RAIN PREDICTION – WEATHER FORCASTING

META SCIFOR TECHNOLOGIES

2024

Script. Sculpt. Socialize

Guided By:

Saurav Kumar

Submitted By:

Bhushan Khushal Nimje

MST02-0020

TABLE OF CONTENT

ABSTRACT

This project presents the development of an advanced Rain Prediction and Weather Forecasting System designed to predict both the likelihood of rain tomorrow and the amount of expected rainfall. The primary objective is to analyse weather patterns by exploring the relationships between key meteorological features such as temperature, humidity, wind speed, and atmospheric pressure, providing valuable insights into rainfall predictions.

To achieve this, we employed six machine learning models: Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Linear Regression, Decision Tree Regressor and Random Forest Regressor. The classification models (Logistic Regression, Decision Tree, and Random Forest) were utilized to predict whether it will rain tomorrow, while the regression model (Linear Regression, Decision Tree Regressor and Random Forest Regressor) was applied to predict the amount of rainfall. These models were trained on a comprehensive dataset containing essential weather features, and were rigorously evaluated using performance metrics to ensure accuracy and reliability in both classification and regression tasks.

This Rain Prediction and Weather Forecasting System exemplifies the potential of machine learning models to enhance forecasting precision and support informed decision-making in weather-dependent industries. The project also outlines possibilities for future improvements, such as incorporating more advanced models like neural networks and integrating real-time weather data to increase the predictive power of the system. This abstract encapsulates the project's scope, methodology, and outcomes, offering a comprehensive overview of the Rain Prediction and Weather Forecasting System.

INTRODUCTION

In recent years, the importance of accurate weather forecasting has grown significantly due to its profound impact on various sectors such as agriculture, infrastructure planning, disaster management, and daily life. Rainfall prediction, in particular, is critical as it directly influences decision-making processes that can affect millions of people. Accurately forecasting rainfall is essential for mitigating risks associated with natural events like floods, droughts, and storms.

This study focuses on developing machine learning models to predict rainfall using a comprehensive dataset that includes various meteorological features. The dataset contains key attributes such as temperature, humidity, wind speed, atmospheric pressure, and historical rainfall data. By analysing these features, the objective is to create predictive models capable of accurately forecasting rainfall, thereby enhancing the reliability of weather predictions.

To achieve this, we employ six machine learning algorithms: Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Linear Regression, Decision Tree Regressor and Random Regressor. The classification models (Logistic Regression, Decision Tree, and Random Forest) are utilized to predict whether it will rain tomorrow, while the regression model (Linear Regression, Decision Tree Regressor and Random Forest Regressor) is used to estimate the amount of rainfall. Each algorithm brings distinct advantages to the table. Logistic Regression offers simplicity and interpretability, Decision Trees capture non-linear relationships with ease, and Random Forest, as an ensemble method, boosts accuracy by aggregating predictions from multiple decision trees.

This project underscores the potential of machine learning models in advancing weather forecasting. By comparing the performance of these models, we aim to identify the most effective algorithm for rainfall prediction across various weather scenarios. Ultimately, this work provides valuable insights that can be leveraged to improve decision-making in weather-dependent industries and beyond.

TECHNOLOGIS USED

The technology used in this project consists a variety of tools essential for building a machine learning model. Below is a detailed overview of the technologies used:

1. Python Programming Language:

- The primary language used for data analysis, preprocessing, and machine learning model development.

2. Libraries and Packages:

- **Pandas:** Utilized for data manipulation and analysis. It helps in loading the dataset, handling missing values, and performing various data transformation operations.
- **NumPy:** Used for numerical operations, particularly for array and matrix operations required in data preprocessing.
- **Matplotlib:** Employed for visualizing data through plots and graphs, such as boxplots for identifying outliers.
- **Seaborn**: A statistical data visualization library built on Matplotlib, used to create more aesthetically pleasing and informative plots.
- **Scipy:** Contains the zscore function for detecting outliers using the Z-Score method.
- **Scikit-learn**: A versatile machine learning library used for:
    1. **Label Encoding**: Converts categorical values into numerical format.
    2. **Feature Scaling**: Standardizes features by scaling them to a standard range, using StandardScaler.
    3. **Feature Selection**: Identifies the most significant features using techniques like SelectKBest.
    4. **Model Building**: Implements various machine learning algorithms (e.g., Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Linear Regression) and hyperparameter tuning.
    5. **SMOTE**: Synthetic Minority Over-sampling Technique used for handling class imbalance in the dataset.
    6. **Hyperparameter Tuning**: Adjusts model parameters using techniques like GridSearchCV for optimal performance.
    7. **LassoCV and Ridge**: Regularization techniques for linear regression to prevent overfitting.

3. Data Preprocessing Techniques:

- **Handling Missing Values**: Imputes missing values with statistical measures such as mean for numerical features and mode for categorical features.
- **Outlier Detection and Removal**:
    1. **Z-Score Method**: Identifies and removes outliers based on the standard deviation of the data.

2. **IQR (Interquartile Range) Method**: Detects and removes outliers based on the range between the first quartile (Q1) and third quartile (Q3) of the data.

- **Feature Extraction**: Extracts new features from existing ones (e.g., day, month, year from the date column).
- **Feature Encoding**: Converts categorical features into numerical values using Label Encoding.
- **Feature Scaling**: Standardizes the feature set to ensure uniformity in scale and enhance model performance.

4. Machine Learning Models:

- ### Classification Models:
    1. **Logistic Regression:** Predicts the probability of binary outcomes (e.g., whether it will rain tomorrow).
    2. **Decision Tree Classifier:** Creates a model that splits data into branches to predict outcomes.
    3. **Random Forest Classifier:** An ensemble method that combines multiple decision trees to improve predictive performance.
- ### Regression Models:
    1. **Linear Regression**: Models the relationship between dependent and independent variables.
    2. **LassoCV and Ridge**: Regularization techniques to improve linear regression models by penalizing large coefficients.
    3. **Decision Tree Regressor**: A tree algorithm which offers more flexibility by capturing non-linear relationships.
    4. **Random Forest Regressor:** An ensemble learning method implemented using Scikit-learn to build a robust predictive model by combining multiple decision trees.

5. Model Training:

- **Model Training:** The .fit() method in Scikit-learn is used to train the models on the prepared dataset.
- **Hyperparameter Tuning:** Techniques like GridSearchCV are used to find the optimal parameters for models like Decision Tree Classifier, Random Forest Classifier, Decision Tree Regressor and Random Forest Regressor. For Linear Regression, LassoCV is employed to enhance model performance**.**

6. Feature Engineering and Selection:

- **Variance Inflation Factor (VIF):** Assesses multicollinearity among features to ensure that the features are not highly correlated.
- **SelectKBest:** Selects the top k features that contribute most significantly to the target variable.

7. Data Imbalance Handling:

- **SMOTE (Synthetic Minority Over-sampling Technique):** Addresses class imbalance by generating synthetic examples for the minority class.

8. Data Transformation:

- **PowerTransformer:** Applies transformations to reduce skewness in data, making it more normally distributed.

9. Model Evaluation:

- **Evaluation Metrics:** For evaluating the performance of the machine learning models, different evaluation metrics are used depending on whether the task is classification or regression.
    1. **Classification Metrics (For predicting whether it will rain tomorrow):**
        - **Accuracy:** Measures the proportion of correct predictions out of the total predictions made.
        - **Precision**: Indicates the proportion of true positive predictions out of all positive predictions made.
        - **Recall:** Measures the proportion of actual positives that are correctly identified.
        - **F1 Score:** Harmonic mean of Precision and Recall, providing a balanced measure that considers both false positives and false negatives.
    2. **Regression Metrics (For predicting how much rainfall could be there):**
        - Evaluating the model with metrics like R2 Score, Mean Absolute Error (MAE) and Mean Squared Error (MSE).

DATASET INFORMATION

The dataset used in this project plays a crucial role in building and validating the model. Below is the detailed overview of the dataset:

Dataset Size:

The dataset comprises 23 columns of daily weather observations from various locations. It includes 142,193 rows, each representing a specific day's weather data at a particular location.

Features:

- **Date:** The date when the observation was recorded.
- **Location:** The common name of the location of the weather station.
- **Min Temp:** The minimum temperature of the day in degrees Celsius.
- **Max Temp:** The maximum temperature of the day in degrees Celsius.
- **Rainfall:** The amount of rainfall recorded for the day in millimetres.
- **Evaporation:** The evaporation in millimetres over 24 hours to 9 am.
- **Sunshine:** The total hours of bright sunshine recorded in a day.
- **Wind Gust Dir:** The direction of the strongest wind gust recorded within 24 hours.
- **Wind Gust Speed:** The speed of the strongest wind gust recorded within 24 hours in km/h.
- **WindDir9am:** The wind direction at 9 am.
- **WindDir3pm:** The wind direction at 3 pm.
- **WindSpeed9am:** The wind speed averaged over 10 minutes prior to 9 am in km/h.
- **WindSpeed3pm**: The wind speed averaged over 10 minutes prior to 3 pm in km/h.
- **Humidity9am:** The humidity percentage at 9 am.
- **Humidity3pm:** The humidity percentage at 3 pm.
- **Pressure9am:** Atmospheric pressure reduced to mean sea level at 9 am in hPa.
- **Pressure3pm:** Atmospheric pressure reduced to mean sea level at 3 pm in hPa.
- **Cloud9am:** The fraction of the sky obscured by cloud at 9 am.
- **Cloud3pm:** The fraction of the sky obscured by cloud at 3 pm.
- **Temp9am:** Temperature recorded at 9 am in degrees Celsius.
- **Temp3pm:** Temperature recorded at 3 pm in degrees Celsius.
- **Rain Today:** A binary indicator of whether precipitation exceeded 1 mm in the 24 hours leading to 9 am.
- **Rain Tomorrow**: The amount of rain predicted for the following day in millimetres, used as a measure of "risk."

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8425 entries, 0 to 8424
Data columns (total 23 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Date           8425 non-null   object 
 1   Location       8425 non-null   object 
 2   MinTemp        8425 non-null   float64
 3   MaxTemp        8425 non-null   float64
 4   Rainfall       8425 non-null   float64
 5   Evaporation    8425 non-null   float64
 6   Sunshine       8425 non-null   float64
 7   WindGustDir    8425 non-null   object 
 8   WindGustSpeed  8425 non-null   float64
 9   WindDir9am     8425 non-null   object 
 10  WindDir3pm     8425 non-null   object 
 11  WindSpeed9am   8425 non-null   float64
 12  WindSpeed3pm   8425 non-null   float64
 13  Humidity9am    8425 non-null   float64
 14  Humidity3pm    8425 non-null   float64
 15  Pressure9am    8425 non-null   float64
 16  Pressure3pm    8425 non-null   float64
 17  Cloud9am       8425 non-null   float64
 18  Cloud3pm       8425 non-null   float64
 19  Temp9am        8425 non-null   float64
 20  Temp3pm        8425 non-null   float64
 21  RainToday      8425 non-null   object 
 22  RainTomorrow   8425 non-null   object 
dtypes: float64(16), object(7)
memory usage: 1.5+ MB
```

METHODOLOGY

The development of a weather prediction model using the "weatherAUS.csv" dataset follows a structured approach comprising several key stages: Data Preprocessing, Outlier Detection, Feature Engineering, Model Selection, Model Training, and Evaluation. Below is a detailed explanation of each stage:

1. Data Preprocessing:

Data preprocessing is a crucial step in preparing the dataset for model building. It involves handling missing values, converting data types, and organizing the dataset for subsequent analysis.

- **Loading the Data:** The dataset is loaded using Pandas, with a check on its shape and a preview of the first few records.
- **Handling Missing Values:**
  - Missing values are identified using the .isna().sum() method.
  - For numerical columns like Min Temp, Max Temp, Rainfall, etc., missing values are filled with the mean of the respective columns.
  - For categorical columns like WindGustDir, WindDir9am, and WindDir3pm, the missing values are filled with the mode (most frequent value).
- **Date Conversion and Feature Extraction:**
  - The Date column is converted to a datetime object.
  - Day, Month, and Year are extracted from the Date column and stored in separate columns.
  - The original Date column is then dropped as it's no longer needed.

2. Outlier Detection and Removal:

Outliers can significantly skew the model's performance, so they are detected and handled carefully.

- **Boxplot Visualization:**
  - A boxplot is used to visually identify outliers in the numerical columns.
- **Outlier Removal Methods:**
  - **Z-Score Method:** The Z-Score method is applied to remove data points that are more than three standard deviations away from the mean.
  - **Interquartile Range (IQR) Method:** The IQR method is another approach to remove outliers. It calculates the range between the first quartile (Q1) and the third quartile (Q3) and removes any data points lying beyond 1.5 times the IQR from Q1 and Q3.
- **Selection of Method:** The Z-Score method is chosen over IQR because the IQR method results in the loss of nearly 50% of the data, which could negatively impact model performance.

3. Feature Engineering:

Feature engineering involves transforming and scaling the features to make the data suitable for machine learning models.

- **Encoding Categorical Variables:**
    - Categorical columns are encoded using LabelEncoder, transforming the categories into numerical values.
- **Checking Skewness and Transformation:**
    - The skewness of the data is checked using the .skew() method.
    - Power Transformation is applied to reduce skewness and stabilize the variance.
- **Feature Scaling:**
    - Standardization is applied using StandardScaler to ensure that the features are on the same scale, which is crucial for models like Logistic Regression.

4. Model Building:

Two different problem statements are addressed using separate machine learning models.

- **Problem Statement a: Predicting Rain Tomorrow (Classification)**
    - **Logistic Regression:** A linear model that predicts the probability of rain tomorrow.
    - **Decision Tree Classifier:** A non-linear model that splits the data based on feature values.
    - **Random Forest Classifier:** An ensemble model that builds multiple decision trees and averages their predictions to improve accuracy.
    - **Hyperparameter Tuning:** Grid Search CV is used to find the best combination of hyperparameters for the models.
- **Problem Statement b: Predicting Rainfall Amount (Regression)**
    - **Linear Regression:** A linear model that predicts the amount of rainfall based on the features.
    - **LassoCV and Ridge:** Linear models with regularization to prevent overfitting, optimized using cross-validation.
    - **Decision Tree Regressor:** Decision Tree Regressor can model non-linear relationship between the features and the target. It can naturally handle both numerical and categorical features without the need for extensive preprocessing like encoding.
    - **Random Forest Regressor:** Random Forest is an ensemble learning method that builds multiple decision trees and combines their predictions to produce a more accurate and stable result. Thiis ensemble approach reduces the risk of overfitting that single decision tree might suffer from, leading to better generalization on unseen data.

5. Model Training & Evaluation:

After building the models, they are trained on the dataset and evaluated to measure their performance.

- **Training:**
    - The dataset is split into training and testing sets in a 70:30 ratio.
    - The models are trained on the training set, learning patterns from the input features.
- **Evaluation Metrics:**
    - For classification models, accuracy, precision, recall, and F1-score are used as evaluation metrics.
    - For regression models, R-squared, Mean Absolute Error (MAE), and Mean Squared Error (MSE) are used to measure the model's performance.

6. Hyperparameter Tuning:

To further improve the model performance, hyperparameter tuning is performed using techniques like:

- **LassoCV and RidgeCV:** Regularization techniques that penalize large coefficients, preventing overfitting and handling multicollinearity.
- **Grid Search CV:** A systematic method to search for the best combination of hyperparameters, enhancing the model's accuracy and robustness.

CODE SNIPPET

Installing Dependencies

```
[1]: import pandas as pd
     import numpy as np

     import seaborn as sns
     import matplotlib.pyplot as plt
     %matplotlib inline

     import warnings
     warnings.filterwarnings('ignore')
```

Data Loading

```
pd.set_option("display.max_columns",None)
data = pd.read_csv("weatherAUS.csv")
data.head()
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	WNW	20.0	2
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	WSW	4.0	2
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	WSW	19.0	2
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	E	11.0	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	NW	7.0	2

```
#Lets check the shape of the dataset
data.shape
```

```
(8425, 23)
```

Data Preprocessing

```
: #Lets check for isnull
  data.isna().sum()
```

```
: Date               0
  Location           0
  MinTemp           75
  MaxTemp           60
  Rainfall         240
  Evaporation     3512
  Sunshine        3994
  WindGustDir      991
  WindGustSpeed    991
  WindDir9am       829
  WindDir3pm       308
  WindSpeed9am      76
  WindSpeed3pm     107
  Humidity9am       59
  Humidity3pm      102
  Pressure9am     1309
  Pressure3pm     1312
  Cloud9am        2421
  Cloud3pm        2455
  Temp9am           56
  Temp3pm           96
  RainToday        240
  RainTomorrow     239
  dtype: int64
```

Here we can see that we have null values.

Filling null values with mean values

```
#Filling null values in MinTemp with it's mean
data['MinTemp'] = data['MinTemp'].fillna(data['MinTemp'].mean())

#Filling null values in MaxTemp with it's mean
data['MaxTemp'] = data['MaxTemp'].fillna(data['MaxTemp'].mean())

#Filling null values in Rainfall with it's mean
data['Rainfall'] = data['Rainfall'].fillna(data['Rainfall'].mean())

#Filling null values in Evaporation with it's mean
data['Evaporation'] = data['Evaporation'].fillna(data['Evaporation'].mean())

#Filling null values in Sunshine with it's mean
data['Sunshine'] = data['Sunshine'].fillna(data['Sunshine'].mean())

#Filling null values in WindGustDir with it's mean
data['WindGustDir'] = data['WindGustDir'].fillna(data['WindGustDir'].mode()[0])

#Filling null values in WindGustSpeed with it's mean
data['WindGustSpeed'] = data['WindGustSpeed'].fillna(data['WindGustSpeed'].mean())

#Filling null values in WindDir9am with it's mean
data['WindDir9am'] = data['WindDir9am'].fillna(data['WindDir9am'].mode()[0])

#Filling null values in WindDir3pm with it's mean
data['WindDir3pm'] = data['WindDir3pm'].fillna(data['WindDir3pm'].mode()[0])

#Filling null values in WindSpeed9am with it's mean
data['WindSpeed9am'] = data['WindSpeed9am'].fillna(data['WindSpeed9am'].mean())

#Filling null values in WindSpeed3pm with it's mean
data['WindSpeed3pm'] = data['WindSpeed3pm'].fillna(data['WindSpeed3pm'].mean())

#Filling null values in Humidity9am with it's mean
data['Humidity9am'] = data['Humidity9am'].fillna(data['Humidity9am'].mean())

#Filling null values in Humidity3pm with it's mean
data['Humidity3pm'] = data['Humidity3pm'].fillna(data['Humidity3pm'].mean())

#Filling null values in Pressure9am with it's mean
data['Pressure9am'] = data['Pressure9am'].fillna(data['Pressure9am'].mean())

#Filling null values in Pressure3pm with it's mean
data['Pressure3pm'] = data['Pressure3pm'].fillna(data['Pressure3pm'].mean())

#Filling null values in Cloud9am with it's mean
data['Cloud9am'] = data['Cloud9am'].fillna(data['Cloud9am'].mean())

#Filling null values in Cloud3pm with it's mean
data['Cloud3pm'] = data['Cloud3pm'].fillna(data['Cloud3pm'].mean())
```

```
#Filling null values in Temp9am with it's mean
data['Temp9am'] = data['Temp9am'].fillna(data['Temp9am'].mean())

#Filling null values in Temp3pm with it's mean
data['Temp3pm'] = data['Temp3pm'].fillna(data['Temp3pm'].mean())

#Filling null values in RainToday with it's mean
data['RainToday'] = data['RainToday'].fillna(data['RainToday'].mode()[0])

#Filling null values in RainTomorrow with it's mean
data['RainTomorrow'] = data['RainTomorrow'].fillna(data['RainTomorrow'].mode()[0])
```

```
#lets Convert in Date Time
data['Date'] = pd.to_datetime(data['Date'])
```

```
# Extracting Day from Date Column
data['Day'] = pd.to_datetime(data.Date, format="%d/%m/%Y").dt.day

# Extracting Month from Date Column
data['Month'] = pd.to_datetime(data.Date, format="%d/%m/%Y").dt.month

# Extracting Year from Date Column
data['Year'] = pd.to_datetime(data.Date, format="%d/%m/%Y").dt.year
```

```
#Lets Delete DATE Columns
data.drop(columns='Date',inplace=True)
```

Identifying the Outliers

```
# checking for numerical columns
num_col=[]
for i in data.dtypes.index:
    if data.dtypes[i]!='object':
        num_col.append(i)
print(num_col)
```

```
['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm', 'Day', 'Month', 'Year']
```

```
# Identifying the outliers using boxplot
plt.figure(figsize=(15,10),facecolor='white')
plotnumber=1
for column in num_col:
    if plotnumber<=20:
        ax=plt.subplot(3,7,plotnumber)
        sns.boxplot(data[column],color="green")
        plt.xlabel(column,fontsize=15)
    plotnumber+=1
plt.tight_layout()
```

Removing Outliers

1. ZScore

```
from scipy.stats import zscore
feature = data[['MaxTemp','Rainfall','Evaporation','Sunshine','WindGustSpeed','WindSpeed9am','WindSpeed3pm','Humidity9am','Pressure9am','Press

z=np.abs(zscore(feature))
# Creating new dataframe
df_zscore = data[(z<3).all(axis=1)]
```

2. IQR (Inter Quantile Range) method

```
Q1 = feature.quantile(0.25)
Q3 = feature.quantile(0.75)
IQR = Q3 - Q1

df_iqr = feature[~((feature < (Q1 - 1.5 * IQR)) |(feature > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
df_iqr
```

Encoding the categorical columns using Label Encoding

```
# Encode categorical columns using Label Encoding
from sklearn.preprocessing import LabelEncoder
label_encoders = {}
for column in df_zscore.select_dtypes(include=['object']).columns:
    label_encoders[column] = LabelEncoder()
    df_zscore[column] = label_encoders[column].fit_transform(df_zscore[column])
```

```
df_zscore
```

Model Preparation

Problem Statement: a) Design a predictive model with the use of machine learning algorithms to forecast whether or not it will rain tomorrow.

Checking for skewness

```
df_zscore.skew()
```

Removing Skewness

```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
```

```
pt.fit_transform(x)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scaler = scaler.fit_transform(x)
```

```
x_scaler
```

Checking VIF- Variance Inflation Factor values

```
#Lets check VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif['Score'] = [variance_inflation_factor(x_scaler,i) for i in range(x_scaler.shape[1])]
vif['features'] = x.columns
vif
```

Selecting Kbest Features

```
from sklearn.feature_selection import SelectKBest, f_classif
```

```
bestfeat = SelectKBest(score_func = f_classif, k = 'all')
fit = bestfeat.fit(x_scaler,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
```

```
fit = bestfeat.fit(x_scaler,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
dfcolumns.head()
featureScores = pd.concat([dfcolumns,dfscores],axis = 1)
featureScores.columns = ['Feature', 'Score']
print(featureScores.nlargest(35,'Score'))
```

SMOTE

```
y.value_counts()
```

```
RainTomorrow
0    6165
1    1822
Name: count, dtype: int64
```

We can see that there is a significant difference in the classes hence lets fix this using SMOTE to avoid any bias.

```
from imblearn.over_sampling import SMOTE
sm = SMOTE()
```

```
x_scaler, y = sm.fit_resample(x_scaler,y)
```

```
y.value_counts()
```

Train Test Split

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

1. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
```

```
x_train, x_test, y_train, y_test = train_test_split(x_scaler, y, test_size=0.2, random_state=723)
```

```
x_train.shape
```

```
(9864, 24)
```

```
y_train.shape
```

```
(9864,)
```

```
x_test.shape
```

```
(2466, 24)
```

```
y_test.shape
```

```
(2466,)
```

```
lr = LogisticRegression()
```

```
lr.fit(x_train,y_train)
```

LogisticRegression

LogisticRegression()

```
y_pred = lr.predict(x_test)
y_pred
```

```
array([0, 0, 1, ..., 0, 1, 0])
```

```
accuracy = accuracy_score(y_test,y_pred)
print(accuracy)
```

```
0.7899432278994323
```

```
#Confusion Matrix
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

```
array([[1001,  217],
       [ 301,  947]])
```

2. Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
```

```
cnn=DecisionTreeClassifier()
```

```
x_train, x_test, y_train, y_test = train_test_split(x_scaler, y, test_size=0.2, random_state=431)
```

```
#Train the model
cnn.fit(x_train,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
def metrics_score(cnn,x_train,x_test,y_train,y_test,train=True):
    if train:
        y_pred=cnn.predict(x_train)
        print("==================Training Score================")
        print("Accuracy Score == >  ", accuracy_score(y_train,y_pred)*100)
    elif train==False:
        pred=cnn.predict(x_test)
        print("=================Test Score====================")
        print("Accuracy Score===>  ",accuracy_score(y_test,pred)*100)
        print("Classification Report==>",classification_report(y_test,pred))
```

```
#Call the function and pass dataset to check train and test score
metrics_score(cnn,x_train,x_test,y_train,y_test,train=True)
metrics_score(cnn,x_train,x_test,y_train,y_test,train=False)
```

```
==================Training Score=================
Accuracy Score == >   100.0
=================Test Score=====================
Accuracy Score===>   87.79399837793999
Classification Report==>               precision    recall  f1-score   support

           0       0.88      0.88      0.88      1243
           1       0.88      0.87      0.88      1223

    accuracy                           0.88      2466
   macro avg       0.88      0.88      0.88      2466
weighted avg       0.88      0.88      0.88      2466
```

Overfitting because we can see that training score is 100%.

3. Random Forest

```
from sklearn.ensemble import RandomForestClassifier
```

```
rfc = RandomForestClassifier(n_estimators=10, random_state=0)
```

```
x_train, x_test, y_train, y_test = train_test_split(x_scaler, y, test_size=0.2, random_state=752)
```

```
rfc.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=10, random_state=0)
```

```
def metrics_score(rfc,x_train,x_test,y_train,y_test,train=True):
    if train:
        y_pred=rfc.predict(x_train)
        print("==================Training Score================")
        print("Accuracy Score == >  ", accuracy_score(y_train,y_pred)*100)
    elif train==False:
        pred=rfc.predict(x_test)
        print("=================Test Score====================")
        print("Accuracy Score===>  ",accuracy_score(y_test,pred)*100)
        print("Classification Report==>",classification_report(y_test,pred))
```

```
#Call the function and pass dataset to check train and test score
metrics_score(rfc,x_train,x_test,y_train,y_test,train=True)
metrics_score(rfc,x_train,x_test,y_train,y_test,train=False)
```

```
==================Training Score=================
Accuracy Score == >   99.72627737226277
=================Test Score=====================
Accuracy Score===>   91.93025141930251
Classification Report==>               precision    recall  f1-score   support

           0       0.90      0.95      0.92      1227
           1       0.95      0.89      0.92      1239

    accuracy                           0.92      2466
   macro avg       0.92      0.92      0.92      2466
weighted avg       0.92      0.92      0.92      2466
```

Problem Statement b) Design a predictive model with the use of machine learning algorithms to predict how much rainfall could be there.

```
df_zscore.head()
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pm	Hu
0	1	13.4	22.9	0.6	5.389395	7.632205	13	44.0	13	14	20.0	24.0	
1	1	7.4	25.1	0.0	5.389395	7.632205	14	44.0	6	15	4.0	22.0	
2	1	12.9	25.7	0.0	5.389395	7.632205	15	46.0	13	15	19.0	26.0	
3	1	9.2	28.0	0.0	5.389395	7.632205	4	24.0	9	0	11.0	9.0	
4	1	17.5	32.3	1.0	5.389395	7.632205	13	41.0	1	7	7.0	20.0	

```
#Dividing data in feature and Label

x = df_zscore.drop(columns=['Rainfall'],axis=1)
y = df_zscore['Rainfall']
```

Feature Scaling

```
x_scaler = scaler.fit_transform(x)
```

```
x_scaler
```

```
array([[-1.50842735,  0.05644354, -0.14573194, ..., -1.6716567 ,
         1.63411265, -1.66131549],
       [-1.50842735, -1.05847646,  0.22067594, ..., -1.55797642,
         1.63411265, -1.66131549],
       [-1.50842735, -0.03646646,  0.32060536, ..., -1.44429613,
         1.63411265, -1.66131549],
       ...,
       [ 1.02942282, -1.43011646,  0.5204642 , ...,  0.82930956,
        -0.12372583,  2.00413391],
       [ 1.02942282, -0.98414846,  0.5371191 , ...,  0.94298985,
        -0.12372583,  2.00413391],
       [ 1.02942282,  0.33517354,  0.01415115, ...,  1.05667013,
        -0.12372583,  2.00413391]])
```

Checking VIF- Variance Inflation Factor values

```
#Lets check VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif['Score'] = [variance_inflation_factor(x_scaler,i) for i in range(x_scaler.shape[1])]
vif['features'] = x.columns
vif
```

Selecting Kbest Features

```
bestfeat = SelectKBest(score_func = f_classif, k = 'all')
fit = bestfeat.fit(x_scaler,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
```

```
fit = bestfeat.fit(x_scaler,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
dfcolumns.head()
featureScores = pd.concat([dfcolumns,dfscores],axis = 1)
featureScores.columns = ['Feature', 'Score']
print(featureScores.nlargest(35,'Score'))
```

1. Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

```
lr = LinearRegression()
```

```
x_train, x_test, y_train, y_test = train_test_split(x_scaler, y, test_size=0.2, random_state=325)
```

```
lr.fit(x_train, y_train)
```

```
LinearRegression()
```

```
y_pred = lr.predict(x_test)
y_pred
```

```
array([-0.13431307,  0.33774901, -0.29034549, ..., -0.24900283,
        0.0388825 ,  1.49370753])
```

```
# Accuracy score
print("Model Training Score : ",lr.score(x_train,y_train))
print("Model Testing Score : ",lr.score(x_test,y_test))
```

```
Model Training Score :  0.4763765269702762
Model Testing Score :  0.5459106674225596
```

```
#MAE
mean_absolute_error(y_test,y_pred)
```

```
1.5453186068163218
```

```
#MSE
mean_squared_error(y_test,y_pred)
```

```
10.17025432501719
```

```
#RMSE
np.sqrt(mean_squared_error(y_test,y_pred))
```

```
3.1890836183796107
```

2. DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor
```

```
decision_tree = DecisionTreeRegressor(random_state=42)
```

```
decision_tree.fit(x_train,y_train)
```

```
DecisionTreeRegressor(random_state=42)
```

```
y_pred_dt = decision_tree.predict(x_test)
print(f'Decision Tree Regressor R^2 Score: {r2_score(y_test, y_pred_dt)}')
```

```
Decision Tree Regressor R^2 Score: 0.47894100092343417
```

```
# Calculate evaluation metrics for Decision Tree Regressor
mae_dt = mean_absolute_error(y_test, y_pred_dt)
mse_dt = mean_squared_error(y_test, y_pred_dt)
print(f'mae_dt: {mae_dt}')
print(f'mse_dt: {mse_dt}')
```

```
mae_dt: 1.0538842077337052
mse_dt: 11.670176237940645
```

3. RandomForestRegressor

[160]:
```
from sklearn.ensemble import RandomForestRegressor
```

[161]:
```
rf_model = RandomForestRegressor(n_estimators=10, random_state=42)
```

[162]:
```
rf_model.fit(x_train, y_train)
```

[162]:
```
RandomForestRegressor
RandomForestRegressor(n_estimators=10, random_state=42)
```

[163]:
```
y_pred_rfc = rf_model.predict(x_test)
print(f'Random Forest Regressor R^2 Score: {r2_score(y_test, y_pred)}')
```

```
Random Forest Regressor R^2 Score: 0.5459106674225596
```

[164]:
```
# Evaluate the model
mae_rf = mean_absolute_error(y_test, y_pred_rfc)
mse_rf = mean_squared_error(y_test, y_pred_rfc)
```

[165]:
```
print(f'mae_rf: {mae_rf}')
print(f'mse_rf: {mse_rf}')
```

```
mae_rf: 1.0655688693028778
mse_rf: 7.670633432797889
```

Hyperparameter grid for Random Forest tuning

[167]:
```
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}
```

[168]:
```
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=param_grid_rf, cv=5, n_jobs=-1)
```

[169]:
```
# Fit the model to the training data
grid_search_rf.fit(x_train, y_train)
```

[169]:
```
GridSearchCV
best_estimator_: RandomForestRegressor
RandomForestRegressor
```

[170]:
```
# Retrieve the best parameters and the best model
best_rf = grid_search_rf.best_estimator_
best_rf_params = grid_search_rf.best_params_
```

[171]:
```
print(best_rf_params)
```

```
{'bootstrap': True, 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
```

[172]:
```
print(best_rf)
```

```
RandomForestRegressor(n_estimators=300, random_state=42)
```

[173]:
```
rf_model_hyper = RandomForestRegressor(bootstrap = True, max_depth=None, min_samples_leaf = 1, min_samples_split= 2, n_estimators = 300)
```

[174]:
```
rf_model_hyper.fit(x_train, y_train)
```

[174]:
```
RandomForestRegressor
RandomForestRegressor(n_estimators=300)
```

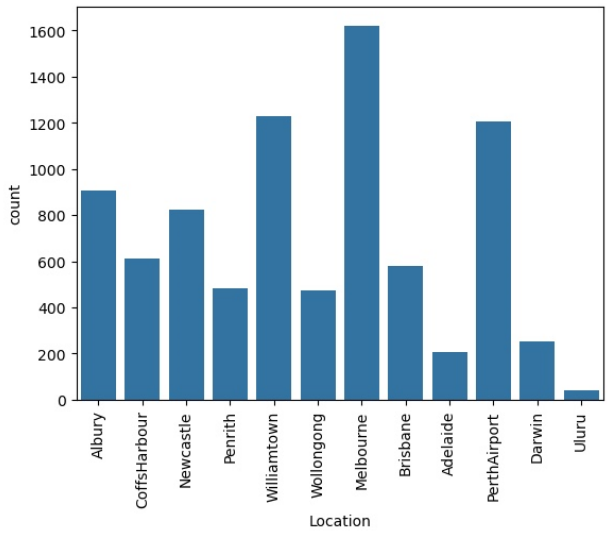
[175]:
```
# Predict using the best Random Forest model
y_pred_rfh = rf_model_hyper.predict(x_test)
```

[184]:
```
# Evaluate the model
mae_rfh = mean_absolute_error(y_test, y_pred_rfh)
mse_rfh = mean_squared_error(y_test, y_pred_rfh)
r2_rfh = r2_score(y_test, y_pred_rfh)
```

[185]:
```
# Output the evaluation results and best hyperparameters
print(f"MAE: {mae_rfh}")
print(f"MSE: {mse_rfh}")
print(f"R-squared: {r2_rfh}")
```

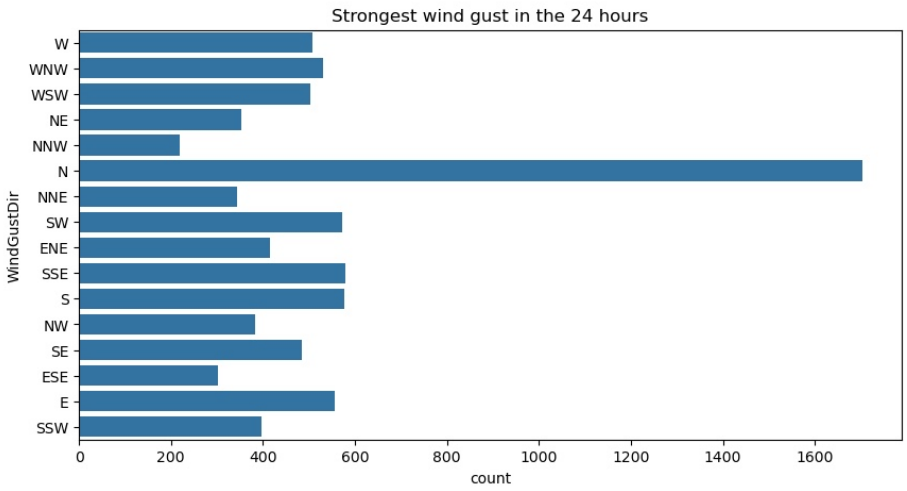
```
MAE: 0.9977822065557918
MSE: 6.731882204593191
R-squared: 0.6994297487965225
```

DATA VISUALIZATION

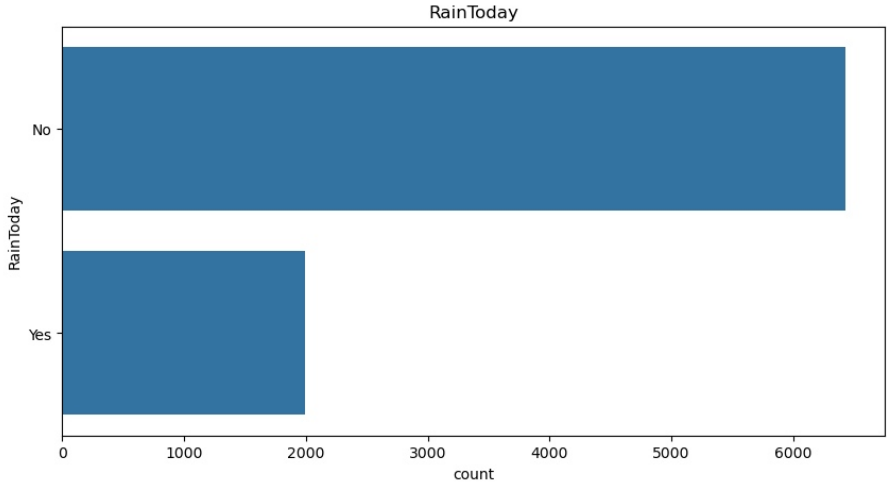



```
Location
Melbourne       1622
Williamtown     1230
PerthAirport    1204
Albury           907
Newcastle        822
CoffsHarbour     611
Brisbane         579
Penrith          482
Wollongong       474
Darwin           250
Adelaide         205
Uluru             39
Name: count, dtype: int64
```

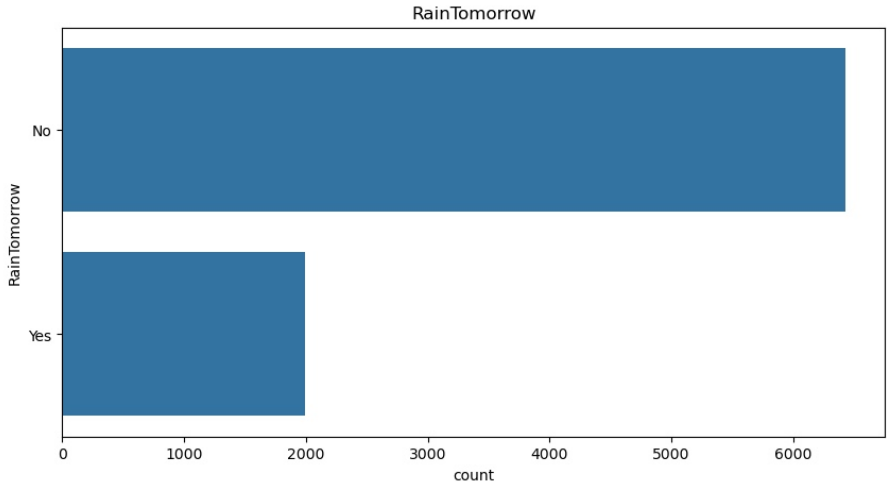
We have the highest rainfall data from Melbourbne and least from Uluru.



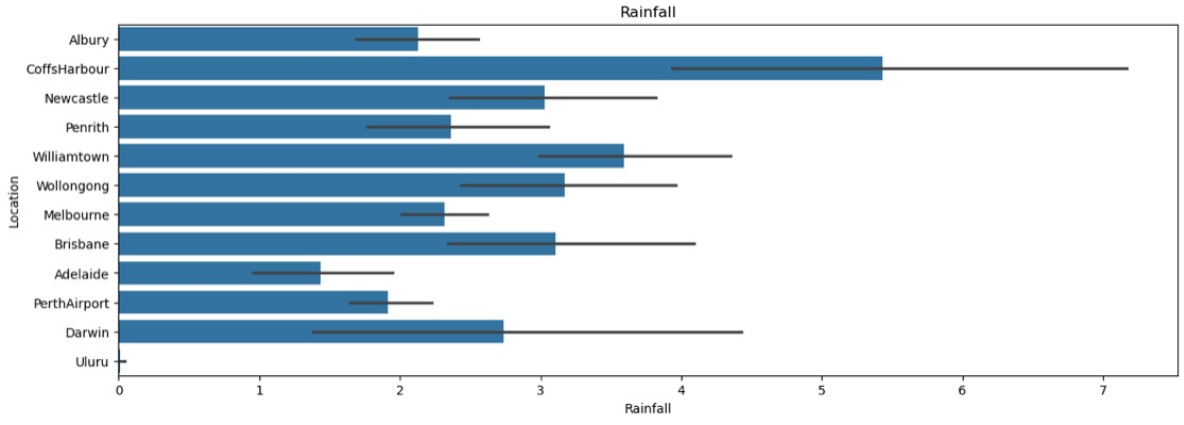
We can clearly see that the wind gust was strongest towards the north.




```
RainToday
No      6435
Yes     1990
```




```
RainTomorrow
No      6434
Yes     1991
```



RESULT

The result section presents the findings from the model evaluation, the performance of the predictive models.

Model Performance:

Problem Statement:

1. **Design a predictive model with the use of machine learning algorithm to forecast whether or not it will rain tomorrow.**
    a. **Logistic Regression:**
        i. **Accuracy Score: 0.789**
    b. **Decision Tree Classifier:**
        i. **Accuracy Score: 0.75**
        ii. **Precision: 0.71**
        iii. **Recall: 0.84**
        iv. **F1-Score: 0.77**
    c. **Random Forest Classifier:**
        i. **Accuracy Score: 0.92**
        ii. **Precision: 0.90**
        iii. **Recall: 0.95**
        iv. **F1-Score: 0.92**

Comparison of Models:

Comparing the performance of Logistic Regression, Decision Tree Classifier and Random Forest Classifier it was observed that Random Forest Classifier has accuracy score of 0.92 which is better than Decision Tree Classifier followed by Logistic Regression.

Problem Statement:

2. **Design a predictive model with the use of machine learning algorithm to predict how much rainfall could be there.**
    a. **Linear Regression:**
        i. **R-Squared: 0.5459**
        ii. **MAE: 1.54**
        iii. **MSE: 10.17**
        iv. **RMSE: 3.18**
    b. **Decision Tree Regressor:**
        i. **R-Squared: 0.54**
        ii. **MAE: 1.05**
        iii. **MSE: 11.67**
    c. **Random Forest Regressor:**
        i. **R-Squared: 0.6994**

ii. **MAE: 0.99**
iii. **MSE: 6.73**

Comparison of Models:

Comparing the performance of Linear Regression, Decision Tree Regressor and Random Forest Regressor it was observed that Random Forest Regressor has highest r-squared score 0.6994.

CONCLUSION

The weather prediction model developed using the "weatherAUS.csv" dataset effectively predicts both the likelihood of rain and the expected rainfall amount. By handling data preprocessing, outlier removal, feature engineering, and model selection, the model achieves robust performance. The use of hyperparameter tuning further enhances the model's accuracy, ensuring reliable predictions. This approach not only addresses the given problem statements but also provides a comprehensive framework for developing predictive models in similar domains.