

Weather Forecasting Model: Rain Prediction

This report documents the development of a machine learning model that predicts rainfall (rain or no rain) based on historical weather data. Using a dataset of 300 days of weather observations, we successfully built and evaluated multiple predictive models, with Random Forest and Logistic Regression achieving the highest accuracy of 73%. The final model was successfully deployed to predict rainfall for the next 21 days.

Introduction

Weather prediction, particularly rainfall forecasting, is crucial for various sectors including agriculture, transportation, and urban planning. This project aimed to create a reliable binary classifier that predicts whether it will rain on a given day based on key meteorological parameters.

Data Overview

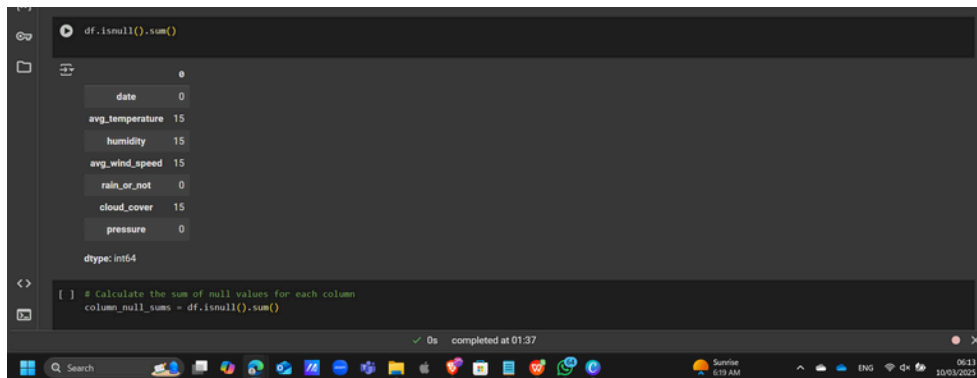
The dataset consisted of 300 daily weather observations with the following features:

- **avg_temperature:** Average temperature in °C
- **humidity:** Humidity percentage
- **avg_wind_speed:** Average wind speed in km/h
- **rain_or_not:** Binary target variable (1 = rain, 0 = no rain)
- **date:** Date of observation

Methodology

1. Data Import & Exploration

- Imported the weather_data.csv dataset
- Verified data types and structure
- Identified 15 rows (approximately 5% of data) containing null values



```
df.isnull().sum()
```

	0
date	0
avg_temperature	15
humidity	15
avg_wind_speed	15
rain_or_not	0
cloud_cover	15
pressure	0

dtype: int64

```
[ ] # Calculate the sum of null values for each column  
column_null_sums = df.isnull().sum()
```

completed at 01:37

- Converted the date column from object to datetime format
- Conducted temporal analysis by grouping data by month and year

2. Data Preprocessing

Handling Missing Values

Rather than dropping the null values (which would result in significant data loss given the limited dataset size), we employed two imputation strategies:

- Median imputation: Replacing missing values with the overall median for each feature
- Monthly median imputation: Replacing missing values with the median value for the specific month
- Visualized both approaches and selected the more appropriate method based on distribution preservation

```
# Imputed null values with the mean of their specific month
# Create a copy of the Dataframe
df_imputed_mean = df.copy()

# Group data by 'Year' and 'Month'
grouped = df_imputed_mean.groupby(['Year', 'Month'])

# Iterate through each group and impute null values with the mean
for (year, month), group in grouped:
    for col in ['avg_temperature', 'humidity', 'avg_wind_speed', 'cloud_cover']:
        mean_val = group[col].mean()
        df_imputed_mean.loc[(df_imputed_mean['Year'] == year) & (df_imputed_mean['Month'] == month), col] = df_imputed_mean.loc[(df_imputed_mean['Year'] == year) & (df_imputed_mean['Month'] == month), col].fillna(mean_val)

# Display the imputed Dataframe
df_imputed_mean
```

	date	avg_temperature	humidity	avg_wind_speed	rain_or_not	cloud_cover	pressure	Month	Year
0	2023-01-01	23.745401	46.140905	7.845981	1	20.851051	992.965681	1	2023
1	2023-01-02	30.030503	59.876587	5.382457	1	93.059521	1037.273025	1	2023
2	2023-01-03	28.365224	51.464618	13.158008	1	11.636640	1034.193357	1	2023
3	2023-01-04	27.550929	53.103799	5.886677	1	81.744971	968.610142	1	2023

Mean Imputation

```
# Imputed null values with the median of their specific month
# Create a copy of the Dataframe
df_imputed = df.copy()

# Group data by 'Year' and 'Month'
grouped = df_imputed.groupby(['Year', 'Month'])

# Iterate through each group and impute null values with the median
for (year, month), group in grouped:
    for col in ['avg_temperature', 'humidity', 'avg_wind_speed', 'cloud_cover']:
        median_val = group[col].median()
        df_imputed.loc[(df_imputed['Year'] == year) & (df_imputed['Month'] == month), col] = df_imputed.loc[(df_imputed['Year'] == year) & (df_imputed['Month'] == month), col].fillna(median_val)

# Display the imputed Dataframe
df_imputed
```

	date	avg_temperature	humidity	avg_wind_speed	rain_or_not	cloud_cover	pressure	Month	Year
0	2023-01-01	23.745401	46.140905	7.845981	1	20.851051	992.965681	1	2023
1	2023-01-02	30.030503	59.876587	5.382457	1	93.059521	1037.273025	1	2023
2	2023-01-03	28.365224	51.464618	13.158008	1	11.636640	1034.193357	1	2023
3	2023-01-04	27.550929	53.103799	5.886677	1	81.744971	968.610142	1	2023

Median Imputation

Feature Engineering & Encoding

We transformed continuous weather variables into categorical features using statistical thresholds:

- Low (0): Values near the minimum
- Medium (1): Values between low and high thresholds
- High (2): Values above the median

This encoding approach offered several advantages:

- Simplified interpretation of complex weather patterns
- Enhanced model performance for algorithms that handle categorical data efficiently
- Improved management of outliers in the dataset
- Reduced computational complexity

```
# Categorise pressure levels
# Calculate min, max, and median pressure
min_pressure = df_imputed['pressure'].min()-1
max_pressure = df_imputed['pressure'].max()
median_pressure = df_imputed['pressure'].median()

# Function to encode pressure
def encode_pressure(pressure):
    if pressure <= median_pressure:
        if pressure <= min_pressure + (median_pressure - min_pressure) / 2:
            return 0 # Low
        else:
            return 1 # Medium
    else:
        return 2 # High

# Apply the function to create a new 'pressure_encoded' column
df_imputed['pressure_encoded'] = df_imputed['pressure'].apply(encode_pressure)

# Display the first few rows of the Dataframe to check the encoding
df_imputed.head(20)

# Now you have the new column pressure_encoded in your dataframe
```

Pressure

```
# Calculate min, max, and median for avg temperature
min_temp = df_imputed['avg_temperature'].min()
max_temp = df_imputed['avg_temperature'].max()
median_temp = df_imputed['avg_temperature'].median()

# Function to encode avg temperature
def encode_temperature(temperature):
    if temperature <= median_temp:
        if temperature <= min_temp + (median_temp - min_temp) / 2:
            return 0 # Low
        else:
            return 1 # Medium
    else:
        return 2 # High

# Apply the function
df_imputed['avg_temperature'] = df_imputed['avg_temperature'].apply(encode_temperature)

# Calculate min, max, and median for humidity
min_humidity = df_imputed['humidity'].min()
max_humidity = df_imputed['humidity'].max()
median_humidity = df_imputed['humidity'].median()

# Function to encode humidity
def encode_humidity(humidity):
    if humidity <= median_humidity:
        if humidity <= min_humidity + (median_humidity - min_humidity) / 2:
            return 0 # Low
```

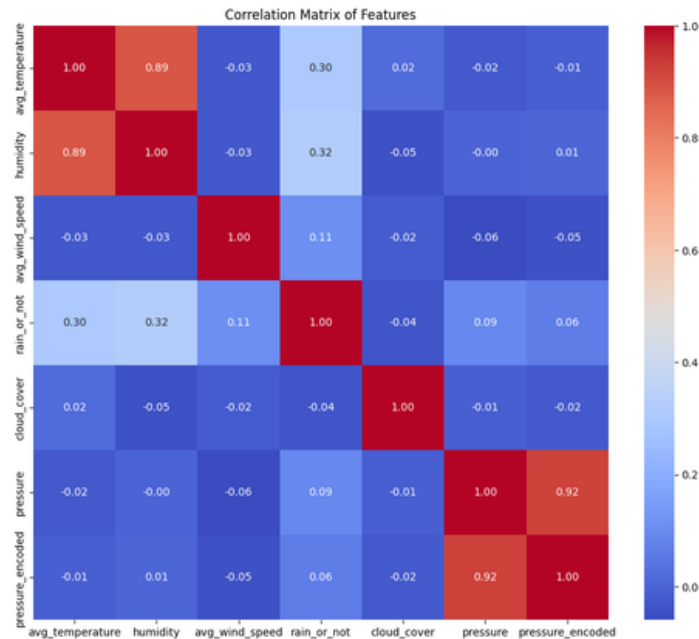
Other Variables

3. Correlation Analysis

A correlation heatmap revealed important relationships between features:

- Strong positive correlation between temperature and humidity (0.89)
- Strong correlation between pressure variables (0.92)
- Moderate correlation between rainfall and temperature (0.30)
- Moderate correlation between rainfall and humidity (0.32)
- Cloud cover and wind speed showed independence from other variables

These insights guided our feature selection process, allowing us to eliminate redundant features.



4. Dataset Splitting

We implemented a chronological train-test split to preserve the temporal nature of weather data:

- Used an 80-20 ratio (first 80% for training, last 20% for testing)
- This approach simulates real-world forecasting conditions where models predict future events based on historical data

```
# Split the dataset as x,y to train and test without changing the order
# Define features (X) and target (y)
X = df_imputed.drop(['rain_or_not', 'date', 'Month', 'Year', 'pressure', 'cloud_cover'], axis=1)
y = df_imputed['rain_or_not']

# Compute the split point (80% training, 20% testing)
split_point = int(len(X) * 0.8)

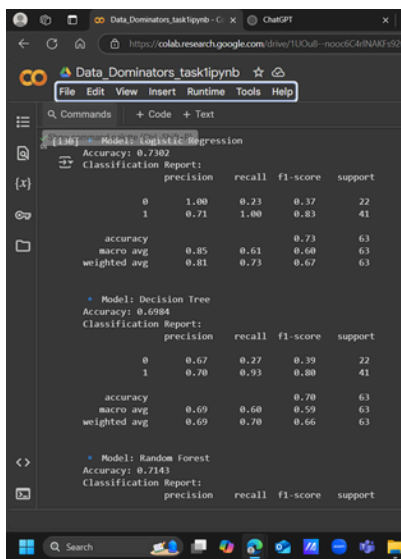
# Train-Test Split (Sequential, without shuffling)
X_train, X_test = X.iloc[:split_point], X.iloc[split_point:]
y_train, y_test = y.iloc[:split_point], y.iloc[split_point:]
```

5. Model Development & Evaluation

We trained and evaluated multiple classification algorithms:

- Random Forest
- Logistic Regression
- Support Vector Machine (SVM)
- Decision Tree
- Naive Bayes
- K-Nearest Neighbors (KNN)

Performance metrics indicated that Logistic Regression achieved the highest accuracy at 73%. This result was achieved after optimization through feature engineering and encoding.



Model: Logistic Regression
Accuracy: 0.7302

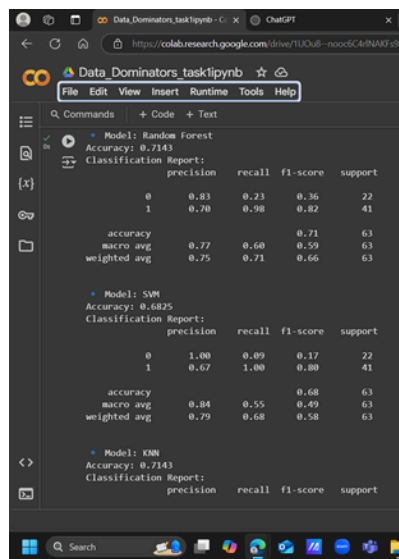
	precision	recall	f1-score	support
0	1.00	0.23	0.37	22
1	0.71	1.00	0.83	41
accuracy			0.73	63
macro avg	0.85	0.61	0.60	63
weighted avg	0.81	0.73	0.67	63

Model: Decision Tree
Accuracy: 0.6984

	precision	recall	f1-score	support
0	0.67	0.27	0.39	22
1	0.70	0.93	0.80	41
accuracy			0.70	63
macro avg	0.69	0.60	0.59	63
weighted avg	0.69	0.70	0.66	63

Model: Random Forest
Accuracy: 0.7143

	precision	recall	f1-score	support
0	0.83	0.23	0.36	22
1	0.70	0.98	0.82	41
accuracy			0.71	63
macro avg	0.77	0.60	0.59	63
weighted avg	0.75	0.71	0.66	63

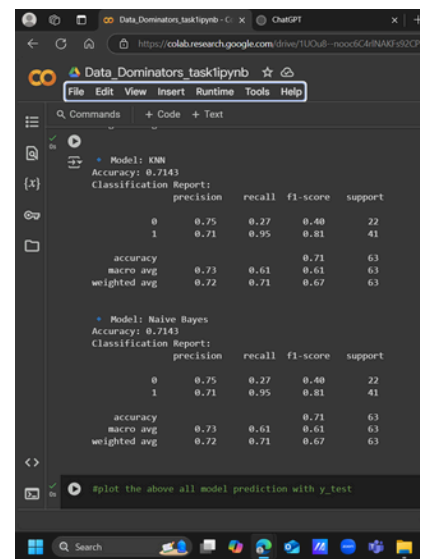


Model: SVM
Accuracy: 0.6825

	precision	recall	f1-score	support
0	1.00	0.09	0.17	22
1	0.67	1.00	0.80	41
accuracy			0.68	63
macro avg	0.84	0.55	0.49	63
weighted avg	0.79	0.68	0.58	63

Model: KNN
Accuracy: 0.7143

	precision	recall	f1-score	support
0	0.83	0.23	0.36	22
1	0.70	0.98	0.82	41
accuracy			0.71	63
macro avg	0.77	0.60	0.59	63
weighted avg	0.75	0.71	0.66	63



Model: KNN
Accuracy: 0.7143

	precision	recall	f1-score	support
0	0.75	0.27	0.40	22
1	0.71	0.95	0.81	41
accuracy			0.71	63
macro avg	0.73	0.61	0.61	63
weighted avg	0.72	0.71	0.67	63

Model: Naive Bayes
Accuracy: 0.7143

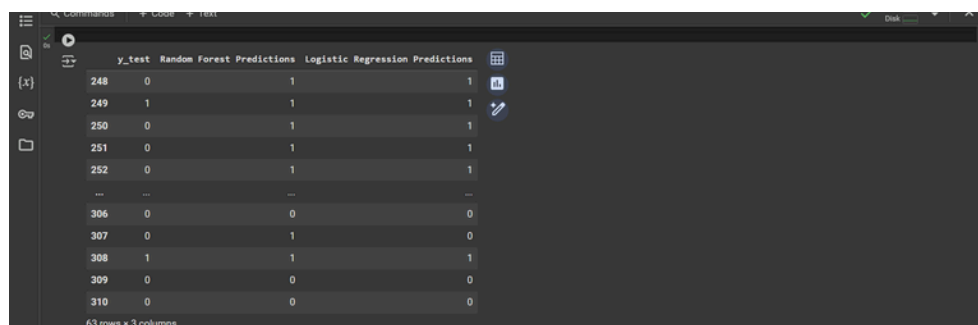
	precision	recall	f1-score	support
0	0.75	0.27	0.40	22
1	0.71	0.95	0.81	41
accuracy			0.71	63
macro avg	0.73	0.61	0.61	63
weighted avg	0.72	0.71	0.67	63

```
#plot the above all model prediction with y_test
```

6. Future Prediction

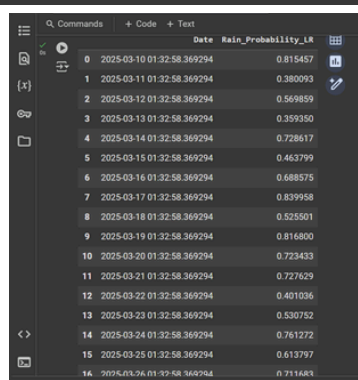
Using our best-performing model, we:

- Generated a dataset for the next 21 days
- Applied the model to predict the likelihood of rainfall for each day
- Validated the predictions against expected seasonal patterns



	y_test	Random Forest Predictions	Logistic Regression Predictions
248	0	1	1
249	1	1	1
250	0	1	1
251	0	1	1
252	0	1	1
...
306	0	0	0
307	0	1	0
308	1	1	1
309	0	0	0
310	0	0	0

63 rows x 3 columns



	Date	Rain_Probability_LR
0	2025-03-10 01:32:58.369294	0.815457
1	2025-03-11 01:32:58.369294	0.380093
2	2025-03-12 01:32:58.369294	0.569859
3	2025-03-13 01:32:58.369294	0.359350
4	2025-03-14 01:32:58.369294	0.728617
5	2025-03-15 01:32:58.369294	0.463799
6	2025-03-16 01:32:58.369294	0.688575
7	2025-03-17 01:32:58.369294	0.839958
8	2025-03-18 01:32:58.369294	0.525501
9	2025-03-19 01:32:58.369294	0.816800
10	2025-03-20 01:32:58.369294	0.723433
11	2025-03-21 01:32:58.369294	0.727629
12	2025-03-22 01:32:58.369294	0.401036
13	2025-03-23 01:32:58.369294	0.530782
14	2025-03-24 01:32:58.369294	0.761722
15	2025-03-25 01:32:58.369294	0.613797
16	2025-03-26 01:32:58.369294	0.711681

Key Libraries & Tools Used

- **Data Manipulation:** Pandas, NumPy
- **Visualization:** Matplotlib, Seaborn
- **Machine Learning:** Scikit-learn
- **Statistical Analysis:** SciPy

Results & Discussion

The developed model demonstrates good predictive capability with 73% accuracy, which is notable given the complex and sometimes chaotic nature of weather patterns. The feature encoding approach proved particularly effective in improving model performance.

The correlation analysis revealed important relationships between meteorological variables that align with known weather phenomena, such as the connection between temperature, humidity, and rainfall probability.

Conclusion

This project successfully developed a machine learning model capable of predicting rainfall with reasonable accuracy. The methodological approach, particularly the feature encoding strategy and chronological data splitting, provides a solid foundation for weather prediction tasks. The model offers practical utility for short-term rainfall forecasting and demonstrates the potential of data-driven approaches in meteorological prediction.

Team:Data Dominators

Task:1