Name: Wind Speed Prediction Dataset

1(a) Selection an Engineering Dataset

Purpose: The goal of this dataset is to predict the weather conditions based on the features of this dataset.

Link and Citation of the dataset: I have collected this dataset from Kaggle for this assignment, as you required. The link and citation of the dataset are below.

https://www.kaggle.com/datasets/fedesoriano/wind-speed-prediction-dataset?resource=download,

citation: fedesoriano. (April 2022). Wind Speed Prediction Dataset. Retrieved [Date Retrieved] from https://www.kaggle.com/datasets/fedesoriano/wind-speed-prediction-dataset

1(b) Loading the dataset

I am a user of the Ubuntu operating system, so instead of a Jupyter notebook, I am using Colab, which has some benefits for saving time.

Following these procedures:

Step 1: I have uploaded the dataset from the machine to a directory called Sample_Data in my Google Drive.

Step 2: Then I mounted the drive by using the simple code below.

```
# Mount the drive
from google.colab import  drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

So, my drive is mounted now!

Import Library:

First, I will import the Pandas library to upload the selected data as a dataframe.

```
# Now import pandas library as pd
import pandas as pd
```

```
# Assume, the name of dataset is wind_predict

wind_predict = pd.read_csv('/content/drive/MyDrive/Sample_Data/Engineering_Data/wind_dataset.csv', index_col='DATE')
```

Now, we will see how many rows and columns are in this dataset, and we will also see a sample of the first 5 rows in the next cell.

```
# First 5 lines of this dataset

wind_predict.head(5)
```

	WIND	IND	RAIN	IND.1	T.MAX	IND.2	T.MIN	T.MIN.G
DATE								
1961-01-01	13.67	0	0.2	0.0	9.5	0.0	3.7	-1.0
1961-01-02	11.50	0	5.1	0.0	7.2	0.0	4.2	1.1
1961-01-03	11.25	0	0.4	0.0	5.5	0.0	0.5	-0.5
1961-01-04	8.63	0	0.2	0.0	5.6	0.0	0.4	-3.2
1961-01-05	11.92	0	10.4	0.0	7.2	1.0	-1.5	-7.5

So, there are 5 rows and 8 columns with an index in this dataset.

Find the total number of rows and columns:

```
# Find number of rows and colums of the Dataset

number_rows, number_colms = wind_predict.shape
print(f'The dataset has {number_rows} rows and {number_colms} columns.')
```

```
The dataset has 6574 rows and 8 columns.
```

1(c) Description of the dataset

About Datset:

The Wind Speed Prediction Dataset is important for weather experts who need accurate wind forecasts for severe weather alerts. Collected from 1961 to 1978, it has over 6,500 daily readings from a weather station in an empty field 21 metres above sea level. The station tracks not just wind but also rain and temperatures. This detailed information helps scientists understand how factors like storms and tornadoes develop, which can cause major damage like power outages and destruction to forests and buildings. The dataset is useful for both looking back at past weather and improving future wind speed predictions.

Attributes Information of the Dataset:

Date (Formatted as YYYY-MM-DD)
Wind Speed Avg: Mean wind velocity measured in knots
First Indicator: Initial indicator metric
Precipitation: Rainfall amount in millimeters
Second Indicator: Subsequent indicator metric
Max Temp: Peak temperature in degrees Celsius
Third Indicator: Additional indicator value
Min Temp: Lowest temperature in degrees Celsius
Grass Min Temp at 09 UTC: Minimum temperature on grass surface measured at 09:00 UTC in degrees Celsius.

Conclusion: I want to work on this dataset for a long time and want to improve it by adding categorical data columns that can help me understand how I can play with it. In the next assignments, the dataset will be improved regarding our requirements.

2(a)

Data Exploration:

Basic Statistics- mean, median, standard deviation, minimum, and maximum values.

Before analysis, I want to make sure that the dataset is clean and well-prepared. Remove any missing or inconsistent values.

Identify the Null Values:

```
print(wind_predict.isnull().sum())
```

```
WIND         0
IND          0
RAIN         0
IND.1       61
T.MAX      621
IND.2       61
T.MIN      674
T.MIN.G    360
dtype: int64
```

Removing the null values:

```
wind_predict.dropna(inplace=True)
```

So, our new dataset is very good for data visualisation and manipulation.

```
# First 5 lines of this dataset after removing NUll:
wind_predict.head(5)
```

DATE	WIND	IND	RAIN	IND.1	T.MAX	IND.2	T.MIN	T.MIN.G
1961-01-01	13.67	0	0.2	0.0	9.5	0.0	3.7	-1.0
1961-01-02	11.50	0	5.1	0.0	7.2	0.0	4.2	1.1
1961-01-03	11.25	0	0.4	0.0	5.5	0.0	0.5	-0.5
1961-01-04	8.63	0	0.2	0.0	5.6	0.0	0.4	-3.2
1961-01-05	11.92	0	10.4	0.0	7.2	1.0	-1.5	-7.5

```
## Lets gather some information about the wind predited dataset
wind_predict.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5638 entries, 1961-01-01 to 1978-12-31
Data columns (total 8 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   WIND     5638 non-null   float64
 1   IND      5638 non-null   int64  
 2   RAIN     5638 non-null   float64
 3   IND.1    5638 non-null   float64
 4   T.MAX    5638 non-null   float64
 5   IND.2    5638 non-null   float64
 6   T.MIN    5638 non-null   float64
 7   T.MIN.G  5638 non-null   float64
dtypes: float64(7), int64(1)
memory usage: 396.4+ KB
```

Describe the dataset:

```
# Statistics of the wind predict dataset: mean, median, standard deviation, minimum, and maximum values for every columns in the
wind_predict.describe().T
```

	count	mean	std	min	25%	50%	75%	max
WIND	5638.0	9.682297	4.938009	0.0	5.91	9.08	12.83	30.37
IND	5638.0	0.399610	1.189562	0.0	0.00	0.00	0.00	4.00
RAIN	5638.0	1.875647	3.973763	0.0	0.00	0.20	2.00	67.00
IND.1	5638.0	0.012593	0.157654	0.0	0.00	0.00	0.00	2.00
T.MAX	5638.0	13.285669	4.890483	-0.1	9.60	13.10	17.20	26.80
IND.2	5638.0	0.093828	0.303538	0.0	0.00	0.00	0.00	3.00
T.MIN	5638.0	6.446949	4.626693	-11.5	3.10	6.50	10.00	18.00
T.MIN.G	5638.0	2.757627	5.576713	-13.5	-1.00	3.00	7.00	15.80

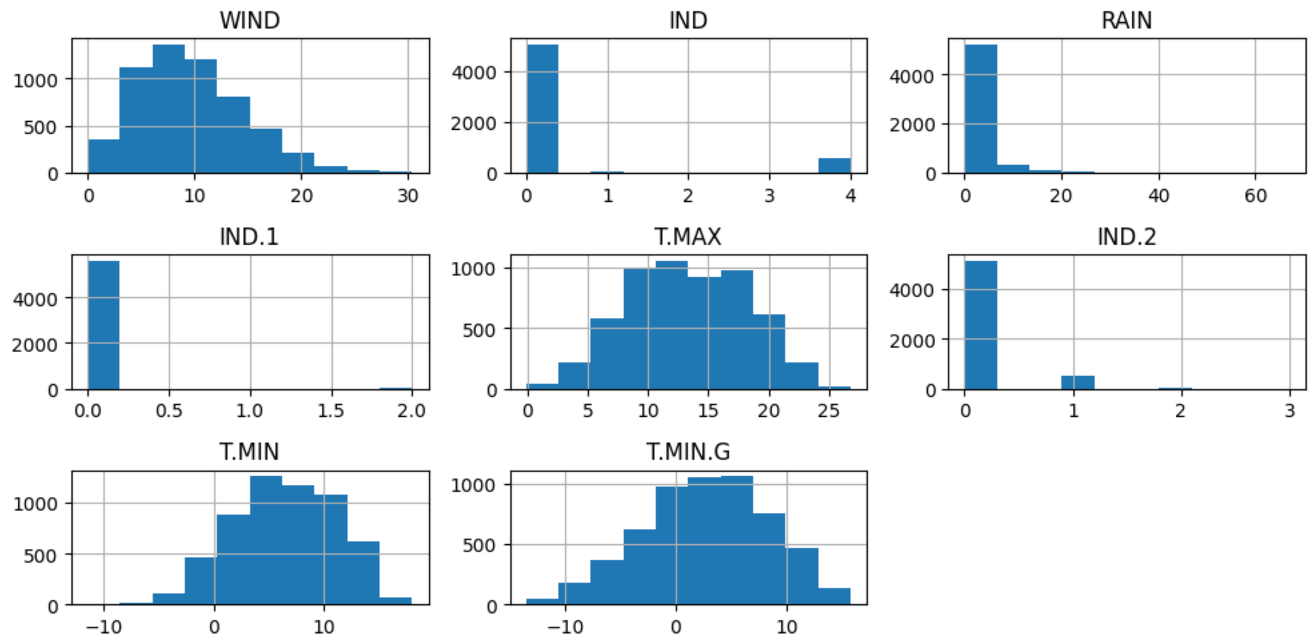
2(b)

```
# import libraries for vusualization

import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew, kurtosis, shapiro
```

Visualizations

Histograms: Plot histograms for each feature to visualize the data distribution.

```
# Histogram plots
wind_predict.hist(figsize=(10, 5))
plt.tight_layout()
plt.show()
```



Conclusion of Histogram plots:

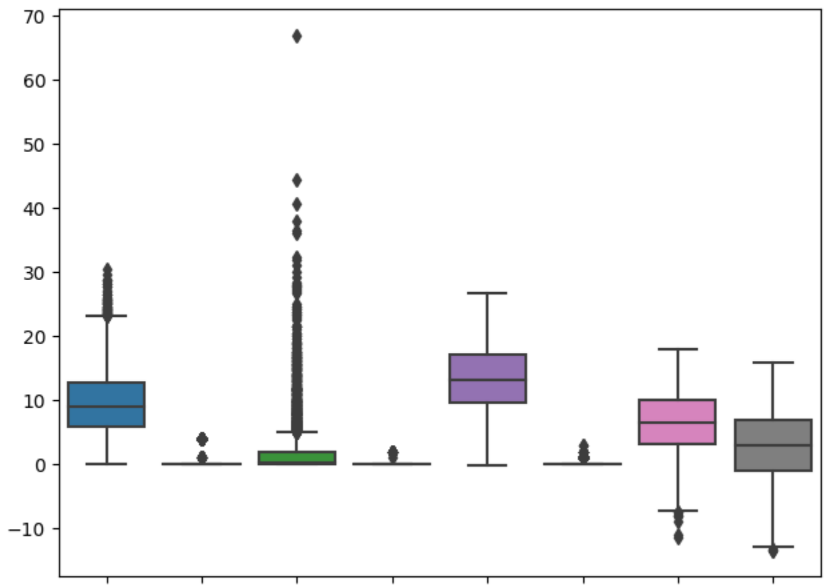
Wind - Right-skewed (or positively skewed) data because it has a tail on the right side.
RAIN - Right-skewed (or positively skewed) data because it has a long tail on the right side.
T.MAX - Normally distributed data, and it has a bell-shaped curve almost.
T.MIN - Left-skewed (or negatively skewed) data, because it has a slight tail on the left side.

I will ignore other features in future analyses..

Conclusion of box plot:

Wind - positively skewed data but lower kurtosis.
RAIN- positively skewed data but higher kurtosis.
T.MAX - Normally distributed data.
T.MIN- negatively skewed data but lower kurtosis.

```
# Box plots
sns.boxplot(data=wind_predict)
plt.tight_layout()
plt.show()
```




```
# Creating a list called numerical feature

numerical_features = ['WIND', 'RAIN', 'T.MAX', 'T.MIN']


# Calculate Skewness and Kurtosis

for feature in numerical_features:
    skewness = wind_predict[feature].skew()
    kurtosis = wind_predict[feature].kurt()
    print(f"{feature} - Skewness: {skewness}, Kurtosis: {kurtosis}\n")
```

```
WIND - Skewness: 0.6394868021222956, Kurtosis: 0.16679322210741265

RAIN - Skewness: 4.3281473442655205, Kurtosis: 30.61501920322467

T.MAX - Skewness: -0.007186803194828085, Kurtosis: -0.734057024975292

T.MIN - Skewness: -0.19036703707673272, Kurtosis: -0.5059306444448715
```

Interpreting Skewness and Kurtosis:

Skewness:

If skewness is less than -1 or greater than 1, the distribution is highly skewed.
If skewness is between -1 and -0.5 or between 0.5 and 1, the distribution is moderately skewed.
If skewness is between -0.5 and 0.5, the distribution is approximately symmetrical.

Kurtosis:

High kurtosis (>3) indicates that data have heavy tails or outliers.

Low kurtosis (<3) indicates that data have light tails or lack of outliers.

Commments:

The distribution of WIND feature is moderatly skewed and it has low kurtosis.

The distribution of RAIN feature is highly skewed and it indicates that data have heavy tails means high kurtosis.

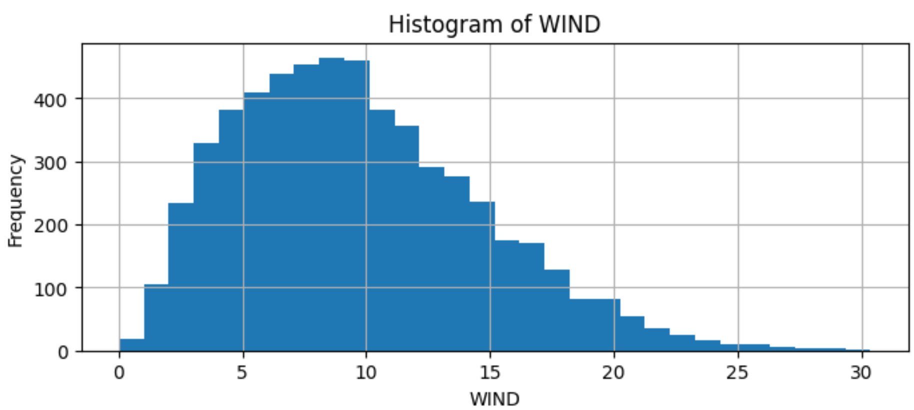
The distribution of T.MAX feature is approximately symmetrical. and it has low kurtosis

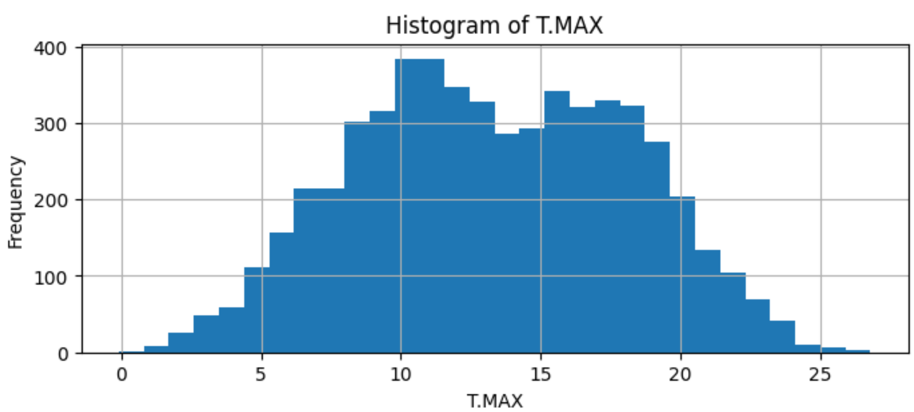
The distribution of T.MIN feature is moderatly skewed and it has low kurtosis.

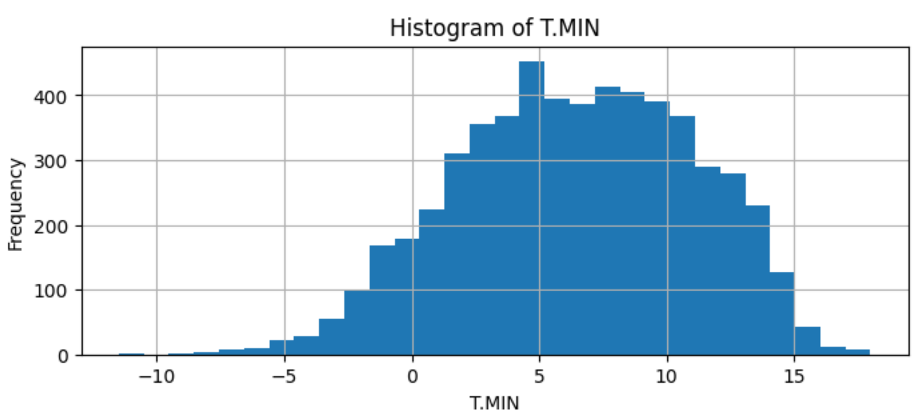
3(a) Data Visualization:

In this solution, I have chosen three features from the dataset. The features are WIND, T.MAX, and T.MIN. I have taken these features for histogram plotting with bin size 30 (For example). In the x-axis, it will print the feature name, and in the y-axis, it will be the frequency of the features. Let's make a code below.

```
# Plotting histograms for each numerical feature
oneORmore_features = ['WIND', 'T.MAX', 'T.MIN']


for feature in oneORmore_features:
    plt.figure(figsize=(8, 3))  # Set the figure size before plotting
    plt.hist(wind_predict[feature], bins=30)
    plt.title(f"Histogram of {feature}")
    plt.xlabel(feature)
    plt.ylabel("Frequency")
    plt.grid()
    #plt.tight_layout()  #elements fit within the figure area
    plt.show()
```



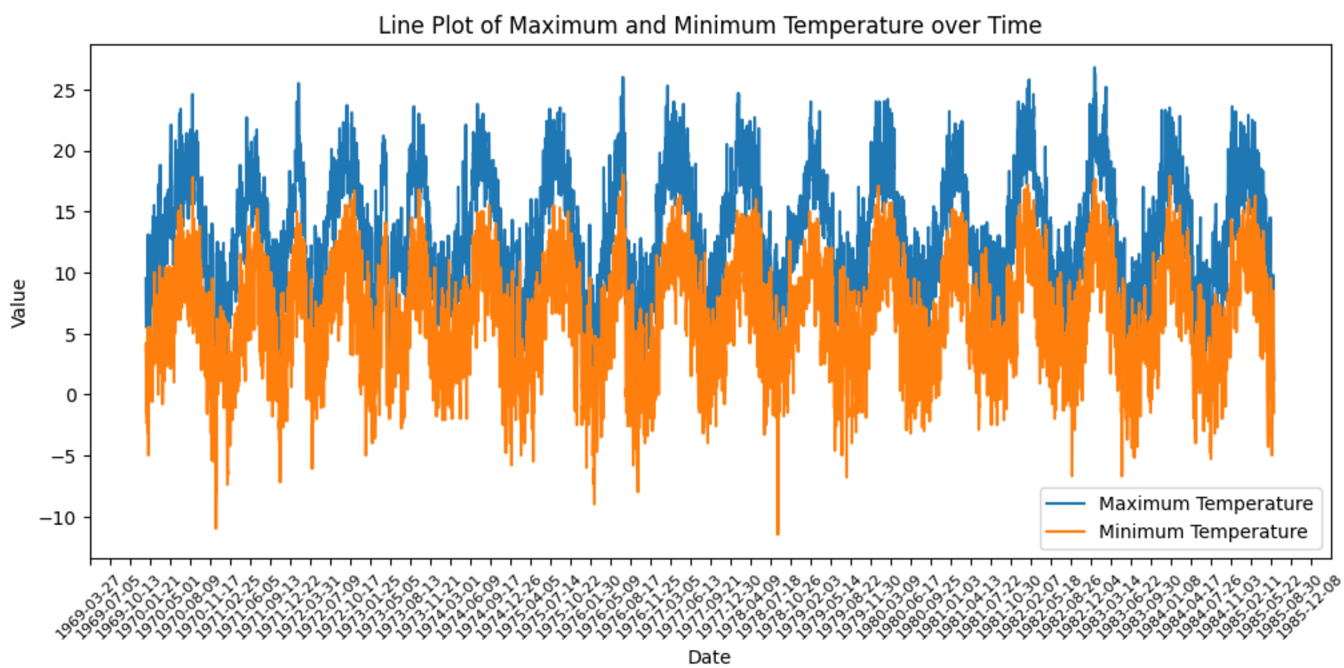




3(b)

Creating a line plot to visualise the relationship between two numerical features can provide insights into how one variable may affect another. In this case, I will make line plots of the maximum temperature (T.MAX) and the minimum temperature (T.MIN) over the date (DATE). I have taken a date interval of 100 between two dates. I hope that it is a very nice visualisation..

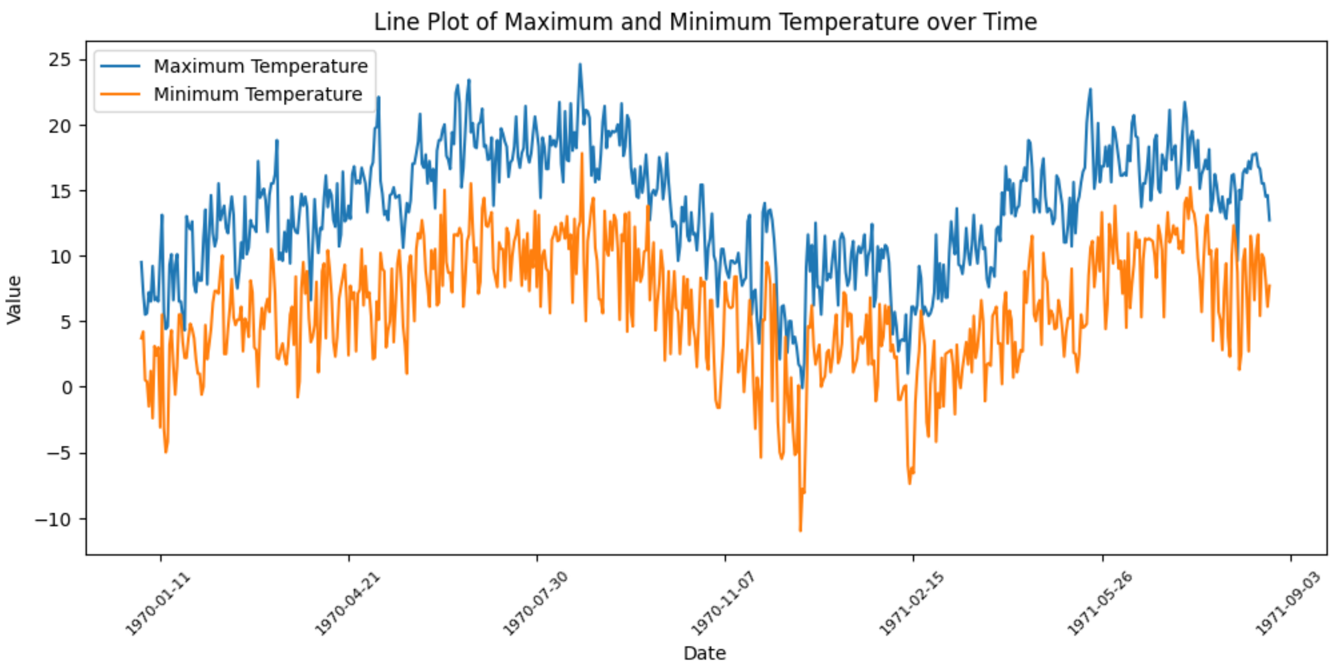
```
# Import library called date
import matplotlib.dates as mdates

# Increase figure size
plt.figure(figsize=(12, 5))

# Line Plot
plt.plot(wind_predict.index, wind_predict['T.MAX'], label='Maximum Temperature')
plt.plot(wind_predict.index, wind_predict['T.MIN'], label='Minimum Temperature')

# Format date and set interval
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=100))

# Rotate and set font size for x-axis labels
plt.xticks(rotation=45, fontsize=8)

plt.title('Line Plot of Maximum and Minimum Temperature over Time')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.show()
```



The above dataset is a large one. For this reason, we cannot clearly see the date on the x-axis. There are several steps to clearly see the date on the x-axis, but here, I will take 1–600 data points for a good visualisation on the x-axis. I have taken an interval of 100 between two dates here.

```
# import a library called dates
import matplotlib.dates as mdates

# Subset the data to first 600 rows
subset_wind_predict = wind_predict.iloc[:600]

# Increase figure size
```

```
plt.figure(figsize=(12, 5))

# Line Plot
plt.plot(subset_wind_predict.index, subset_wind_predict['T.MAX'], label='Maximum Temperature')
plt.plot(subset_wind_predict.index, subset_wind_predict['T.MIN'], label='Minimum Temperature')

# Format date and assume the interval
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=100))

# Rotate and set font size for x-axis labels
plt.xticks(rotation=45, fontsize=8)

plt.title('Line Plot of Maximum and Minimum Temperature over Time')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.show()
```



4(a)

If I'd like to select a subset of rows based on a specific condition related to one of the features, I like to use conditional indexing with Pandas. I want to select rows from the Wind Speed Prediction Dataset where the maximum temperature is greater than 18 °C (for example). In the next cell, we will see a subset of our dataset where the maximum temperature (feature = T.MAX) is greater than 18 °C.

```
# Create subset
subset_oneOffeatures = wind_predict[wind_predict['T.MAX'] > 18]
print(subset_oneOffeatures)
```

```
             WIND  IND  RAIN  IND.1  T.MAX  IND.2  T.MIN  T.MIN.G
DATE
1961-03-16   2.75    0   0.0    0.0   18.8    0.0    2.2     -1.2
1961-05-11   3.67    0   0.0    0.0   19.7    0.0    2.2      0.3
1961-05-12   5.17    0   0.0    0.0   19.8    0.0    6.5      1.4
1961-05-13   5.21    0   0.2    0.0   22.1    0.0    5.1      0.1
1961-06-03  10.41    0   0.0    0.0   18.7    0.0   11.7     10.8
...           ...  ...   ...    ...    ...    ...    ...      ...
1978-10-08   7.62    0   1.8    0.0   19.5    0.0   13.9      9.0
```

```
1978-10-09   7.62    0   4.5    0.0   18.4    0.0   13.4     10.2
1978-10-11  11.29    0   0.0    0.0   19.8    0.0   13.5     11.0
1978-10-25  16.17    0   0.0    0.0   18.5    0.0   12.7     10.7
1978-11-04  14.58    4   0.0    0.0   18.3    0.0   10.9      9.6

[1085 rows x 8 columns]
```

4(b)

Let's make simple statistics of the dataset to create a categorical data column. I have chosen the median value to differentiate the four categorical regions. See below my strategy for solving 4(b).

```
# Simple relevant statistics of the wind_predict dataset

wind_median_value = wind_predict['WIND'].median()
rain_median_value = wind_predict['RAIN'].median()
max_temp_median_value = wind_predict['T.MAX'].median()
min_temp_median_value = wind_predict['T.MIN'].median()

print("The median of WIND attribute is", wind_median_value)
print("The median of RAIN attribute is", rain_median_value)
print("The median of T.MAX attribute is", max_temp_median_value)
print("The median of T.MIN attribute is", min_temp_median_value)
```

```
The median of WIND attribute is 9.08
The median of RAIN attribute is 0.2
The median of T.MAX attribute is 13.1
The median of T.MIN attribute is 6.5
```

```
# Create the categorical data column of the Dataset.

def categorical_data_column(row):
    if row['WIND'] > wind_median_value and row['RAIN'] > rain_median_value:
        return 'Windy & Wet'
    elif row['T.MAX'] > max_temp_median_value and row['RAIN'] < rain_median_value:
        return 'Hot & Dry'
    elif row['T.MIN'] < min_temp_median_value and row['WIND'] < wind_median_value:
        return 'Cold & Calm'
    else:
        return 'Moderate'  # This captures everything else that doesn't fall into the first three categories

# Now print the new dataset with a column called ClimateCategory
wind_predict['ClimateCategory'] = wind_predict.apply(categorical_data_column, axis=1)
```

```
# Lets see the first five rows of our new dataset
wind_predict.head(5)
```

DATE	WIND	IND	RAIN	IND.1	T.MAX	IND.2	T.MIN	T.MIN.G	ClimateCategory
1961-01-01	13.67	0	0.2	0.0	9.5	0.0	3.7	-1.0	Moderate
1961-01-02	11.50	0	5.1	0.0	7.2	0.0	4.2	1.1	Windy & Wet
1961-01-03	11.25	0	0.4	0.0	5.5	0.0	0.5	-0.5	Windy & Wet
1961-01-04	8.63	0	0.2	0.0	5.6	0.0	0.4	-3.2	Cold & Calm
1961-01-05	11.92	0	10.4	0.0	7.2	1.0	-1.5	-7.5	Windy & Wet

```
# We have our caterorcal data column called ClimateCategory and now we make a summary by using group method
grouped_data_summary= wind_predict.groupby('ClimateCategory')

# Calculate the mean value for each group here
mean_values_summary = grouped_data_summary[['WIND', 'RAIN', 'T.MAX', 'T.MIN']].mean()

# Calculate the median value for each group here
median_values_summary = grouped_data_summary[['WIND', 'RAIN', 'T.MAX', 'T.MIN']].median()

# Display the results here
print("Summary of Mean values by Climate Category:")
print(mean_values_summary)
print("\nSummary of Median values by Climate Category:")
print(median_values_summary)
```

```
Summary of Mean values by Climate Category:
                      WIND      RAIN      T.MAX     T.MIN
ClimateCategory
Cold & Calm       6.063645  1.469594   9.194051  1.820397
Hot & Dry         7.276959  0.007326  17.904147  9.477471
Moderate          9.735926  1.853704  12.777229  7.175720
Windy & Wet      14.003973  3.766607  12.324731  6.119474

Summary of Median values by Climate Category:
                  WIND  RAIN  T.MAX  T.MIN
ClimateCategory
Cold & Calm       6.38   0.1    9.2    2.2
Hot & Dry         6.54   0.0   17.7    9.9
Moderate          9.08   0.2   12.2    7.5
Windy & Wet      13.17   2.0   12.0    6.0
```

4(c)

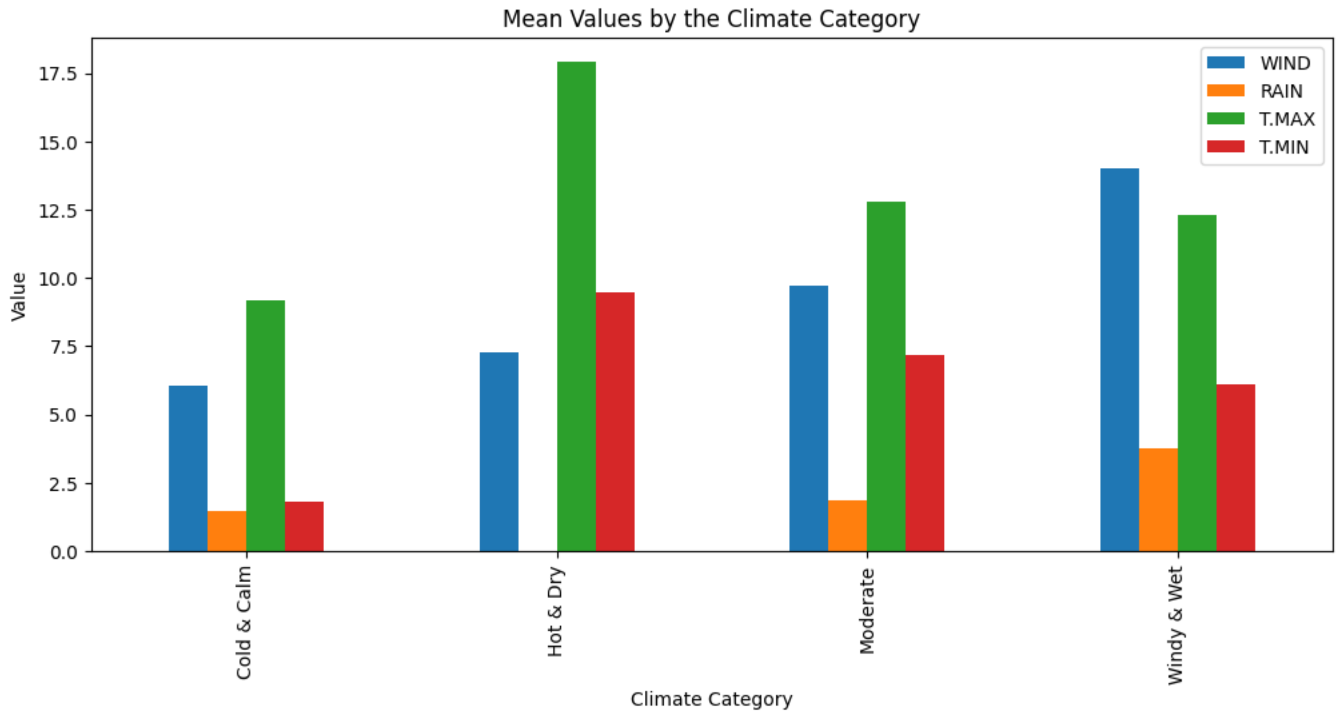
I've grouped data by the categorical feature and calculated summary statistics for the columns 'WIND', 'RAIN', 'T.MAX', and 'T.MIN'. So, I can visualise these summary statistics to better understand the differences between each region (or category).

Mean Values by using Bar Plots:

Bar plots can be used to compare the mean values of each feature across different situations. The below plot by Climate Category column of the dataset simply indicates the mean value of 'WIND', 'RAIN', 'T.MAX', and 'T.MIN' features.

```
# Plotting mean values for each feature by using four diffrent situations
mean_values_summary = grouped_data_summary[['WIND', 'RAIN', 'T.MAX', 'T.MIN']].mean()

#, figsize=(10, 6)
mean_values_summary.plot(kind='bar', figsize=(12, 5))
plt.title("Mean Values by the Climate Category")
plt.ylabel("Value")
```

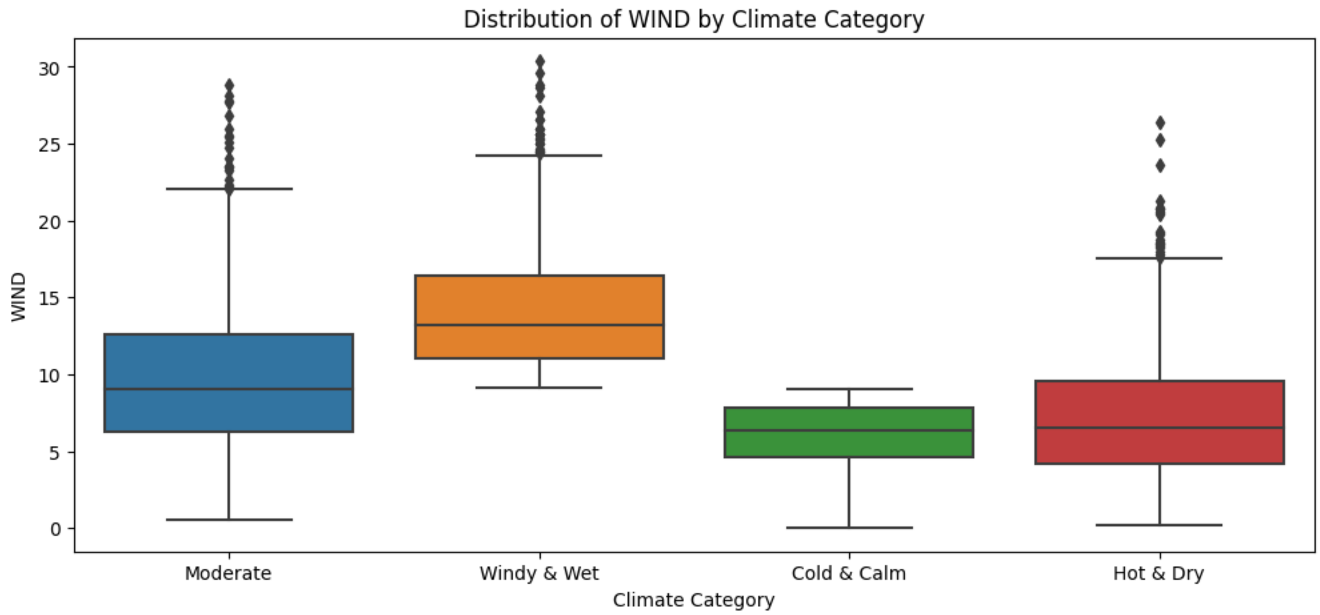
```
plt.xlabel("Climate Category")
plt.show()
```



Box plots:

Box plots are helpful to describe the distribution of data and can give insights into the median, quartiles, and potential outliers of each feature for each region. Let me explain a little bit about the box plots in the below cell. The medians of the WIND attribute in the box plots are 9.08 for Moderate, 13.17 for Windy & Wet, 6.38 for Windy & Calm, and 6.54 for Hot & Dry. The middle solid black line is describing the median value. Similarly, we can describe the quartiles of the box plots, which I have already done in my previous work on this assignment.

```
# Box plot for WIND by ClimateCategory
plt.figure(figsize=(12, 5))
sns.boxplot(x='ClimateCategory', y='WIND', data=wind_predict)
plt.title("Distribution of WIND by Climate Category")
plt.ylabel("WIND")
plt.xlabel("Climate Category")
#plt.tight_layout()
plt.show()
```



3D Plots:

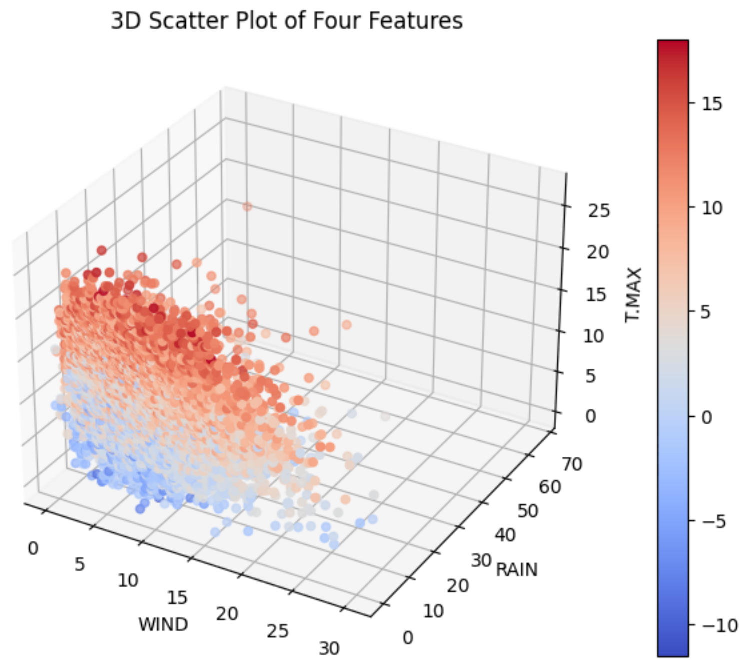
The line from mpl_toolkits.mplot3d import Axes3D is an import declaration in Python that brings in the Axes3D class from the mpl_toolkits.mplot3d module. This class is a component of Matplotlib, a plotting toolkit for Python, and was created specifically to handle 3D charts. I like to use this module when I need 3D plots with multiple variables.

So, what is the meaning of the 3D plot?

3D plots are graphical displays of data on three axes (x, y, and z) to understand and visualise data in three dimensions. These plots can be particularly useful when we want to show the relationships between four features simultaneously and visualise the relationship between numerical variables. We can identify patterns, clusters, or groupings among data points. Above is a scatter type of 3D plot, especially considering the features 'WIND', 'RAIN', 'T.MAX', and 'T.MIN'.

Here, I have set 'WIND' on the x-axis, 'RAIN' on the y-axis, and 'T.MAX' on the z-axis, where 'T.MIN' is a colour bar that is used to visualise the relationships among wind speed, rain amount, and maximum temperature by colouring.

```
# import module Axes3D
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(12, 5))
ax = fig.add_subplot(111, projection='3d')

p = ax.scatter(wind_predict['WIND'], wind_predict['RAIN'], wind_predict['T.MAX'], c=wind_predict['T.MIN'], cmap='coolwarm')
fig.colorbar(p)

ax.set_xlabel('WIND')
ax.set_ylabel('RAIN')
ax.set_zlabel('T.MAX')
ax.set_title('3D Scatter Plot of Four Features')
plt.show()
```


3D Scatter Plot of Four Features
WIND
RAIN
T.MAX
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70
0
5
10
15
20
25
−10
−5
0
5
10
15

#***