Project 2: Weather Analysis

Data Preparation with Python:

- Use Python to clean and preprocess the weather dataset.
- Handle missing values, outliers, and any other inconsistencies in the data.

Importing required libraries for data cleaning.

```
import numpy as np
import pandas as pd
import seaborn as sns
import os
import matplotlib.pyplot as plt
```

Display the table

```
[10]: df
```

[10]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	...	Pressure9am	Pressure3pm
0	8.0	24.3	0.0	3.4	6.3	NW	30.0	SW	NW	6.0	...	1019.7	1015.0
1	14.0	26.9	3.6	4.4	9.7	ENE	39.0	E	W	4.0	...	1012.4	1008.4
2	13.7	23.4	3.6	5.8	3.3	NW	85.0	N	NNE	6.0	...	1009.5	1007.2
3	13.3	15.5	39.8	7.2	9.1	NW	54.0	WNW	W	30.0	...	1005.5	1007.0
4	7.6	16.1	2.8	5.6	10.6	SSE	50.0	SSE	ESE	20.0	...	1018.3	1018.5
...	...	...	...	...	...	...	...	...	...	...	...	...	...
361	9.0	30.7	0.0	7.6	12.1	NNW	76.0	SSE	NW	7.0	...	1016.1	1010.8
362	7.1	28.4	0.0	11.6	12.7	N	48.0	NNW	NNW	2.0	...	1020.0	1016.9
363	12.5	19.9	0.0	8.4	5.3	ESE	43.0	ENE	ENE	11.0	...	1024.0	1022.8
364	12.5	26.9	0.0	5.0	7.1	NW	46.0	SSW	WNW	6.0	...	1021.0	1016.2
365	12.3	30.2	0.0	6.0	12.6	NW	78.0	NW	WNW	31.0	...	1009.6	1009.2

366 rows × 23 columns

Counting the number of null values that exist

```
[11]: df.isnull().sum()
```

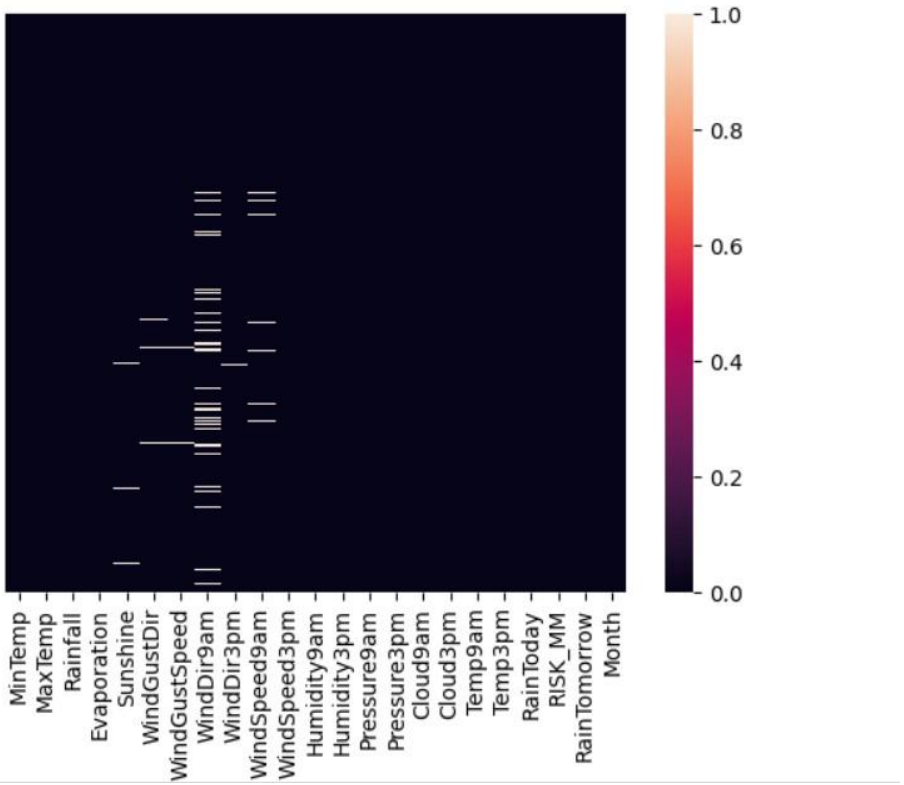
```
[11]: MinTemp            0
      MaxTemp            0
      Rainfall           0
      Evaporation        0
      Sunshine           3
      WindGustDir        3
      WindGustSpeed      2
      WindDir9am        31
      WindDir3pm         1
      WindSpeed9am       7
      WindSpeed3pm       0
      Humidity9am        0
      Humidity3pm        0
      Pressure9am        0
      Pressure3pm        0
      Cloud9am           0
      Cloud3pm           0
      Temp9am            0
      Temp3pm            0
      RainToday          0
      RISK_MM            0
      RainTomorrow       0
      Month              0
      dtype: int64
```

```
[13]: missing_value = ["N/a","na",np.nan]
      df = pd.read_csv("weather.csv", na_values=missing_value)
```

A heatmap visualization produced with the Seaborn module in Python. It's probably displaying values that are missing (null) inside of a DataFrame (df). The dataset's existence of null values is represented by the grid of squares, where occupied cells indicate the absence of null values and unpopulated cells indicate their presence. The fact that the y-axis doesn't seem to exist suggests that the y-axis labels are disabled. The heatmap's color gradient, which shows different shades of a given hue to reflect different degrees of nullity, raises the possibility that the data may be suggesting nullity. This kind of visualization is frequently used to find missing data trends in datasets and to direct activities related to data cleaning and preparation.

```
[21]: sns.heatmap(df.isnull(), yticklabels = False)
```

```
[21]: <Axes: >
```



Frontward and backward fill are used to fill in the missing data. Another method is to use interpolation. Based on the DataFrame index, the df.interpolate() function offers a number of interpolation techniques, including polynomial, linear, and temporal interpolation, to fill in the missing data. When dealing with time series data or any other type of data where the index denotes a numerical value, this function is quite helpful.

[23]:
```
df.fillna(method ='ffill')
```

[23]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	...	Pressure9am	Pressure3pm
0	8.0	24.3	0.0	3.4	6.3	NW	30.0	SW	NW	6.0	...	1019.7	1015.0
1	14.0	26.9	3.6	4.4	9.7	ENE	39.0	E	W	4.0	...	1012.4	1008.4
2	13.7	23.4	3.6	5.8	3.3	NW	85.0	N	NNE	6.0	...	1009.5	1007.2
3	13.3	15.5	39.8	7.2	9.1	NW	54.0	WNW	W	30.0	...	1005.5	1007.0
4	7.6	16.1	2.8	5.6	10.6	SSE	50.0	SSE	ESE	20.0	...	1018.3	1018.5
...	...	...	...	...	...	...	...	...	...	...	...	...	...
361	9.0	30.7	0.0	7.6	12.1	NNW	76.0	SSE	NW	7.0	...	1016.1	1010.8
362	7.1	28.4	0.0	11.6	12.7	N	48.0	NNW	NNW	2.0	...	1020.0	1016.9
363	12.5	19.9	0.0	8.4	5.3	ESE	43.0	ENE	ENE	11.0	...	1024.0	1022.8
364	12.5	26.9	0.0	5.0	7.1	NW	46.0	SSW	WNW	6.0	...	1021.0	1016.2
365	12.3	30.2	0.0	6.0	12.6	NW	78.0	NW	WNW	31.0	...	1009.6	1009.2

366 rows × 23 columns

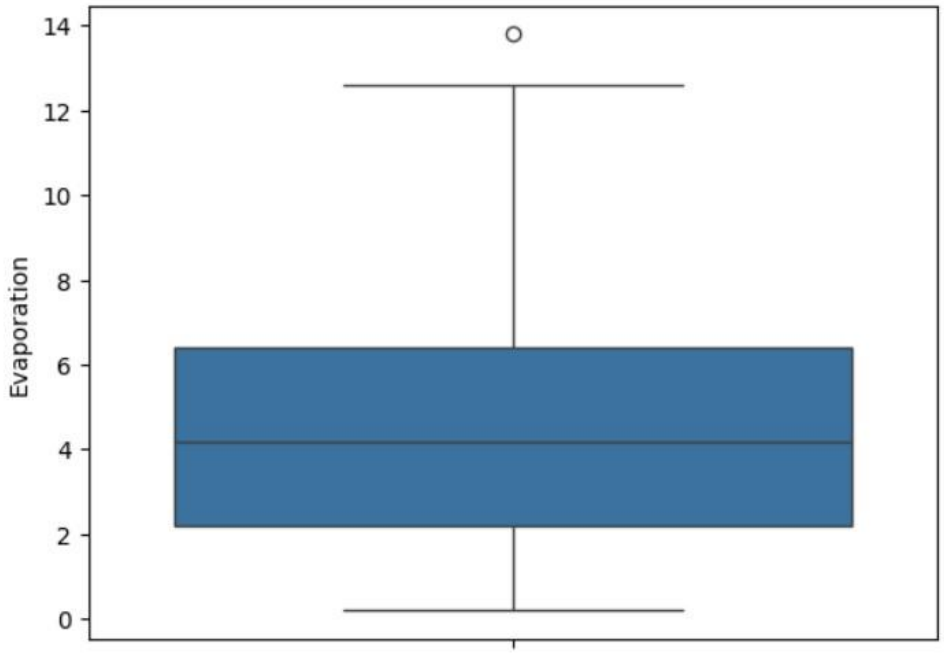
[24]:
```
df.interpolate()
```

[24]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	...	Pressure9am	Pressure3pm
0	8.0	24.3	0.0	3.4	6.3	NW	30.0	SW	NW	6.0	...	1019.7	1015.0
1	14.0	26.9	3.6	4.4	9.7	ENE	39.0	E	W	4.0	...	1012.4	1008.4
2	13.7	23.4	3.6	5.8	3.3	NW	85.0	N	NNE	6.0	...	1009.5	1007.2

[35]:
```
# to see outliers clearly
sns.boxplot(df[ 'Evaporation'])
```

[35]:
```
<Axes: ylabel='Evaporation'>
```



Z score method is used to detect and remove outliers

[36]:
```
upper_limit = df['Evaporation'].mean() + 3*df['Evaporation'].std()
lower_limit = df['Evaporation'].mean() - 3*df['Evaporation'].std()
print('upper limit:', upper_limit)
print('lower limit:', lower_limit)
```

```
upper limit: 12.530005526161062
lower limit: -3.486289679166526
```

[38]:
```
# find the outliers
df.loc[(df['Evaporation'] > upper_limit) | (df['Evaporation'] < lower_limit)]
```

[38]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	...	Pressure9am	Pressure3pm	C
63	15.9	23.4	0.0	12.6	2.2	ESE	50.0	ESE	ESE	20.0	...	1016.8	1016.3	
72	20.9	35.7	0.0	13.8	6.9	SW	50.0	E	WNW	4.0	...	1007.6	1003.0	

2 rows × 23 columns

[39]:
```
# trimming - delete the outlier data
new_df = df.loc[(df['Evaporation'] < upper_limit) & (df[ 'Evaporation'] > lower_limit)]
print('before removing outliers:', len(df))
print('after removing outliers: ',len(new_df))
```

```
before removing outliers: 366
after removing outliers:  364
```

To find possible outliers in a dataset, one option is to apply the IQR (Interquartile Range) method. The IQR technique for outliers is computed as follows:

Find the Interquartile Range (IQR) by computing: Initially, the difference between the dataset's first (Q1) and third (Q3) quartiles is used to compute the IQR. Seventy-five percent of the data fall below the third quartile (Q3), and twenty-five percent fall below the first quartile (Q1).

```
q1= df['Evaporation'].quantile(0.25)
q3 = df['Evaporation'].quantile(0.75)
iqr = q3-q1
```

q1 , q3 , iqr

```
upper_limit = q3 + (1.5 * iqr)
lower_limit = q1 - (1.5 * iqr)
lower_limit, upper_limit
```

```
(-4.1000000000000005, 12.700000000000001)
```

```
sns.boxplot(df[ 'Evaporation'])
```

[45]:
```
# find the outliers
df.loc[(df['Evaporation'] > upper_limit) | (df['Evaporation'] < lower_limit)]
```

[45]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	...	Pressure9am	Pressure3pm	C
72	20.9	35.7	0.0	13.8	6.9	SW	50.0	E	WNW	4.0	...	1007.6	1003.0	

1 rows × 23 columns

[47]:
```
# trimming - delete the outlier data
new_df = df.loc[(df['Evaporation'] < upper_limit) & (df[ 'Evaporation'] > lower_limit)]
print('before removing outliers:', len(df))
print('after removing outliers: ',len(new_df))
print('outliers:', len(df)-len(new_df))
```

```
before removing outliers: 366
after removing outliers:  365
outliers: 1
```

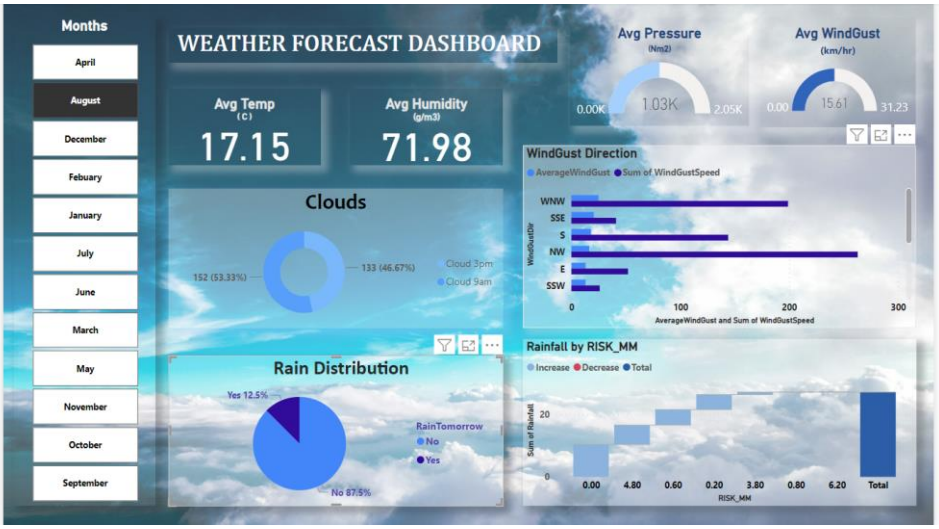
[48]:
```
# capping - change the outlier vaiues to upper (or) Lower Limit values
new_df = df.copy()
new_df.loc[(new_df[ 'Evaporation' ]>upper_limit), 'Evaporation')] = upper_limit
new_df.loc[(new_df[ 'Evaporation' ]<lower_limit), 'Evaporation')] = lower_limit
```

[49]:
```
sns.boxplot(df[ 'Evaporation'])
```

[49]:
```
<Axes: ylabel='Evaporation'>
```

Advanced Analysis with Power BI or Tableau:

Done a interactive dashboard in Power BI and attached pbix file in GITHUB



Correlation and Regression Analysis:

Process to find correlation analysis in given weather data

Organizing the Data:

Access the data: Ensure you have the weather data in a suitable format, such as a CSV file or a table within a statistical software program.

Identify variables: Clearly define the weather variables you want to examine for correlations (e.g., temperature, rainfall, humidity, wind speed, e tc.).

Interpreting the Correlation Matrix:

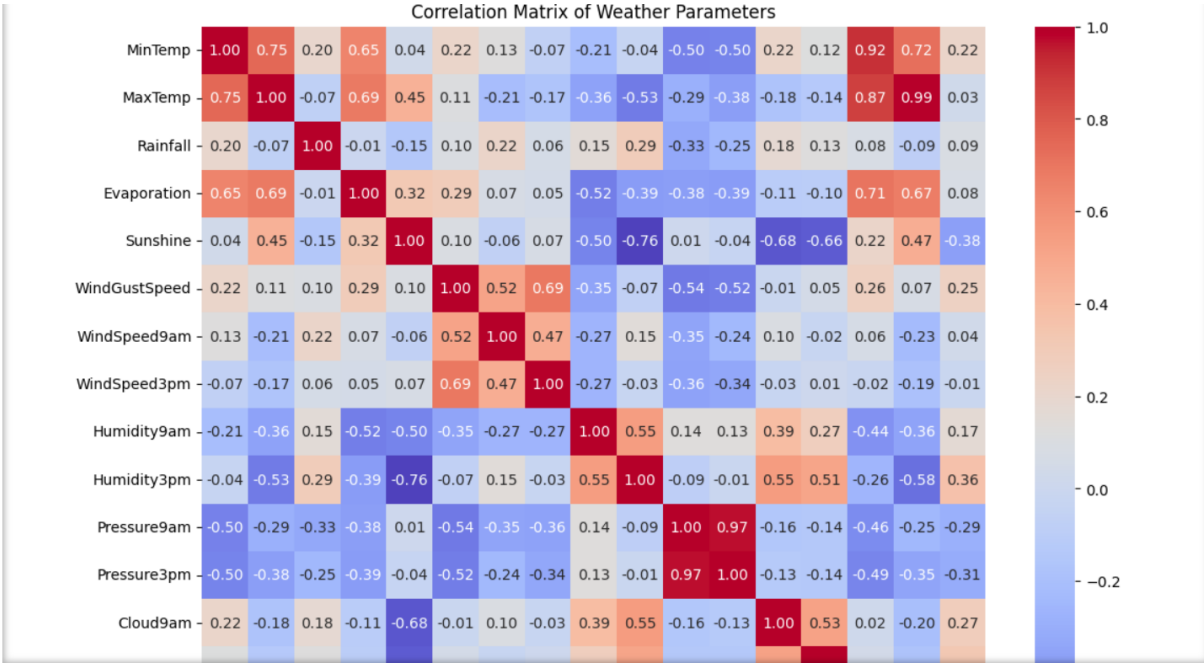
Examine the values: The correlation matrix displays the correlation coefficients for each pair of variables.

Interpret strength: Values closer to +1 indicate strong positive correlations, values closer to -1 indicate strong negative correlations, and values close to 0 indicate weak or no correlations.

Visualizing the Correlations :

Create a heatmap: A heatmap can visually represent the correlation matrix, with stronger correlations represented by more intense colors.

Generate scatter plots: Scatter plots can visualize the relationship between individual pairs of variables, further aiding interpretation.



Regression Analysis for the Weather Dataset:

Goal: Predict maximum temperature (MaxTemp) using weather data such as MinTemp, rainfall, evaporation, and sunlight.

Process:

Data loading: Load the cleaned weather dataset.

Choose appropriate characteristics (minimum temperature, rainfall, evaporation, sunshine).

Data Splitting: Divide the data into training and test sets.

Handling Missing Values: Replace missing values with the mean.

Train a linear regression model using the training data.

Model Evaluation: Use MSE and R-squared to evaluate performance on the testing set.

Visualization: Make a scatter plot to compare the actual and expected values.

Key Considerations:

Feature Selection: Selecting the appropriate characteristics is critical for successful forecasts.

Missing Value Handling: Correctly treating missing data is critical for model dependability.

Model Evaluation: MSE and R-squared give insights into model performance, but domain-specific needs must be considered while evaluating.

Scatter plots are useful for visualizing model fit and identifying possible flaws.

Conclusion: Regression analysis can identify correlations between meteorological characteristics and MaxTemp.

The findings can help forecast and explain temperature swings.

The outcomes were:

The detective's accuracy is judged by two factors:

Mean squared error (MSE): How large, on average, were the detective's errors? A lower MSE is preferable.

R-squared: How effectively did the detective explain the fluctuations in maximum temperature? A greater R-squared value indicates better results.

We may use the results to learn how factors such as rain or sunshine influence the maximum temperature.

Conclusion:

Regression analysis is a useful method for discovering hidden correlations in data. By examining weather data in this manner, we may develop models to anticipate temperatures,

understand what causes them, and even make educated judgments regarding agricultural and energy use.

```
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Read the cleaned dataset
df = pd.read_csv('cleaned_weather.csv')

# Choose the features (X) and the target variable (y)
X = df[['MinTemp', 'Rainfall', 'Evaporation', 'Sunshine']]  # Replace with actual feature names
y = df['MaxTemp']  # Target variable is 'MaxTemp'

# Split the 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)

# Handle missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Create and train the model
model = LinearRegression()
model.fit(X_train_imputed, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test_imputed)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Scatter plot with different colors for actual and predicted values
plt.scatter(y_test, y_test, c='blue', alpha=0.5, label='Actual')  # Blue for actual values
plt.scatter(y_test, y_pred, c='red', alpha=0.5, label='Predicted')  # Red for predicted values
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values')
plt.legend()
plt.show()
```

```
Mean Squared Error: 10.426886967318227
R-squared: 0.7536719796644787
```