most frequent value), which is quite common for categorical data. The code uses the `fillna` function to fill in the missing data (NaN values) in the `Date`, `Location`, `WindGustDir`, `WindDir9am`, and `WindDir3pm` columns with the mode of each respective column.

The expression `oversampled['Column'].fillna(oversampled['Column'].mode()[0])` replaces all NaN values in a specific column with the most frequent value in that column. The `mode()[0]` expression is used to retrieve the first value among the values returned by the mode (because mode can return multiple values).

This process is suitable for categorical data where missing values are not random and tend to have a specific value. Particularly in datasets with time-varying data, such as weather data, situations where wind directions may be dominant on certain days or locations, using the mode can be a sensible approach. However, this method may not necessarily reflect the true distribution of the dataset, and there is no guarantee that the mode's high frequency will also be present in the missing data points. Therefore, such imputation processes should be performed carefully, taking into account the structure of the dataset and the analysis objectives.


Fill Missing Datas

```
# Impute categorical var with Mode
oversampled['Date'] = oversampled['Date'].fillna(oversampled['Date'].mode()[0])
oversampled['Location'] = oversampled['Location'].fillna(oversampled['Location'].mode()[0])
oversampled['WindGustDir'] = oversampled['WindGustDir'].fillna(oversampled['WindGustDir'].mode()[0])
oversampled['WindDir9am'] = oversampled['WindDir9am'].fillna(oversampled['WindDir9am'].mode()[0])
oversampled['WindDir3pm'] = oversampled['WindDir3pm'].fillna(oversampled['WindDir3pm'].mode()[0])
```

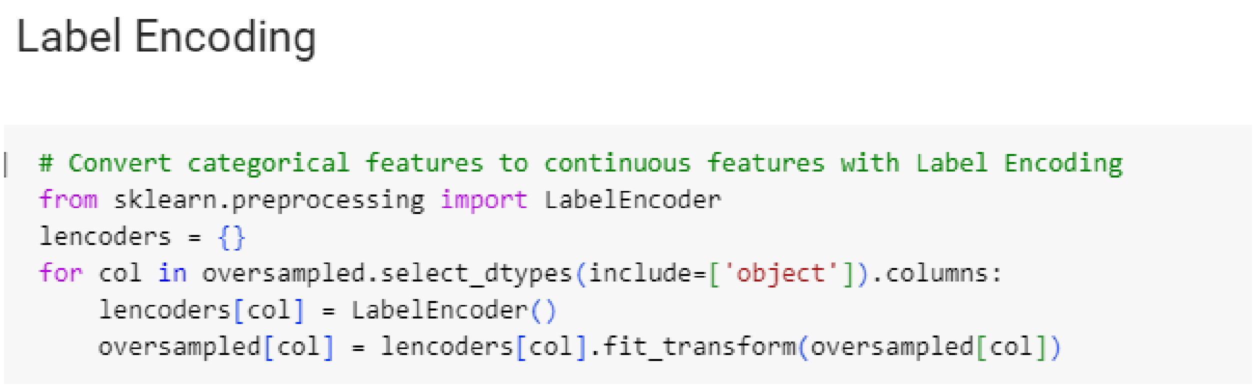
Label Encoding

The Python code in the visual demonstrates a method used for converting categorical data columns into numerical values. This method utilizes the `LabelEncoder` class from the Scikit-learn library. Label encoding transforms each unique categorical value into a numerical label, allowing machine learning algorithms to work with this data.

In the code snippet, the `LabelEncoder` class is first imported. Then, a loop is initiated over the columns in the dataset that have the `object` data type (`select_dtypes(include=['object']).columns`). For each categorical column, a `LabelEncoder` instance is created, and these instances are stored in a dictionary structure (`lencoders`).

Next, the values of each column are transformed into numerical labels using the `.fit_transform()` method. This method performs two operations together: the `.fit()` method learns the unique values in the column and assigns a numerical label to each of them, while the `.transform()` method uses these assignments to convert categorical values into numerical labels.

Label encoding is typically preferred for ordinal categorical data rather than nominal categorical data because it establishes an order or magnitude relationship among the generated numerical labels. For example, 'small,' 'medium,' and 'large' categories can be encoded as 0, 1, and 2, respectively. However, this ranking may not always be desired or meaningful, so other methods like one-hot encoding might be more suitable depending on the use case. This code converts all `object` type columns in the dataset into numerical values, making the dataset suitable for machine learning models.



Label Encoding

```python
# Convert categorical features to continuous features with Label Encoding
from sklearn.preprocessing import LabelEncoder
lencoders = {}
for col in oversampled.select_dtypes(include=['object']).columns:
    lencoders[col] = LabelEncoder()
    oversampled[col] = lencoders[col].fit_transform(oversampled[col])
```

MICE – Iterative Imputer

The Python code in the visual demonstrates a process that involves the use of the Multiple Imputation by Chained Equations (MICE) method for imputing missing data. This method uses a multi-step approach to fill in missing data by making predictions from other variables.

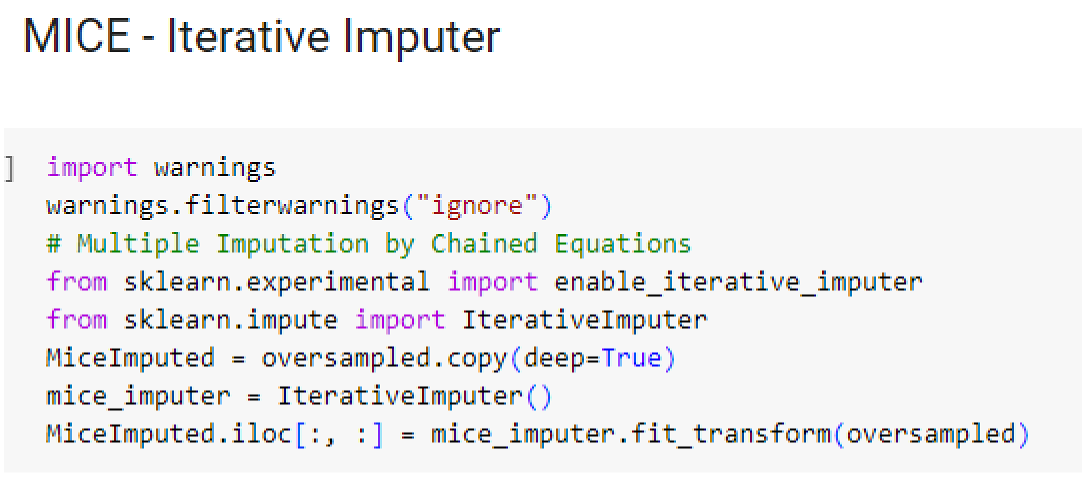
First, a filter is set using the `warnings` library to suppress potential warning messages. This is a common practice to hide warnings that can sometimes arise when experimental features are used.

The use of the MICE method is enabled with `enable_iterative_imputer`. The `IterativeImputer` class from the experimental module of the Scikit-learn library is imported. This class uses information from other features to predict missing data points and attempts to estimate missing values for each feature by building a model for it. The process is repeated several times, iterating over imputations for missing values for each feature, and striving to produce progressively better predictions.

Next, a deep copy of the `oversampled` dataset is taken. This ensures that the imputation process is carried out on a copy without overwriting the original data, preserving the original dataset.

An `IterativeImputer()` object is created, and it is applied to the `oversampled` dataset using the `.fit_transform()` method. This method performs the necessary calculations to impute missing data and returns the imputed dataset. The obtained results are assigned to all columns of the `MiceImputed` DataFrame.

The MICE method is particularly useful when missing values in the dataset are not random and potentially have relationships with other variables. This technique often produces more accurate predictions compared to simple imputation methods (such as filling with mean, median, or mode) because it captures the structure of missing data better by leveraging information from other variables. However, this method comes with a higher computational cost and may require careful parameter tuning.


MICE - Iterative Imputer

```
]  import warnings
   warnings.filterwarnings("ignore")
   # Multiple Imputation by Chained Equations
   from sklearn.experimental import enable_iterative_imputer
   from sklearn.impute import IterativeImputer
   MiceImputed = oversampled.copy(deep=True)
   mice_imputer = IterativeImputer()
   MiceImputed.iloc[:, :] = mice_imputer.fit_transform(oversampled)
```

Detecting Outlier

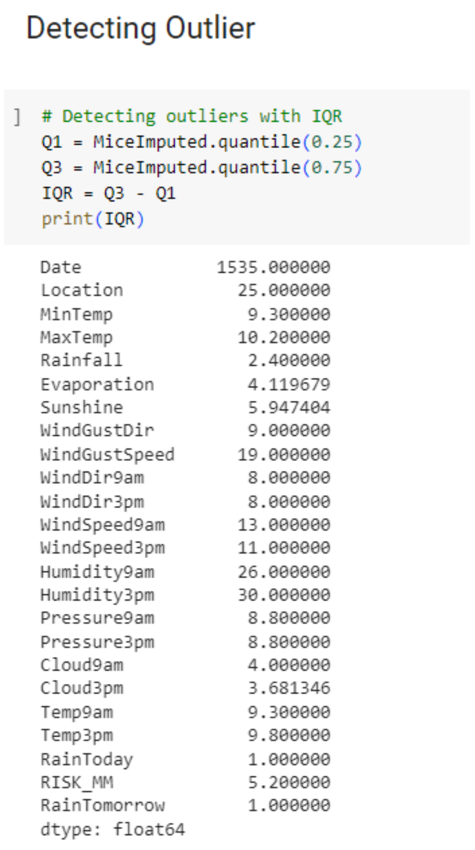
The Python code in the visual demonstrates a method used to detect outliers in the dataset, employing the Interquartile Range (IQR) method, which is commonly used for identifying outliers.

In the code, the `quantile` function is used to calculate the first quartile (Q1) and the third quartile (Q3) values for each column of the `MiceImputed` DataFrame. Q1 represents the lower 25% percentile of the data, while Q3 represents the upper 25% percentile of the data. The IQR is calculated as the difference between these two values and represents the distribution of the middle 50% of the data in the dataset.

IQR values are commonly used for outlier detection. Typically, data points that are smaller than Q1 - 1.5*IQR or larger than Q3 + 1.5*IQR are considered outliers. This method is useful in cases where the data distribution is skewed, for example, when a dataset contains very high or very low values.

The table in the visual displays the calculated IQR values for each column. These values indicate how much the data distribution spreads for each column and can be used as a reference point for outlier detection. For example, we can see that the IQR for the `Rainfall` column is 2.4, indicating relatively low variability in rainfall amounts (most days have low rainfall). On the other hand, a column like `Date` having a very high IQR value may be due to significant differences between dates.

These IQR values can be used in subsequent steps to filter out outliers, which is an important part of the data preprocessing process. Outliers can disrupt statistical analyses and machine learning modeling, so they are often either removed or further examined before analysis.



Detecting Outlier

```
# Detecting outliers with IQR
Q1 = MiceImputed.quantile(0.25)
Q3 = MiceImputed.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
Date             1535.000000
Location           25.000000
MinTemp             9.300000
MaxTemp            10.200000
Rainfall            2.400000
Evaporation         4.119679
Sunshine            5.947404
WindGustDir         9.000000
WindGustSpeed      19.000000
WindDir9am          8.000000
WindDir3pm          8.000000
WindSpeed9am       13.000000
WindSpeed3pm       11.000000
Humidity9am        26.000000
Humidity3pm        30.000000
Pressure9am         8.800000
Pressure3pm         8.800000
Cloud9am            4.000000
Cloud3pm            3.681346
Temp9am             9.300000
Temp3pm             9.800000
RainToday           1.000000
RISK_MM             5.200000
RainTomorrow        1.000000
dtype: float64
```

Removing Outliers

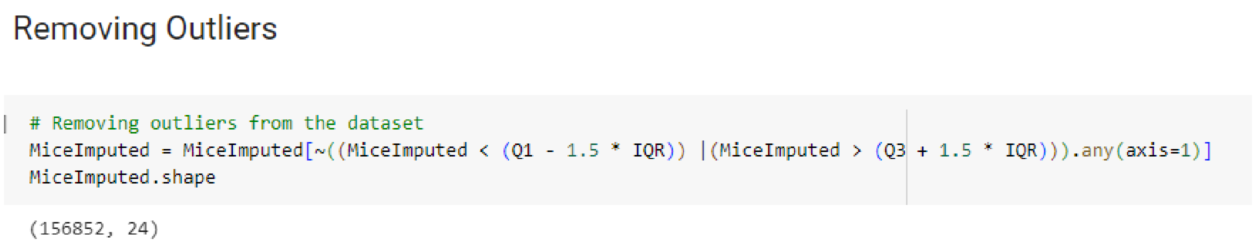
The Python code shown in the visual demonstrates a method used to remove outliers from the dataset. The code identifies data points beyond a certain threshold using the previously calculated Interquartile Range (IQR) and defines them as outliers, subsequently removing them from the dataset.

The expression used in the code checks whether values for each column are less than Q1 - 1.5*IQR or greater than Q3 + 1.5*IQR. Data points that satisfy any of these conditions are considered outliers. The `~` sign denotes 'not' in Boolean indexing and is used here to select non-outliers, meaning normal data points.

The `any(axis=1)` function selects rows where there is an outlier in any column. In other words, if any value in a row's columns is an outlier, that row is generally marked as an outlier and removed from the dataset.

After this process, an output is obtained using the `MiceImputed.shape` command, which shows the dimensions of the new dataset. This output indicates that the dataset has 156,852 rows and 24 columns after removing outliers. This allows us to understand how much the dataset has reduced in size after removing outliers.

Removing outliers can make the dataset cleaner and more consistent, but sometimes outliers may contain important information or be part of the nature of the dataset. Therefore, the removal of outliers should always be carefully evaluated.



Removing Outliers

```
# Removing outliers from the dataset
MiceImputed = MiceImputed[~((MiceImputed < (Q1 - 1.5 * IQR)) |(MiceImputed > (Q3 + 1.5 * IQR))).any(axis=1)]
MiceImputed.shape
```

```
(156852, 24)
```

Correlation Matrix

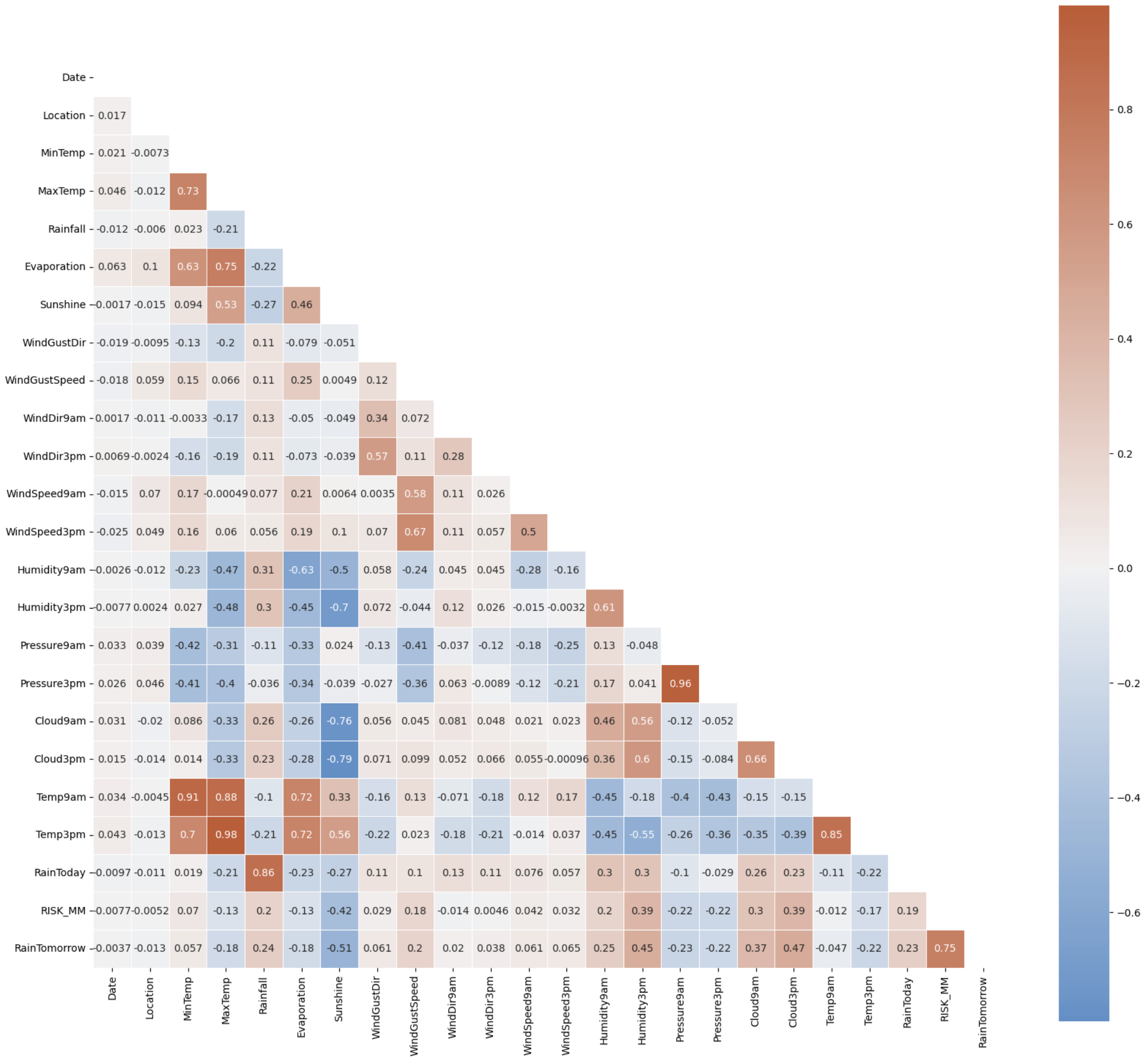
The visual displays a heatmap containing a correlation matrix calculated for a dataset. The correlation matrix is a statistical measure that expresses the strength and direction of relationships between variables in the dataset. Correlation values range from -1 to 1; 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation.

In this heatmap, dark red colors represent high positive correlations, dark blue colors represent high negative correlations, and light colors indicate lower correlation degrees. The entire diagonal line is colored red, which is an expected result because each variable has a perfect positive correlation with itself (1.0).

For example, there is a positive correlation of 0.63 between `MaxTemp` and `Evaporation`, while there is almost a perfect positive correlation (0.96) between `Pressure9am` and `Pressure3pm`. This indicates a strong relationship between morning pressure and afternoon pressure. Additionally, there is a relatively high correlation of 0.73 between `MinTemp` and `MaxTemp`, which is expected because there is often a relationship between the minimum and maximum temperatures.

The `RainTomorrow` column is particularly important as it is related to predicting rainfall for the next day. Here, relatively high positive correlations between `RainTomorrow` and `Humidity3pm` (0.45) and `Cloud3pm` (0.38) are noticeable. This suggests that high humidity and cloudiness may be associated with the probability of rainfall the next day.

The correlation matrix is useful for understanding which variables contain strong relationships with the target variable (in this case, `RainTomorrow`) and can be used during feature selection or machine learning modeling. It can also be used to evaluate whether highly correlated variables need to be removed, as such variables can lead to overfitting of the model.


Date
Location
MinTemp
MaxTemp
Rainfall
Evaporation
Sunshine
WindGustDir
WindGustSpeed
WindDir9am
WindDir3pm
WindSpeed9am
WindSpeed3pm
Humidity9am
Humidity3pm
Pressure9am
Pressure3pm
Cloud9am
Cloud3pm
Temp9am
Temp3pm
RainToday
RISK_MM
RainTomorrow
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

MICE – Imputed Graphs

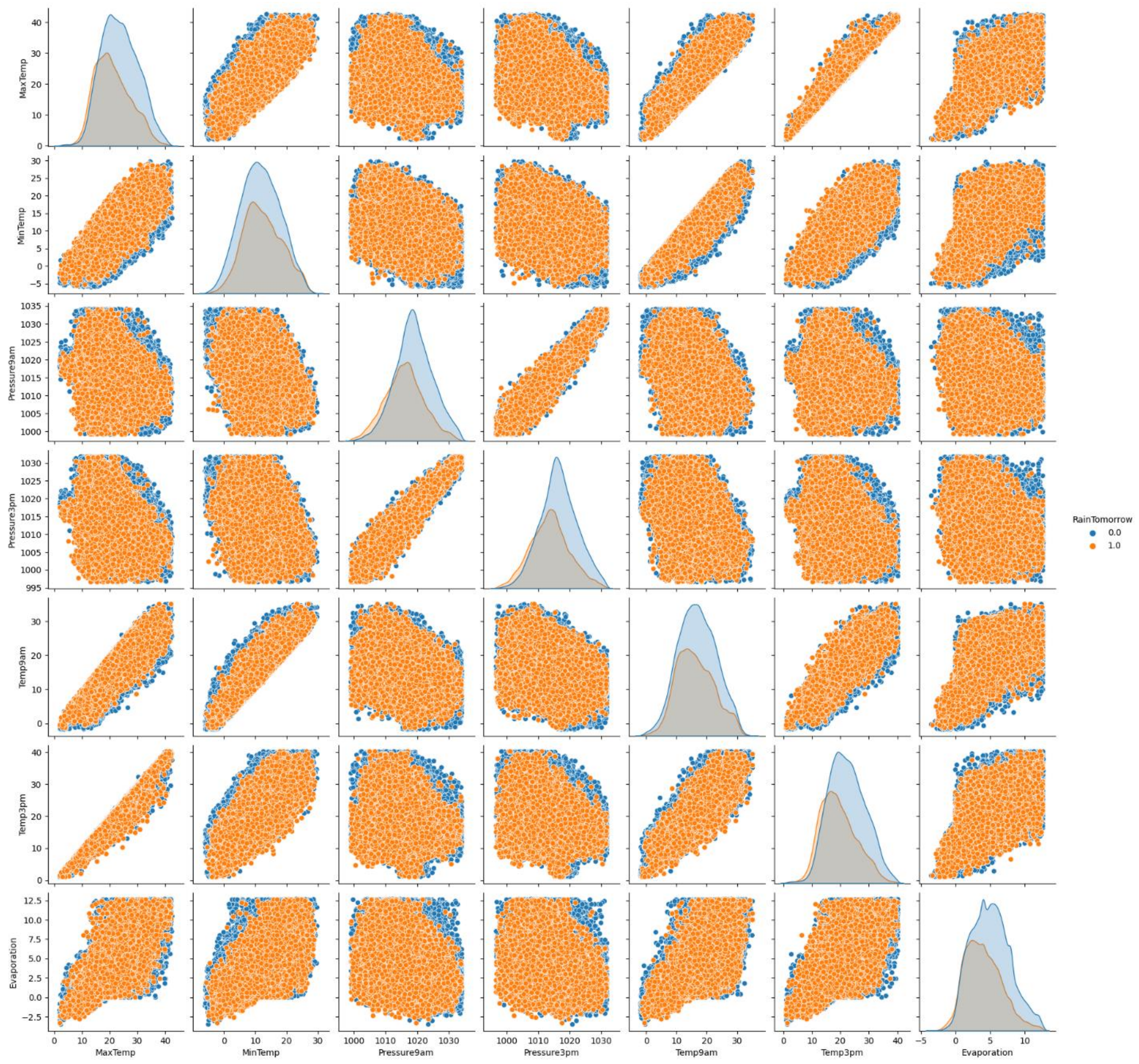
The visual displays a pairplot matrix created using the Seaborn library's `pairplot` function. This matrix includes scatter plots and distribution plots for pairwise combinations of specified variables (MaxTemp, MinTemp, Pressure9am, Pressure3pm, Temp9am, Temp3pm, and Evaporation). The 'RainTomorrow' column is used as the `hue` parameter to visually separate data points based on the next day's rainfall status (1.0 for rain, 0.0 for no rain).

A pairplot matrix is an effective way to visualize relationships between variables and their relationships with the target variable. Scatter plots allow you to observe the nature (linear, non-linear, or no relationship) and strength of relationships between variables. The plots on the diagonal (distribution plots) show the distribution of each variable, providing information about the shape of these distributions (e.g., symmetric, skewed, etc.).

In this visual, blue dots represent days without rain (RainTomorrow = 0), while orange dots represent days with rain (RainTomorrow = 1). For example, looking at the plot between `MaxTemp` and `Evaporation`, you can observe a positive relationship between these two variables; as temperature increases, evaporation also increases. Additionally, it can be seen that rainy days (orange dots) generally have lower maximum temperature and evaporation values.

The plot between `Pressure9am` and `Pressure3pm` shows an almost perfect linear relationship, indicating that morning pressure values are closely related to afternoon pressure values. However, considering the coloring (hue) for rainfall prediction, there is no clear distinction between rainfall status and these two variables.

Overall, this pairplot matrix is a useful tool for exploring the dataset before conducting machine learning modeling and plays an important role in determining which variables may be useful for the model. However, when there are many variables or the dataset is large, creating a pairplot can be time-consuming, and the plots can become complex. In such cases, more focused analyses or dimensionality reduction techniques may be more appropriate.


MaxTemp
MinTemp
Pressure9am
Pressure3pm
Temp9am
Temp3pm
Evaporation
RainTomorrow
0.0
1.0

Data Standardizing

The Python code in the visual demonstrates the process of data standardization. This process scales columns of data with different scales to a common scale to ensure more effective functioning of machine learning models. The `MinMaxScaler` method used in the code scales each feature to a range of 0 to 1, which transforms the minimum value of each feature to 0 and the maximum value to 1. Explanation:

1. The `MinMaxScaler` class is imported from the `sklearn.preprocessing` module. This class is used for feature scaling and transforms the values of each feature to fit within a specified range.

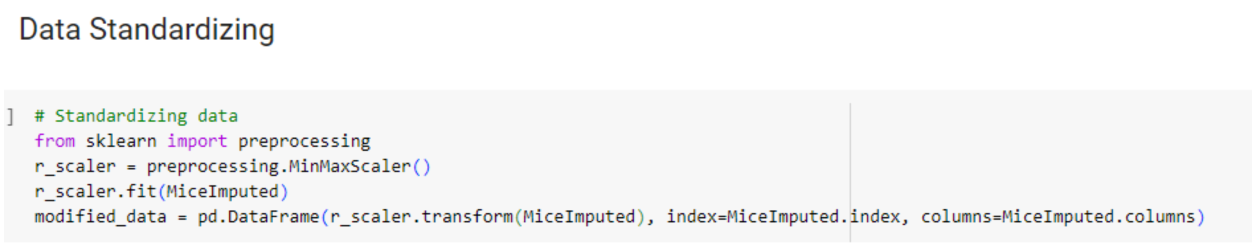
2. An instance (`r_scaler`) is created from the `MinMaxScaler` class.

3. The `.fit()` method is used to compute the minimum and maximum values for each feature in the `MiceImputed` dataset. This gathers the statistics required for scaling.

4. The `.transform()` method is used to scale all features in the dataset using the collected statistics. The scaled data will transform each feature's values to fall within the range of 0 to 1.

5. The scaled data is then transformed into a new DataFrame (`modified_data`). This DataFrame preserves the index and column names of the original `MiceImputed` DataFrame.

Scaling allows all features in the dataset to be evaluated equally by the model and prevents biases arising from feature scales. Particularly, algorithms based on distances between features (e.g., k-Nearest Neighbors and support vector machines) and optimization algorithms (e.g., those using gradient descent) perform better with scaled data.


Data Standardizing

```
# Standardizing data
from sklearn import preprocessing
r_scaler = preprocessing.MinMaxScaler()
r_scaler.fit(MiceImputed)
modified_data = pd.DataFrame(r_scaler.transform(MiceImputed), index=MiceImputed.index, columns=MiceImputed.columns)
```

Chi – Square

The Python code in the visual demonstrates a method used for feature selection, specifically using the Chi-Square statistic to determine the most important features. The Chi-Square test is used for categorical data to perform an independence test and measure the strength of the relationship between a feature and the target variable. Explanation:

1. The `SelectKBest` and `chi2` functions are imported from the `sklearn.feature_selection` module. `SelectKBest` is a method used for selecting the best features, and `chi2` is the scoring function used for this selection.

2. From the `modified_data` DataFrame, `X` (features matrix) and `y` (target vector) are created. Here, `X` includes all columns except the 'RainTomorrow' column, and `y` is the 'RainTomorrow' column itself.

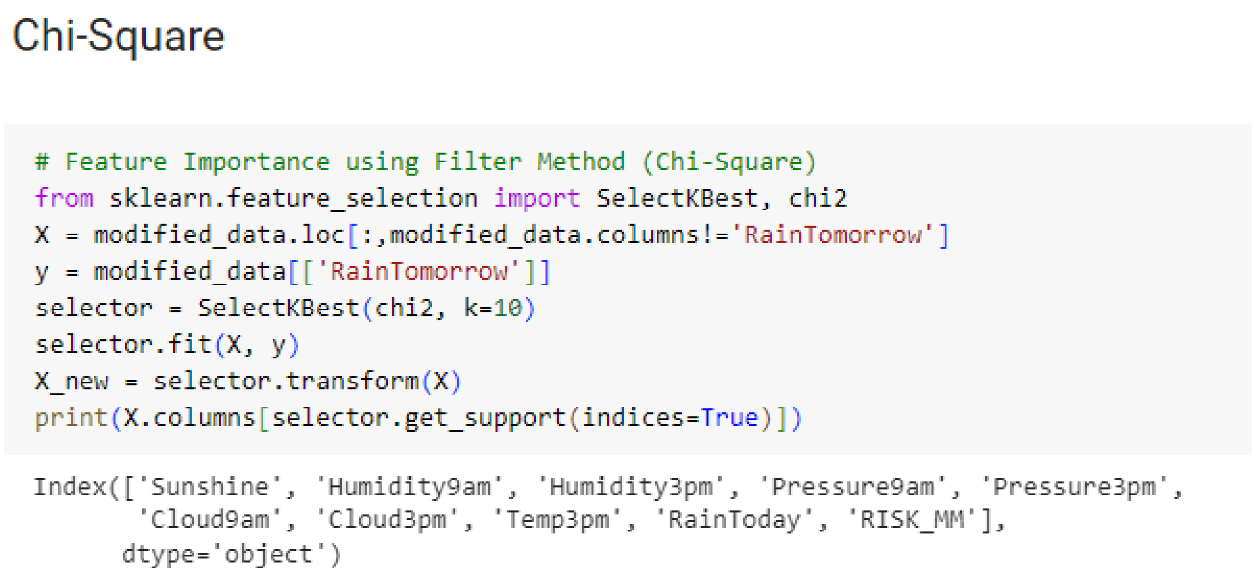
3. A `selector` object is created with `SelectKBest(chi2, k=10)`. This `selector` will choose the top 10 features based on the Chi-Square score.

4. The `.fit(X, y)` method applies the Chi-Square test on the given features matrix (`X`) and target vector (`y`).

5. The `.transform(X)` method creates a new features matrix (`X_new`) containing the selected features.

6. Finally, the names of the selected top 10 features are obtained using `selector.get_support(indices=True)` and printed.

As a result, the `print` statement displays the names of the selected top features. These features are listed as 'Sunshine', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'RainToday', and 'RISK_MM'. These features are determined to have the strongest relationship with the 'RainTomorrow' target variable and are recommended for use when training a machine learning model. Feature selection is important to reduce model complexity, shorten training times, and improve generalizability.



Chi-Square

```python
# Feature Importance using Filter Method (Chi-Square)
from sklearn.feature_selection import SelectKBest, chi2
X = modified_data.loc[:,modified_data.columns!='RainTomorrow']
y = modified_data[['RainTomorrow']]
selector = SelectKBest(chi2, k=10)
selector.fit(X, y)
X_new = selector.transform(X)
print(X.columns[selector.get_support(indices=True)])
```

```
Index(['Sunshine', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm',
       'Cloud9am', 'Cloud3pm', 'Temp3pm', 'RainToday', 'RISK_MM'],
      dtype='object')
```

Logistic Regression

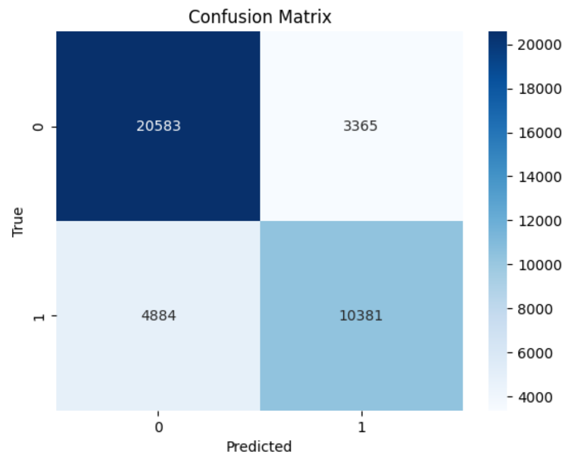
The visuals depict two different metrics used to evaluate the performance of a logistic regression model: the Receiver Operating Characteristic (ROC) Curve and the Confusion Matrix.

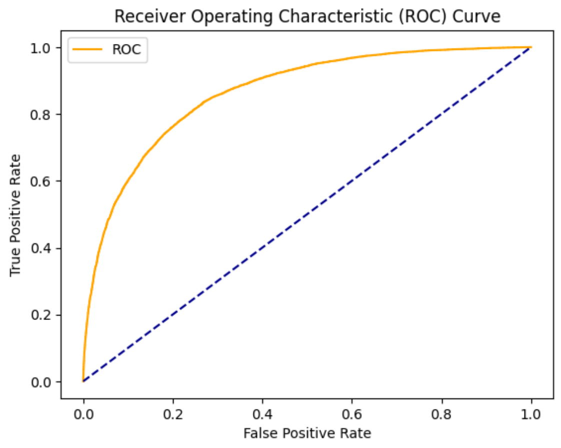
The first visual displays the ROC curve of the model. The ROC curve is used to assess the classification performance of the model at various threshold values. The X-axis of the curve represents the False Positive Rate (FPR), while the Y-axis represents the True Positive Rate (TPR, also known as sensitivity or recall). A perfect classifier would have an ROC curve close to the upper-left corner (TPR=1, FPR=0). In the visual, the ROC curve is situated below a perfect classifier and above a random guess (dotted line). This indicates that the model performs better than a random guess but is not perfect.

The second visual shows the confusion matrix of the model. The confusion matrix displays the distribution of model predictions with respect to actual labels.

These values in the matrix help us understand how the model predicts both the positive class (rain) and the negative class (no rain). The model performs quite well in making true positive predictions (rain), but it also makes a certain number of false positive and false negative predictions. This suggests a balanced approach in terms of the model's precision and recall.

Overall, while the model's performance appears to be acceptable, there may be room for improvements to reduce the number of false positives and false negatives. To gain a more detailed understanding of the model's precision and recall, it is recommended to calculate other metrics such as precision, recall, and F1 score. These metrics are used to evaluate the model's performance more comprehensively.





```
Accuracy = 0.78963609007217
ROC Area under Curve = 0.76976981489154
Cohen's Kappa = 0.5494516226632338
Time taken = 4.06925892829895
              precision    recall  f1-score   support

         0.0    0.80822   0.85949   0.83307     23948
         1.0    0.75520   0.68005   0.71566     15265

    accuracy                        0.78964     39213
   macro avg    0.78171   0.76977   0.77436     39213
weighted avg    0.78758   0.78964   0.78736     39213
```

Decision Tree

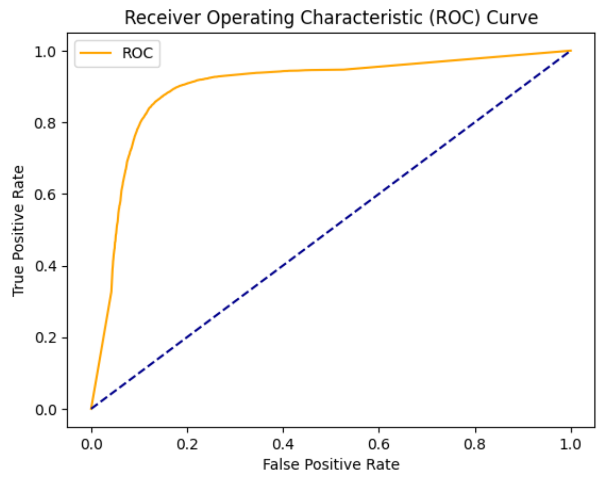
The visuals include an ROC Curve and a Confusion Matrix, both depicting the classification performance of a decision tree model.

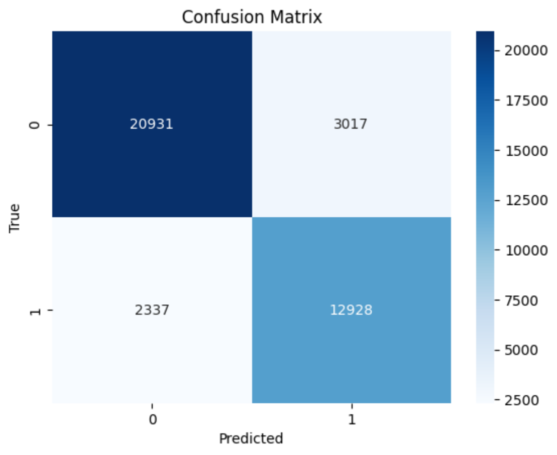
The ROC Curve in the first visual is a graph that visualizes the classification performance of the model. The ROC Curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR). An ideal classifier maximizes TPR while minimizing FPR, causing the curve to move towards the upper-left corner. In this case, the ROC curve lies between an ideal and random guess, indicating that the model makes some true positive predictions while also producing false positives. The closer the curve is to the upper part, the better the model's performance.

The second visual displays the Confusion Matrix of the model. This matrix shows the counts of correct and incorrect predictions made by the model.

The matrix reveals that the model predicts no rain days quite successfully (high TN), but it produces some false negatives (FN) on days when it will rain. The model's ability to correctly predict rainy days (TP) indicates balanced performance relative to the false alarm rate (FP).

Overall, the decision tree model exhibits balanced performance with a good TPR and an acceptable FPR. However, evaluating a model's performance based solely on these two visuals is not sufficient. The model should also be assessed using metrics like precision, recall, F1 score, and others to provide a more comprehensive understanding of its actual performance.





```
Accuracy = 0.8634636472598373
ROC Area under Curve = 0.8604616955581945
Cohen's Kappa = 0.7151480370968681
Time taken = 0.5194563865661621
              precision    recall  f1-score   support

         0.0    0.89956   0.87402   0.88661     23948
         1.0    0.81079   0.84690   0.82845     15265

    accuracy                        0.86346     39213
   macro avg    0.85517   0.86046   0.85753     39213
weighted avg    0.86500   0.86346   0.86397     39213
```

Random forest

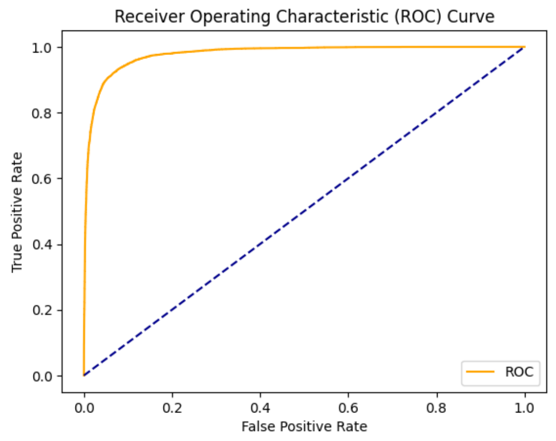
The visuals include an ROC Curve and a Confusion Matrix depicting the classification performance of a Random Forest classifier model.

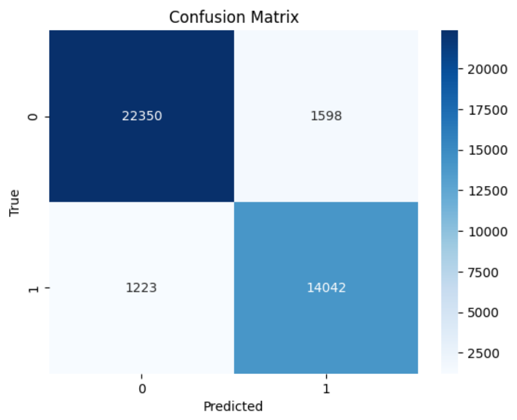
The ROC Curve in the first visual is a graph that visualizes the classification performance of the model. The ROC Curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR). An ideal classifier maximizes TPR while minimizing FPR, causing the curve to move towards the upper-left corner. In this case, the ROC curve is quite close to perfect performance, indicating that the model effectively distinguishes the positive class and keeps the false positives to a minimum.

The second visual displays the Confusion Matrix of the model. This matrix shows the counts of correct and incorrect predictions made by the model.

The matrix reveals that the model predicts no rain days quite successfully (high TN) and also makes a high number of true positive predictions on rainy days (high TP). The counts of false positives (FP) and false negatives (FN) are relatively low, indicating that the model exhibits balanced performance overall.

In general, based on both the shape of the ROC Curve and the values in the Confusion Matrix, it can be concluded that the Random Forest model performs well in predicting rain. However, evaluating the model's precision, recall, F1 score, and other performance metrics alongside these assessments would provide a more comprehensive evaluation.





```
Accuracy = 0.9280595720806876
ROC Area under Curve = 0.9265770863620824
Cohen's Kappa = 0.8493714659078331
Time taken = 34.458879470825195
              precision    recall  f1-score   support

         0.0    0.94812   0.93327   0.94064     23948
         1.0    0.89783   0.91988   0.90872     15265

    accuracy                        0.92806     39213
   macro avg    0.92297   0.92658   0.92468     39213
weighted avg    0.92854   0.92806   0.92821     39213
```

XGBoost

The visuals include an ROC Curve and a Confusion Matrix depicting the classification performance of an XGBoost classifier model.

The ROC Curve in the first visual is a graph that visualizes the classification performance of the model. The ROC Curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR). An ideal classifier maximizes TPR while minimizing FPR, causing the curve to move towards the upper-left corner. In this case, the ROC curve is quite close to perfect performance, indicating that the model effectively distinguishes the positive class and keeps false positives to a minimum.

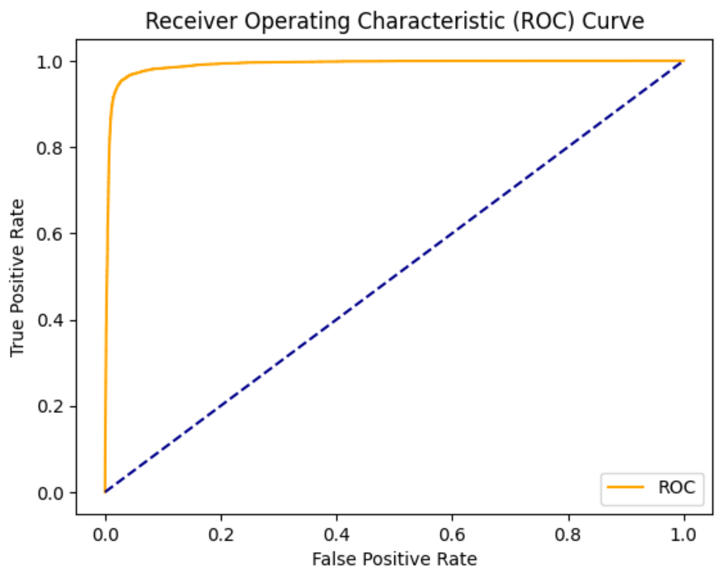
The second visual displays the Confusion Matrix of the model. This matrix shows the counts of correct and incorrect predictions made by the model:

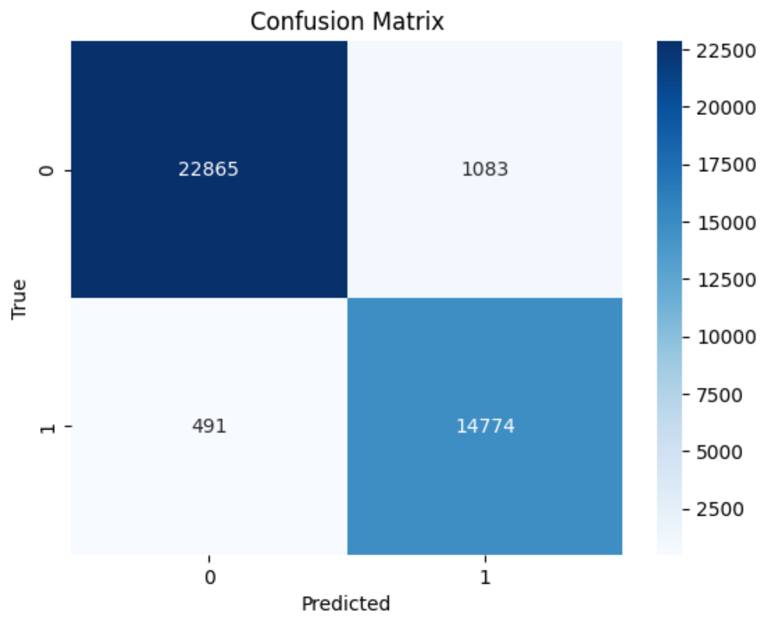
The matrix reveals that the XGBoost model effectively predicts no-rain days (high TN) and makes a low number of false negative predictions on rainy days (low FN). At the same time, the count of false positives (FP) is relatively low, indicating that the model maintains a low rate of false alarms.

When the ROC curve and the Confusion Matrix are considered together, it can be concluded that the XGBoost model's classification performance is quite good. The ROC curve shows that the model has good sensitivity (TPR), while the Confusion Matrix indicates that the model exhibits balanced precision and recall performance.

This performance implies that the model minimizes false positives (marking no-rain days as rainy) and false negatives (marking rainy days as no-rain) to a minimum degree. This suggests that the XGBoost model is capable of making reliable and balanced precipitation predictions.

To gain a more comprehensive understanding of how the model performs in real-world data, it would be advisable to evaluate it with other performance metrics such as precision, recall, and F1 score. However, direct information about these metrics cannot be obtained from the provided visuals, and additional analyses may be required.





```
Accuracy = 0.9598602504271543
ROC Area under Curve = 0.9613059666727413
Cohen's Kappa = 0.916170658468024
Time taken = 64.77473568916321
              precision    recall  f1-score   support

         0.0    0.97898   0.95478   0.96673     23948
         1.0    0.93170   0.96783   0.94942     15265

    accuracy                        0.95986     39213
   macro avg    0.95534   0.96131   0.95808     39213
weighted avg    0.96057   0.95986   0.95999     39213
```

Decision Boundaries

The visual depicts the decision boundaries of four different classification models: Logistic Regression, Decision Tree, Random Forest, and XGBoost. Each model has been trained to classify the likelihood of rain the following day based on the "Sunshine" and "Humidity9am" features in a two-dimensional space, and decision boundaries have been drawn. The "Cloud3pm" feature has also been taken into account, but a fixed value has been used for visualization purposes. Each subplot represents the decision boundaries of a different model:

-Logistic Regression: It shows a smoother and more linear boundary, indicating that the model tends to make a linear separation.

-Decision Tree: It exhibits sharper and more irregular boundaries, suggesting that the model can potentially overfit the data.

-Random Forest: Compared to the Decision Tree, it shows more regular boundaries but still has complex decision boundaries, indicating that the model considers different feature combinations.

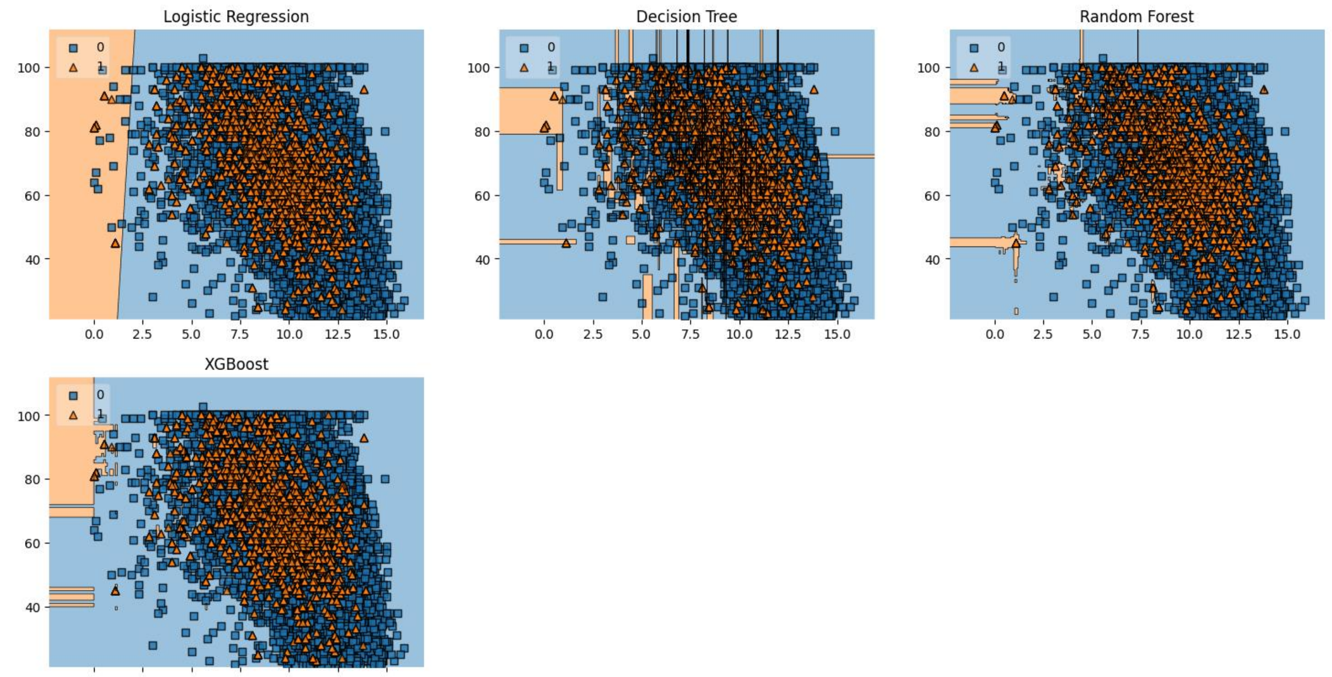
-XGBoost: The decision boundaries are similar to Random Forest but smoother and more inclined to capture general trends.

Colors represent different classes: orange represents no rain (0), and blue represents rain (1). Points within the decision region indicate the class predicted by the model for that region. Triangles and squares represent the actual labels. For example, orange triangles represent days when it didn't rain in reality, and blue squares represent days when it did rain.

This visualization is useful for understanding the decision-making mechanisms of different models and how predicted classes are distributed. To evaluate the overall performance of the model, besides decision boundaries, other performance metrics such as confusion matrix, ROC curve, and other metrics should be considered. Decision boundaries can be a good tool to comprehend the logic of model decision-making but may not be sufficient to understand how well the model generalizes to real-world data.

From these graphics, we can observe how well each model performs in specific regions and where they make misclassifications. For example, a model having many mixed triangles and squares in blue and orange regions suggests that the model struggles to classify the data correctly in that region.

Evaluating the performance of each of these four models, taking into account the complexity of the dataset and features, is essential. Additionally, fine-tuning the models' hyperparameters and employing techniques like cross-validation can further enhance performance.


Logistic Regression
Decision Tree
Random Forest
XGBoost
0
1
100
80
60
40
0.0
2.5
5.0
7.5
10.0
12.5
15.0

Resources

1. "Automated predictive analytics tool for rainfall forecasting." *Scientific Reports*. [Online]. Available: https://www.nature.com/articles/s41598-021-84102-0.

2. N. K. Goyal and V. Diwanji, "Machine learning techniques to predict daily rainfall amount." *Journal of Big Data*, vol. 6, no. 1, Dec. 2019. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0266-3.

3. N. Oswal, "Predicting Rainfall using Machine Learning Techniques," ResearchGate, 2020. [Online]. Available: https://www.researchgate.net/publication/341844875_Predicting_Rainfall_using_Machine_Learning_Techniques.

4. "Rainfall Prediction System Using Machine Learning Fusion for Smart Cities," *PMC*, 2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9099780/. [Accessed: Jan. 25, 2024].