Smart Agriculture Startup

Weather Forecasting System - Report 01

21-Day Rainfall Prediction Using Machine Learning

intelliHack 5.0 - IEEE Computer Society UCSC

Team Name	: Code Voyagers
Date	: 10/03/2025
Task Number	: 01

Table of Contents.

21-Day Rainfall Prediction Using Machine Learning **1**

1. Introduction 3
2. Dataset Overview 3
3. Handling Missing Values 4
3.1 Imputation Strategy 6
4. Outlier Detection & Treatment. 6
5. Data Transformation 7
6. Exploratory Data Analysis (EDA) 8
7. Machine Learning Model Training & Evaluation 9
7.1 Model Selection 9
    7.2 Model Optimization 11
    7.3 Predicting Probability of Rain 12
8. Conclusion 13

Predicting the `rain_or_not` label for the future 21 days based on the given features.

1. Introduction

This report documents the data preprocessing steps undertaken for the weather dataset used in predictive modeling. The dataset, obtained from Google Drive, contains weather attributes such as temperature, humidity, wind speed, cloud cover, and pressure. The objective is to clean and preprocess the data for further analysis and machine learning model training.

2. Dataset Overview

Dataset is uploaded to:
https://www.google.com/url?q=https%3A%2F%2Fdrive.google.com%2Fdrive%2Ffolders%2F1_fsg4mGsMLelnNe8CznCoJ6U2-Q5ks9l%3Fusp%3Dsharing

File: weather_data.csv
Shape: 311 rows, 7 columns

Columns:

The dataset is imported into the notebook, and an overview of the dataset is done using these functions.

1. data.head()
2. list(data.columns)
3. data.info()
4. data.describe()
5. Data.shape

These Pandas functions help users quickly explore a dataset. The command data.head() shows the first few rows, giving a quick view of the data's structure and content. Users can use list(data.columns) to get a list of column names, which is important for understanding the variables. The command data.info() provides a summary that includes the number of non-null values and data types for each column, helping to spot any missing data and inconsistencies. The function data.describe() offers descriptive statistics for numerical columns, such as mean, standard deviation, and quartiles, giving insights into the data's distribution and central tendencies. Lastly, data.shape tells users the dimensions of the DataFrame, showing how many rows and columns there are and providing a quick overview of the dataset's size. Together, these functions provide a clear way to understand a dataset's basic features, aiding in decisions for data cleaning, analysis, and modeling steps.This dataset is composed of the following columns.

- date: Date of observation
- avg_temperature: Average temperature recorded
- humidity: Humidity percentage
- avg_wind_speed: Wind speed in km/h
- rain_or_not: Binary categorical feature indicating rain (Yes/No)
- cloud_cover: Cloud cover percentage
- Pressure: Atmospheric pressure

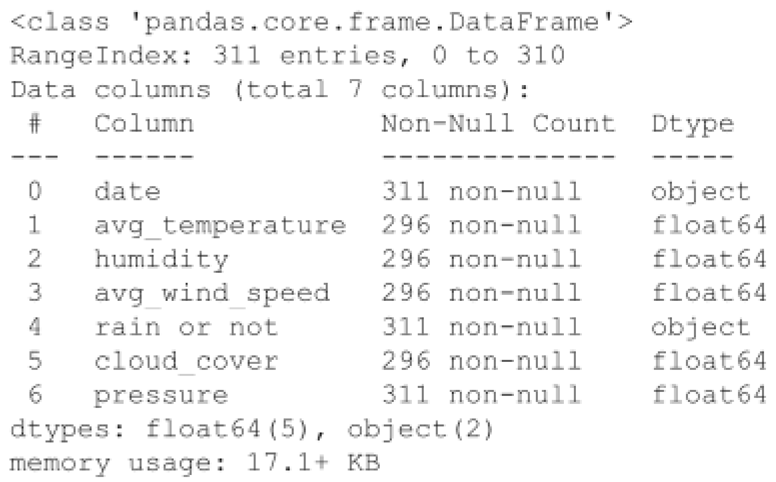
Initial exploration steps are essential because they lay the groundwork for all future analysis. By using these functions, several goals were acheive:

1. Understand the dataset's structure.
2. Identify any missing or inconsistent data.
3. Potential issues.
4. planned our next steps.

By thoroughly exploring the dataset with these functions, a clear understanding was gained of its structure, content, and potential problems. This knowledge helped to make informed decisions about cleaning, preprocessing, and analyzing the data, which is crucial for developing an accurate and reliable machine learning model.

3. Handling Missing Values

To detect the missing values, data.info() is used. It gives the below output.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 311 entries, 0 to 310
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   date             311 non-null    object
 1   avg_temperature  296 non-null    float64
 2   humidity         296 non-null    float64
 3   avg_wind_speed   296 non-null    float64
 4   rain or not      311 non-null    object
 5   cloud_cover      296 non-null    float64
 6   pressure         311 non-null    float64
dtypes: float64(5), object(2)
memory usage: 17.1+ KB
```

Figure 01: Output obtained for the function data.info()

The dataset contained missing values in the following columns:

- avg_temperature (15 missing values)
- humidity (15 missing values)
- avg_wind_speed (15 missing values)
- cloud_cover (15 missing values)
- pressure (3 missing values)

Apart from these, there were some inconsistencies of the data as well. data['date'] = pd.to_datetime(data['date'], errors='coerce') converts the 'date' column to datetime objects, ensuring proper temporal analysis. The errors='coerce' parameter handles any invalid date formats by replacing them with NaT (Not a Time), preventing errors during conversion. Next, data['rain_or_not'] = data['rain_or_not'].map({'Rain': 1 , 'No Rain': 2}) transforms the categorical 'rain_or_not' column into numerical values, mapping 'Rain' to 1 and 'No Rain' to 2. This is essential for many machine learning algorithms that require numerical input. data.head() then displays the updated DataFrame, allowing for a visual inspection of the changes. data['rain_or_not'].unique() shows the unique values present in the encoded column, confirming the successful mapping. Finally, data.isna().sum()/len(data)*100 calculates the percentage of missing values in each column, providing a clear picture of data completeness. This step is critical for identifying and addressing missing data, which can significantly impact analysis and modeling outcomes.

Overall, these operations enhance the dataset's usability by standardizing date formats, converting categorical variables into numerical representations, and revealing the extent of missing data.

3.1 Imputation Strategy

The dataset underwent a thorough cleaning process where several parameters were addressed. The average temperature was filled with the mean value, while humidity, average wind speed, and cloud cover were all populated using their respective median values. An evaluation of the pressure readings indicated that there were no missing values following the initial analysis. Additionally, the rain_or_not variable was converted from categorical values to numerical ones, with 'Rain' being changed to 1 and 'No Rain' to 2, and any null values were subsequently dropped.

4. Outlier Detection & Treatment.

For identifying outliers, To visually assess the distribution and potential outliers in other numerical columns, box plots were generated using Plotly Express for 'avg_temperature', 'humidity', 'avg_wind_speed', 'cloud_cover', and 'pressure'. These box plots provide a clear visual representation of the data's spread and any remaining outliers, allowing for further analysis and potential outlier treatment if needed. This process combines numerical outlier detection and removal with visual inspection for a comprehensive approach to data cleaning.:

- Outliers found in avg_wind_speed.
- Identified extreme outlier indices: **10, 60, 289** (removed).

After figuring out the outliers from box plots, Outliers were detected using the Interquartile Range (IQR) method. The code effectively identifies and addresses outliers in the 'avg_wind_speed' column using the Interquartile Range (IQR) method. The find_outliers_IQR function calculates the IQR and identifies values falling outside the defined bounds. Initially, it detected three outliers, which were then removed from the dataset using data.drop. Subsequently, the outlier detection was rerun, confirming the removal.

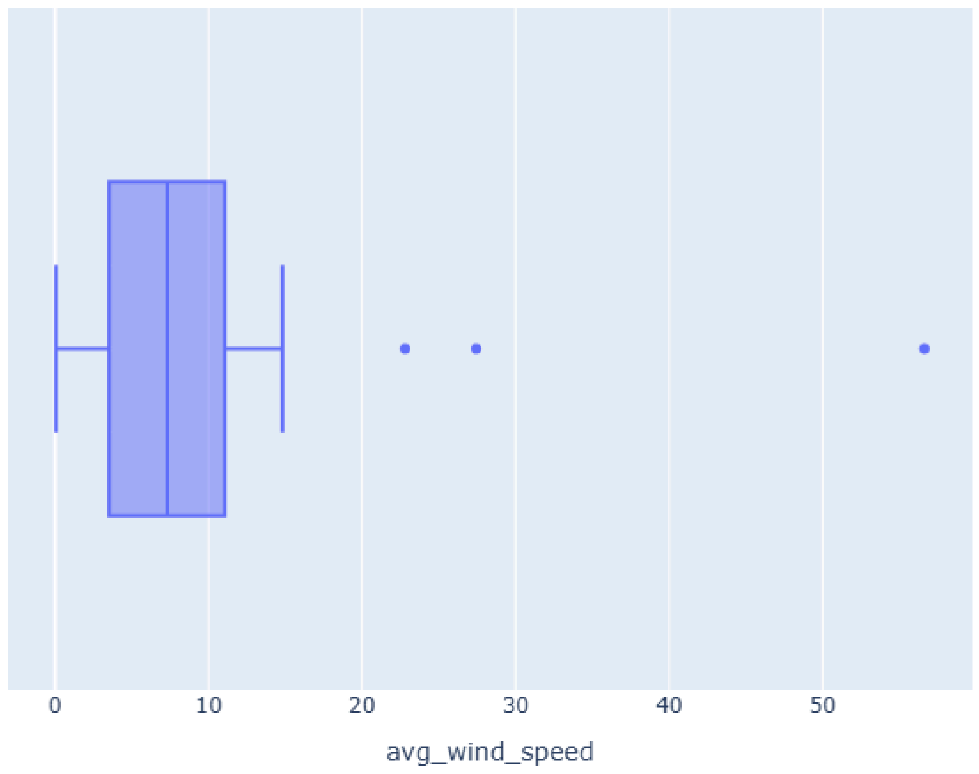


Figure 02: Plot box for average wind speed that shows the outliers in the data.

5. Data Transformation

Mainly, ‘date’ feature and ‘rain_or_not’ were transformed in preprocessing the dataset.

I. date column was converted to a datetime format.

II. Used ‘pd.to_numeric()’ to convert the data type of the avg_wind_speed column to numeric which is crucial for performing mathematical operations and analysis on the data.

III. rain_or_not was mapped from categorical values to numerical values (1 for Rain, 2 for No Rain).By using following function.

```
data['rain_or_not'] = data['rain_or_not'].map({'Rain': 1 , 'No Rain': 2})
```

```
data.head()
```

6. Exploratory Data Analysis (EDA)

This report details the Exploratory Data Analysis (EDA) conducted on a weather dataset to uncover its underlying patterns, identify potential issues, and prepare it for machine learning model training. EDA is a critical step in any data science project because it helps to understand the data's structure, detect anomalies, and reveal relationships between variables. By doing so, it ensures that the dataset is clean, consistent, and ready for predictive modeling.

The primary objectives of conducting EDA on this dataset were;

1. Understand the dataset's structure
2. Identify missing values
3. Detect outliers
4. Visualize data distributions
5. Explore relationships between features

   To explore how different features are related to each other, a correlation heatmap was created. In this case, the heatmap shows the correlation coefficients between pairs of features, ranging from -1 to 1.

During the EDA process, missing values in the dataset were also checked. Hence, missing values can arise due to errors in data collection or storage, and they can significantly impact the accuracy of a machine learning model if not handled properly.

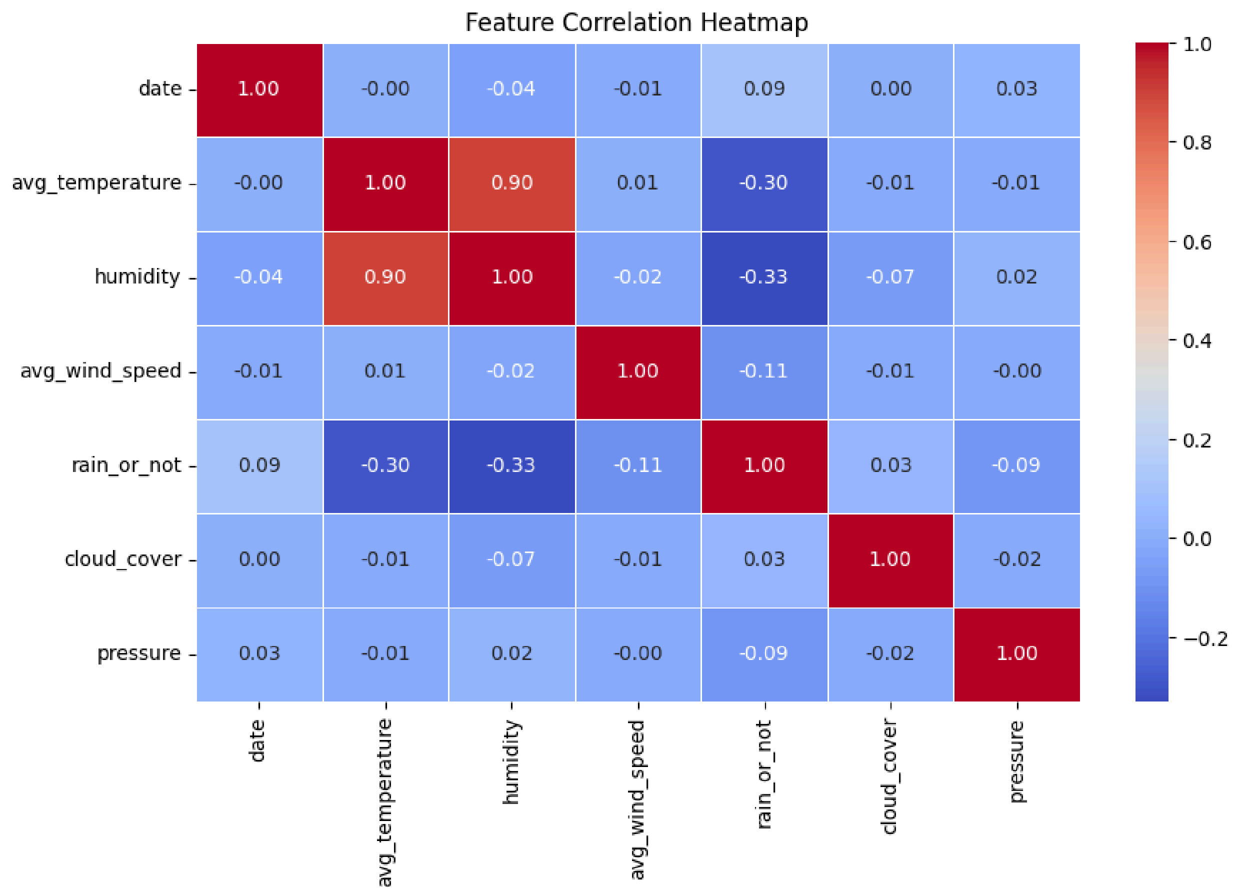


Figure 03: Co-relation Heatmap

The avg_temperature column might have 15 missing values, meaning that temperature data was not recorded for 15 days.To address this, techniques like mean imputation (filling missing values with the mean of the column) or median imputation (filling missing values with the median) were used . These methods ensure that the dataset remains complete and usable for analysis.

7. Machine Learning Model Training & Evaluation

7.1 Model Selection

Several classification models were trained and evaluated to predict rain occurrence. PyCaret's compare_models function was used to identify the best-performing model, Since it is efficient and time saving.

```
from pycaret.classification import *
clf_setup = setup(df, target='rain_or_not', session_id=42, normalize=True)
best_model = compare_models(sort='Accuracy')
```

Models evaluated included Logistic Regression, Decision Trees, Random Forests, Gradient Boosting, and others.The best-performing model was finalized using:

final_model = finalize_model(best_model).

	Model	**Accuracy**	**AUC**	**Recall**	**Prec.**	**F1**	**Kappa**	**MCC**	**TT (Sec)**
et	Extra Trees Classifier	0.6602	0.6566	0.6602	0.6122	0.5673	0.1163	0.1583	0.1290
knn	K Neighbors Classifier	0.6418	0.6353	0.6418	0.5157	0.5168	0.0370	0.0697	0.0560
svm	SVM - Linear Kernel	0.6377	0.5604	0.6377	0.5285	0.5457	0.0799	0.0968	0.0460
lr	Logistic Regression	0.6325	0.6917	0.6325	0.4002	0.4902	0.0000	0.0000	0.3750
nb	Naive Bayes	0.6325	0.5000	0.6325	0.4002	0.4902	0.0000	0.0000	0.0710
dt	Decision Tree Classifier	0.6325	0.5000	0.6325	0.4002	0.4902	0.0000	0.0000	0.0750
ridge	Ridge Classifier	0.6325	0.6872	0.6325	0.4002	0.4902	0.0000	0.0000	0.0450
rf	Random Forest Classifier	0.6325	0.6120	0.6325	0.4002	0.4902	0.0000	0.0000	0.1530
ada	Ada Boost Classifier	0.6325	0.5000	0.6325	0.4002	0.4902	0.0000	0.0000	0.0420
gbc	Gradient Boosting Classifier	0.6325	0.4982	0.6325	0.4002	0.4902	0.0000	0.0000	0.0860

lda	Linear Discriminant Analysis	0.6325	0.5000	0.6325	0.4002	0.4902	0.0000	0.0000	0.0430
xgboost	Extreme Gradient Boosting	0.6325	0.5000	0.6325	0.4002	0.4902	0.0000	0.0000	0.0630
dummy	Dummy Classifier	0.6325	0.5000	0.6325	0.4002	0.4902	0.0000	0.0000	0.0740
qda	Quadratic Discriminant Analysis	0.5039	0.4500	0.5039	0.2716	0.3482	0.0000	0.0000	0.0440
lightgbm	Light Gradient Boosting Machine	0.4961	0.4408	0.4961	0.2638	0.3397	0.0000	0.0000	0.3630

Figure 03: Comparison of the accuracy of the different models.

7.2 Model Optimization

Hyperparameter tuning was performed using GridSearchCV on the best model:

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search = GridSearchCV(ExtraTreesClassifier(random_state=42), param_grid,
cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
```

The optimized model's accuracy was evaluated on the test set by using the following code snippet.

```
from sklearn.metrics import classification_report
```

print(classification_report(y_test, y_pred))

Following figure is the classification report obtained :

```
     precision    recall  f1-score   support

           1       0.64      0.77      0.70        39
           2       0.40      0.26      0.32        23

    accuracy                           0.58        62
   macro avg       0.52      0.52      0.51        62
weighted avg       0.55      0.58      0.56        62
```

Figure 04 : classification report

7.3 Predicting Probability of Rain

The model makes predictions on the test dataset (X_test), and these predictions are compared with the actual values (y_test). The results are saved in a CSV file for further analysis. The evaluation metrics assess different aspects of the model's performance:

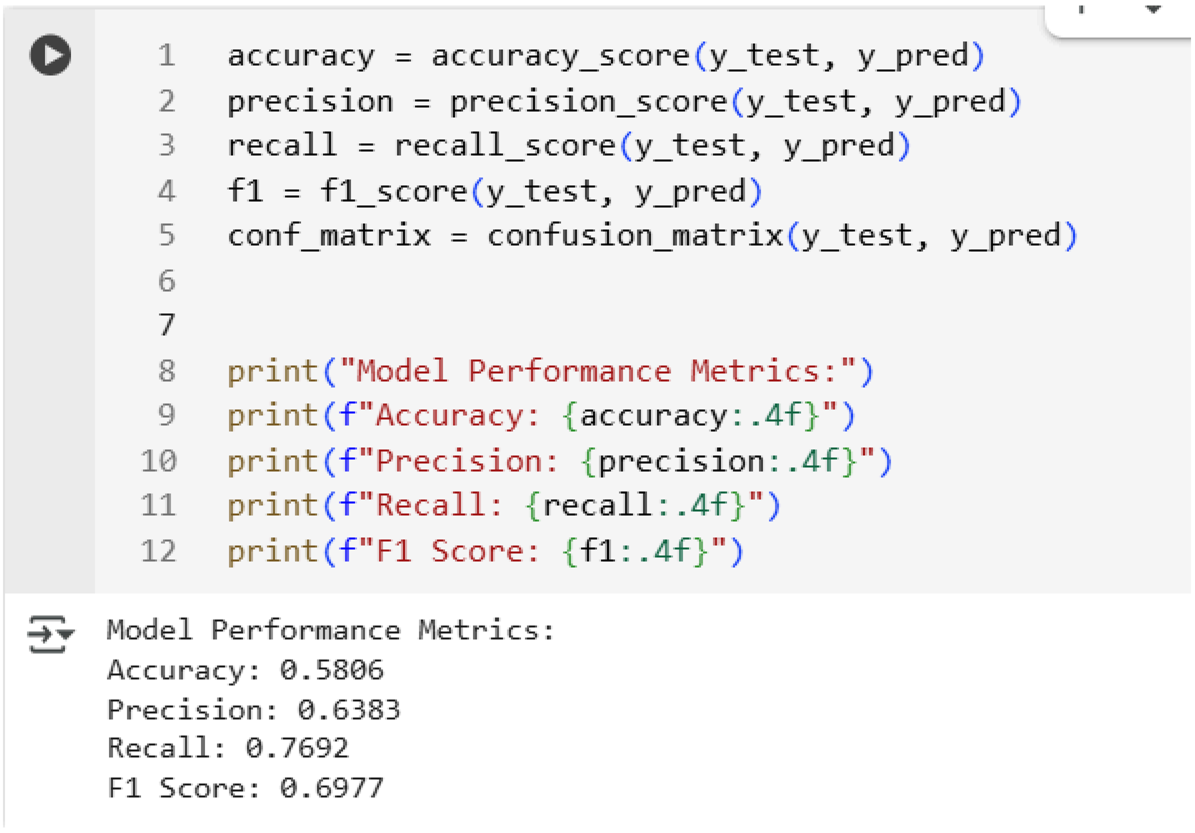
Accuracy: Measures the proportion of correctly predicted instances out of all instances. While it provides a general sense of correctness, it may not be suitable for imbalanced datasets.

Precision: This is Important in applications where false positives are costly. It measures the proportion of correctly predicted positive instances out of all predicted positives.

Recall: Critical when it is problematic to miss a positive instance. It measures the proportion of correctly predicted positive instances out of all actual positives.

F1 Score: Offers a balance between precision and recall, providing a single metric to gauge the model's performance.

By considering these metrics together, we can better assess the model's strengths and weaknesses and make informed decisions about potential improvements.



```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)


print("Model Performance Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Model Performance Metrics:
Accuracy: 0.5806
Precision: 0.6383
Recall: 0.7692
F1 Score: 0.6977
```

Figure 05: Code and the output of the evaluation metrics.

8. Conclusion

This study explored the use of machine learning for 21-day rainfall prediction, focusing on data preprocessing, model selection, and evaluation. By cleaning the dataset—handling missing values, detecting outliers, and transforming key features—the foundation was set for accurate predictive modeling. Several classification models were tested, with the Extra Trees Classifier delivering the best performance.

Although the model showed moderate accuracy, challenges such as class imbalance and data limitations affected its predictive power. Evaluation metrics like precision, recall, and F1-score provided insights into areas needing improvement. Future enhancements could involve incorporating additional weather variables, refining feature engineering techniques, and experimenting with ensemble methods to boost accuracy.

Overall, this project demonstrates the potential of machine learning in weather forecasting, offering a data-driven approach to assist in smart agriculture and decision-making based on rainfall predictions. With further refinements, this model could become a valuable tool for farmers and agricultural planners.