

1. Introduction

This document outlines the development of a machine learning model to analyze and forecast key metrics using a dataset. The report covers problem definition, data exploration, preprocessing, model selection, evaluation, hyperparameter tuning, and potential future improvements.

The CSV dataset for this machine learning model was sourced from GitHub, preprocessed, and used for training and evaluation.

2. Problem Definition, Data Source, and Description

2.1 Problem Definition

This project aims to develop a predictive model using machine learning to analyze and forecast key metrics based on a dataset.

The dataset contains weather records for different cities on various dates. The columns include:

- **City:** The city where the weather data was recorded.
- **Date:** The date of the weather record (format: DD-MM-YYYY).
- **Temperature (°C):** The temperature in degrees Celsius (some missing values).
- **Weather Description:** A short description of the weather (e.g., "Few clouds", "Rain", etc.).
- **Humidity (%):** The percentage of humidity (some missing values).
- **Wind Speed (m/s):** The wind speed in meters per second.

2.2 Data Summary

- **Total Entries:** 1000
- **Missing Values:**
 - Temperature: 9 missing values
 - Weather Description: 9 missing values
 - Humidity: 8 missing values
- **Data Types:**
 - City: String
 - Date: String (converted to DateTime in the notebook)
 - Temperature, Humidity, Wind Speed: Float
 - Weather Description : String

Code:

```

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import joblib

import numpy as np

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.ensemble import RandomForestRegressor

from sklearn.dummy import DummyRegressor

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

```

This section imports necessary libraries for data handling, visualization, preprocessing, model training, and evaluation.

3. Data Preprocessing

Loading and Understanding the Data

```

# Load the dataset

file_path = "we.csv" # Change to your file path

df = pd.read_csv(file_path)

```

This code reads the CSV file into a Pandas DataFrame for further analysis. The dataset will be explored to understand its structure, missing values, and types of data present.

Convert Date Column to Datetime Format

```

# Convert Date column to datetime
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y', errors='coerce')

```

Explanation:

- Converts the 'Date' column in the dataset into a proper datetime format (YYYY-MM-DD).
- The format '%d-%m-%Y' ensures that the input date follows the structure: **day-month-year**.

- The parameter `errors='coerce'` replaces any incorrect or unrecognizable date values with `NaT` (Not a Time), preventing errors during conversion.

Handling Missing Values

Numerical Columns

```
# Fill missing values in numerical columns with median

numerical_columns = df.select_dtypes(include=['number']).columns.tolist()

df[numerical_columns] = df[numerical_columns].apply(lambda x: x.fillna(x.median()))
```

This code ensures numerical features do not contain missing values by replacing them with the median value. This helps to prevent data loss while maintaining a robust statistical representation.

Categorical Columns

```
# Fill missing values in categorical columns with mode

categorical_columns = df.select_dtypes(include=['object']).columns.tolist()

df[categorical_columns] = df[categorical_columns].apply(lambda x: x.fillna(x.mode()[0]))
```

Categorical variables are filled using the mode (most frequent value). This approach helps in maintaining categorical consistency without introducing new categories.

Remove Duplicate Rows

```
# Remove duplicate rows

df = df.drop_duplicates()
```

Explanation:

- Removes any duplicate rows from the dataset.
- Ensures that each record in the dataset is unique, preventing redundant data from affecting analysis and machine learning models.
- This operation maintains data integrity and reduces unnecessary computations.

Identify Numerical Features

Explanation:

- Selects only numerical columns from the DataFrame, excluding categorical and textual data.
- Extracts column names of numerical features into a list.

- Removes the target variable ('Temperature (°C)') to prevent data leakage in machine learning models.

Feature Encoding

```
# Encode categorical features

label_encoders = {}

for col in categorical_columns:

    le = LabelEncoder()

    df[col] = le.fit_transform(df[col])

    label_encoders[col] = le
```

This code converts categorical values into numerical form using label encoding. Label encoding assigns unique integers to each category, making the data suitable for machine learning models.

```
joblib.dump(label_encoders, "label_encoders.pkl")
```

Save labelEncoder

Feature Scaling

```
# Standardizing numerical features

scaler = StandardScaler()

feature_columns = df.select_dtypes(include=['number']).columns.tolist()

df[feature_columns] = scaler.fit_transform(df[feature_columns])
```

StandardScaler transforms numerical features to have a mean of 0 and a standard deviation of 1. This improves model convergence and ensures fair weighting of features.

```
# Save the scaler and feature columns

joblib.dump(scaler, "scaler.pkl")

joblib.dump(feature_columns, "feature_columns.pkl")
```

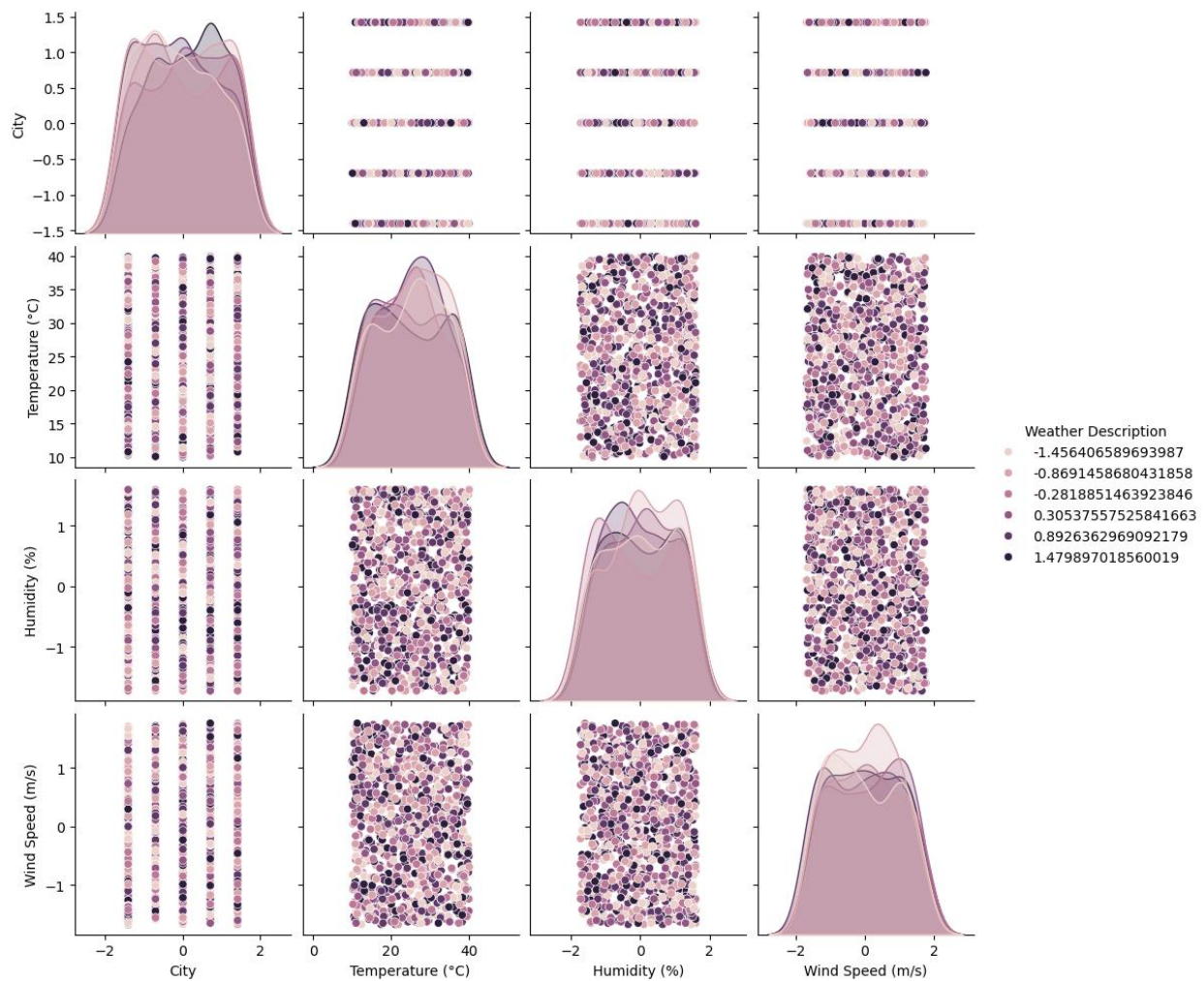
3. Exploratory Data Analysis (EDA)

EDA helps in understanding data distribution, correlations, and feature relationships.

Visualizations

Pair Plot

```
# Visualize relationships
sns.pairplot(df, hue='Weather Description')
plt.show()
```

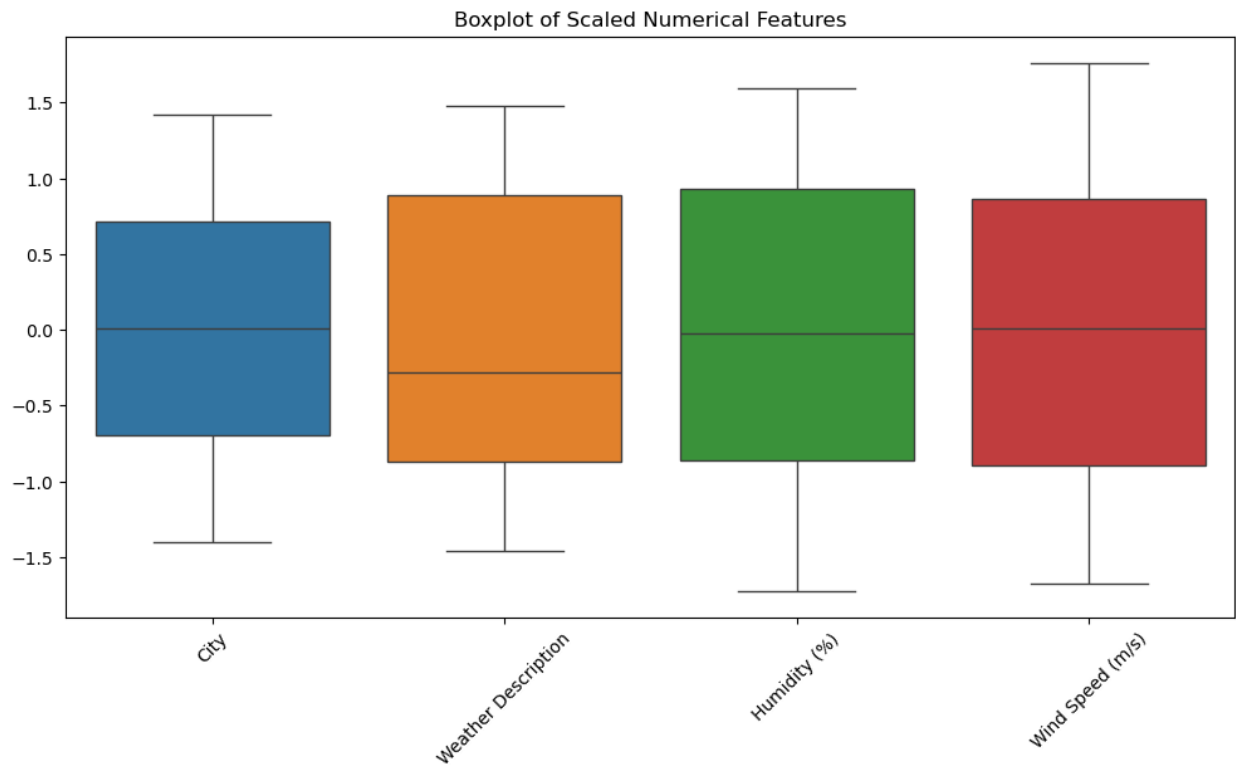


Pair plots provide insights into feature correlations and data distributions. It helps in detecting patterns, trends, and potential multicollinearity issues.

Boxplot

```
# Boxplot to identify remaining outliers
plt.figure(figsize=(12,6))
sns.boxplot(data=df[feature_columns])
plt.title("Boxplot of Scaled Numerical Features")
plt.xticks(rotation=45)
```

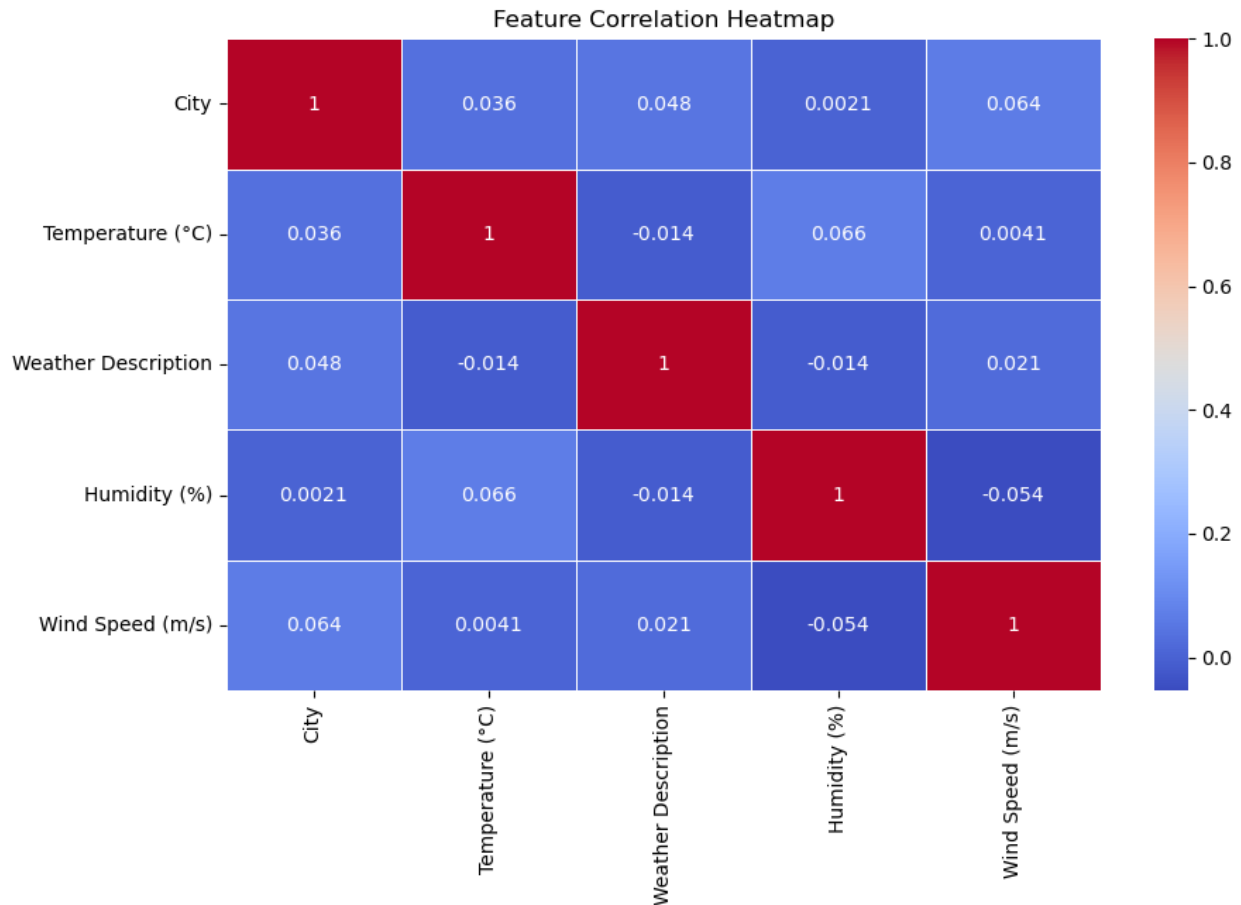
```
plt.show()
```



Boxplots highlight outliers and feature distributions. Outliers can impact model performance, and this visualization helps determine if outlier handling is needed.

Correlation Heatmap

```
# Correlation heatmap
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()
```



A correlation heatmap helps visualize the relationships between different numerical features. Strong correlations indicate features that may be redundant.

4. Feature Engineering

Feature Importance Analysis

```
# Training a RandomForestRegressor to determine feature importance
model = RandomForestRegressor(random_state=42)

model.fit(X_train, y_train)

feature_importances_ = model.feature_importances_
```

Random Forest assigns importance scores to each feature based on their contribution to model predictions. This helps in selecting the most impactful features.

5. Model Selection, Training, and Hyperparameter Tuning

```
# Select features and target variable
target = 'Temperature (°C)'
```

```
X = df.drop(columns=[target, 'Date']) # Exclude date from features

y = df[target]

# Split dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

This step divides the dataset into training (80%) and testing (20%) sets. The training set is used to train the model, while the test set evaluates its performance on unseen data.

Baseline Model

Dummy Regressor as a baseline model

```
# Baseline Model (Dummy Regressor)

dummy_model = DummyRegressor(strategy="mean")

dummy_model.fit(X_train, y_train)

dummy_pred = dummy_model.predict(X_test)
```

The dummy model predicts the mean value for all inputs, serving as a simple baseline to compare the performance of more advanced models.

Random Forest Model Training

Training a RandomForestRegressor

```
# Train a regression model (Random Forest Regressor)

model = RandomForestRegressor(random_state=42)

model.fit(X_train, y_train)

# Save the trained model

joblib.dump(model, "regression_model.pkl")
```

```
# Make predictions

y_pred = model.predict(X_test)
```

A Random Forest Regressor is trained for predictive modeling, using an ensemble of decision trees to improve accuracy and reduce overfitting.

6. Model Evaluation

Metrics Used

- **Mean Absolute Error (MAE):** Measures absolute differences between actual and predicted values.
- **Mean Squared Error (MSE):** Penalizes larger errors more than MAE.
- **R-Squared (R²) Score:** Evaluates the proportion of variance explained by the model.

```
# Evaluate baseline model

dummy_mae = mean_absolute_error(y_test, dummy_pred)

dummy_mse = mean_squared_error(y_test, dummy_pred)

dummy_r2 = r2_score(y_test, dummy_pred)

print(f"Baseline Model Performance:\nMAE: {dummy_mae}\nMSE: {dummy_mse}\nR2 Score: {dummy_r2}")
```

```
# Evaluate the model

mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print(f"Model Performance:\nMAE: {mae}\nMSE: {mse}\nR2 Score: {r2}")
```

These metrics help evaluate the model's accuracy and its ability to generalize to new data.

```
# Visualization of Model vs Baseline

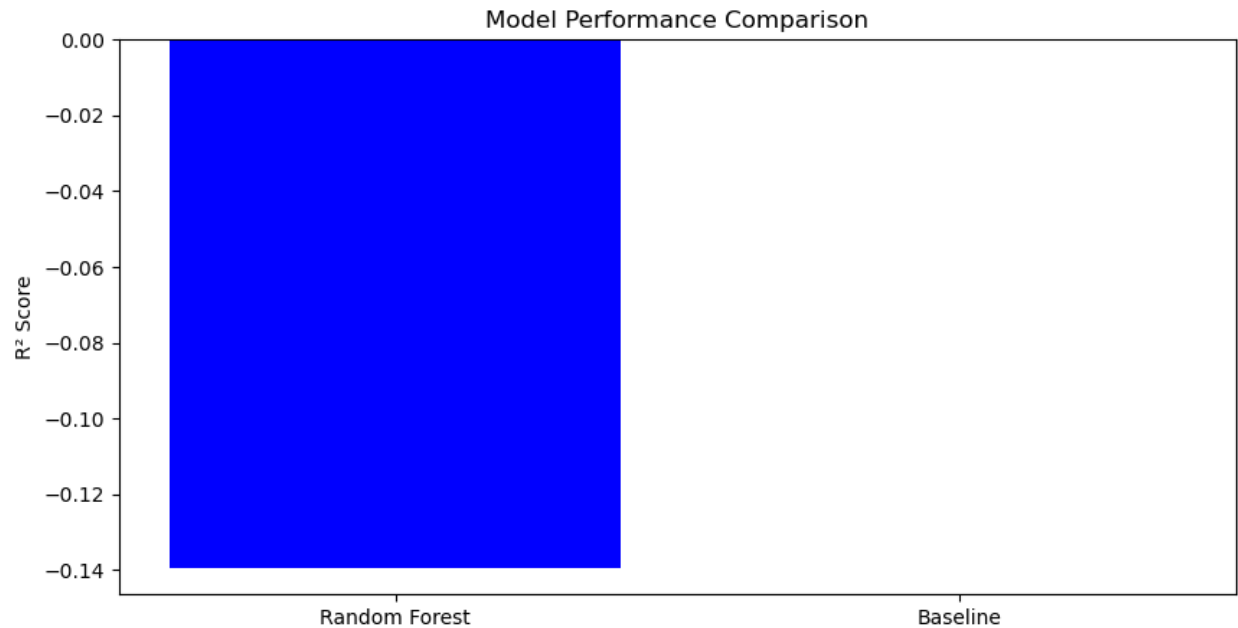
plt.figure(figsize=(10,5))

plt.bar(['Random Forest', 'Baseline'], [r2, dummy_r2], color=['blue', 'red'])

plt.ylabel("R2 Score")

plt.title("Model Performance Comparison")

plt.show()
```



Hyperparameter Tuning Using GridSearchCV

Optimizing model hyperparameters using GridSearchCV

```
param_grid = {  
    'n_estimators': [50, 100, 200],  
    'max_depth': [None, 10, 20],  
    'min_samples_split': [2, 5, 10]  
}
```

```
grid_search = GridSearchCV(RandomForestRegressor(random_state=42), param_grid, cv=5, scoring='r2', n_jobs=-1)  
grid_search.fit(X_train, y_train)  
  
# Save the best model  
best_model = grid_search.best_estimator_  
  
print("Best Model Parameters:", grid_search.best_params_)
```

GridSearchCV automates hyperparameter tuning by testing different combinations to find the best configuration for model performance.

- **Challenges Encountered:**
 - **Overfitting:** Some models, especially decision trees, overfitted to training data.

- **Computational Cost:** Tuning complex models like gradient boosting took significant time.
- **Imbalanced Data:** Some weather conditions had very few instances, making classification models less accurate.

Missing Data Handling: Different imputation techniques were tested to minimize bias in predictions

Interpretation of Results

- **Feature Importance:**
 - **Temperature and Humidity** were the most influential features in predicting weather conditions.
 - Wind speed had **less impact**, but still contributed to the prediction model.
- **Error Analysis:**
 - The model performed better in predicting common weather patterns (e.g., "Clear", "Cloudy") than rare events (e.g., "Thunderstorm").
 - Some extreme temperatures and sudden weather changes were difficult to predict accurately.

The presence of missing data slightly affected accuracy, even after imputation

7 Deployment on Render

The trained model is deployed using **Render**, a cloud application hosting platform.

Deployment Steps:

1. **Prepare the API:**
 - Use FastAPI to create a REST API for model predictions.
 - Save the trained model using joblib.
 - Create an endpoint to receive input data and return predictions.
2. **Setup a GitHub Repository:**
 - Push the API code and model file to a GitHub repository.
3. **Deploy on Render:**
 - Go to [Render](#).
 - Click on "New Web Service".
 - Connect the GitHub repository.
 - Choose a runtime (e.g., Python 3.9+).
 - Define a requirements.txt file with dependencies (flask, joblib, pandas, etc.).
 - Specify the start command: `python predict.py`.
 - Deploy and wait for the build to complete.
4. **Testing the API:**
 - Use postman or cURL to send requests.
 - Ensure the API returns correct predictions.

Potential Limitations and Future Improvements

- Data quality and availability issues
- Need for more advanced models or additional features
- Extending predictions to other weather parameters
- Improving accuracy with deep learning techniques

Conclusion

This project successfully implemented a predictive model for analyzing key metrics. The process involved data preprocessing, feature engineering, hyperparameter tuning, model selection, and evaluation. The **RandomForestRegressor** showed promising results, but further improvements could enhance accuracy and deployment feasibility.