Assignment - 1

Name - Sayali Pawar

PRN - 123B1B229

DIV - D

PROBLEM STATEMENT:

Pre Processing Techniques: Create a dummy dataset or with missing values and duplicate entries or select any data set with missing values (such as Iris dataset, breast cancer dataset) from any repository of data such as SK-Learn, UCI library, Kaggle dataset library etc. Write a program or use a suitable tool to perform the following operations on the selected dataset and display the result.

1. Removal of duplicates
2. Handle missing values
3. Normalizing the data using normalizing technique
4. Apply min-max scalar / Robust scalar / standard scalar to scale the data
5. Use measures of Central Tendency and Dispersion of Data

1. Importing Libraries and Loading Dataset

```
# Importing libraries
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
# Load the dataset
data = pd.read_csv("/content/drive/MyDrive/TY_Machine_Learning/Assignment 1/weatherHistory.csv", delimiter=",")
print("Original shape:", data.shape)
```

```
Original shape: (96453, 12)
```

2. Initial Data Exploration

```
data.head()
```

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout the day.
2	2006-04-01 02:00:00.000	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96453 entries, 0 to 96452
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Formatted Date            96453 non-null  object
 1   Summary                   96453 non-null  object
```

```
 2   Precip Type              95936 non-null  object
 3   Temperature (C)          96453 non-null  float64
 4   Apparent Temperature (C) 96453 non-null  float64
 5   Humidity                 96453 non-null  float64
 6   Wind Speed (km/h)        96453 non-null  float64
 7   Wind Bearing (degrees)   96453 non-null  float64
 8   Visibility (km)          96453 non-null  float64
 9   Loud Cover               96453 non-null  float64
 10  Pressure (millibars)     96453 non-null  float64
 11  Daily Summary            96453 non-null  object
dtypes: float64(8), object(4)
memory usage: 8.8+ MB
```

```
data.describe()
```

	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)
count	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.000000	96453.0	96453.000000
mean	11.932678	10.855029	0.734899	10.810640	187.509232	10.347325	0.0	1003.235956
std	9.551546	10.696847	0.195473	6.913571	107.383428	4.192123	0.0	116.969906
min	-21.822222	-27.716667	0.000000	0.000000	0.000000	0.000000	0.0	0.000000
25%	4.688889	2.311111	0.600000	5.828200	116.000000	8.339800	0.0	1011.900000
50%	12.000000	12.000000	0.780000	9.965900	180.000000	10.046400	0.0	1016.450000
75%	18.838889	18.838889	0.890000	14.135800	290.000000	14.812000	0.0	1021.090000

```
data.dtypes
```

	0
Formatted Date	object
Summary	object
Precip Type	object
Temperature (C)	float64
Apparent Temperature (C)	float64
Humidity	float64
Wind Speed (km/h)	float64
Wind Bearing (degrees)	float64
Visibility (km)	float64
Loud Cover	float64
Pressure (millibars)	float64
Daily Summary	object

dtype: object

3. Data Cleaning and Normalizing

```
# 3.1 Remove Duplicates
duplicate = data[data.duplicated()]
print(duplicate.count())
```

```
Formatted Date              24
Summary                     24
Precip Type                 24
Temperature (C)             24
Apparent Temperature (C)    24
Humidity                    24
Wind Speed (km/h)           24
Wind Bearing (degrees)      24
Visibility (km)             24
Loud Cover                  24
Pressure (millibars)        24
Daily Summary               24
```

```
dtype: int64
```

```
data.drop_duplicates(keep=False, inplace=True)
data.shape
```

```
(96405, 12)
```

```
# 3.2 Replacing missing values
data.isnull()
```

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...
96448	False	False	False	False	False	False	False	False	False	False	False	False
96449	False	False	False	False	False	False	False	False	False	False	False	False
96450	False	False	False	False	False	False	False	False	False	False	False	False
96451	False	False	False	False	False	False	False	False	False	False	False	False
96452	False	False	False	False	False	False	False	False	False	False	False	False

```
data.isnull().sum()
```

	0
Formatted Date	0
Summary	0
Precip Type	517
Temperature (C)	0
Apparent Temperature (C)	0
Humidity	0
Wind Speed (km/h)	0
Wind Bearing (degrees)	0
Visibility (km)	0
Loud Cover	0
Pressure (millibars)	0
Daily Summary	0

dtype: int64

```
#Separate numeric and categorical columns
num_cols = data.select_dtypes(include=[np.number]).columns
cat_cols = data.select_dtypes(exclude=[np.number]).columns

# Impute missing values
# --- numeric: median ---
from sklearn.impute import SimpleImputer
imputer_median = SimpleImputer(strategy="median")
data[num_cols] = imputer_median.fit_transform(data[num_cols])

# --- categorical: most frequent ---
imputer_freq = SimpleImputer(strategy="most_frequent")
data[cat_cols] = imputer_freq.fit_transform(data[cat_cols])
```

```
print("Missing values after imputation:")
print(data.isnull().sum())
```

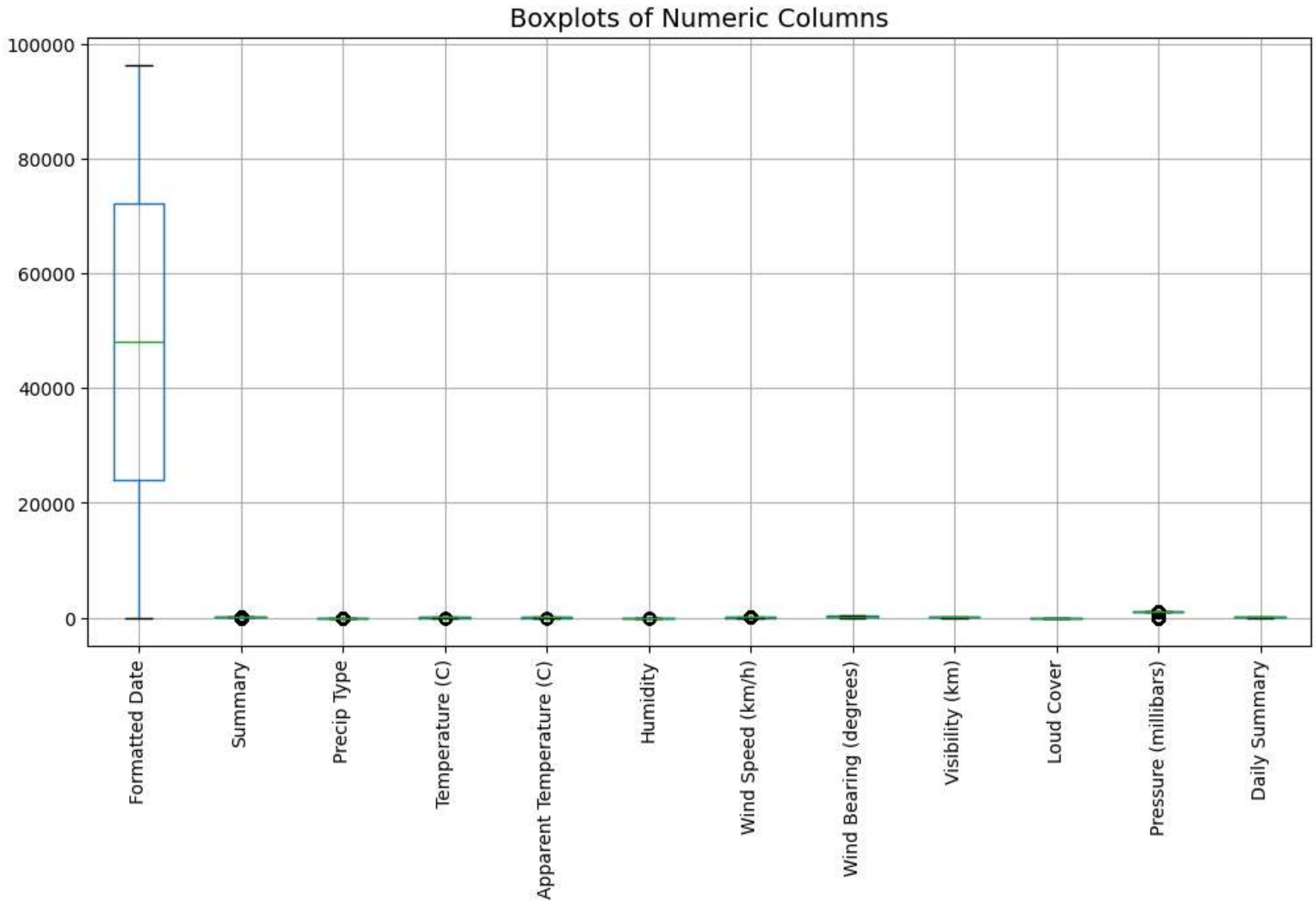
```
Missing values after imputation:
Formatted Date             0
Summary                    0
Precip Type                0
Temperature (C)            0
Apparent Temperature (C)   0
Humidity                   0
Wind Speed (km/h)          0
Wind Bearing (degrees)     0
Visibility (km)            0
Loud Cover                 0
Pressure (millibars)       0
Daily Summary              0
dtype: int64
```

```
# 3.3 Encode categorical columns with LabelEncoder
from sklearn.preprocessing import LabelEncoder
label_encoders = {}
for col in cat_cols:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le   # save encoders in case you need inverse transform

print("\nFirst 5 rows after Label Encoding:")
print(data.head())
```

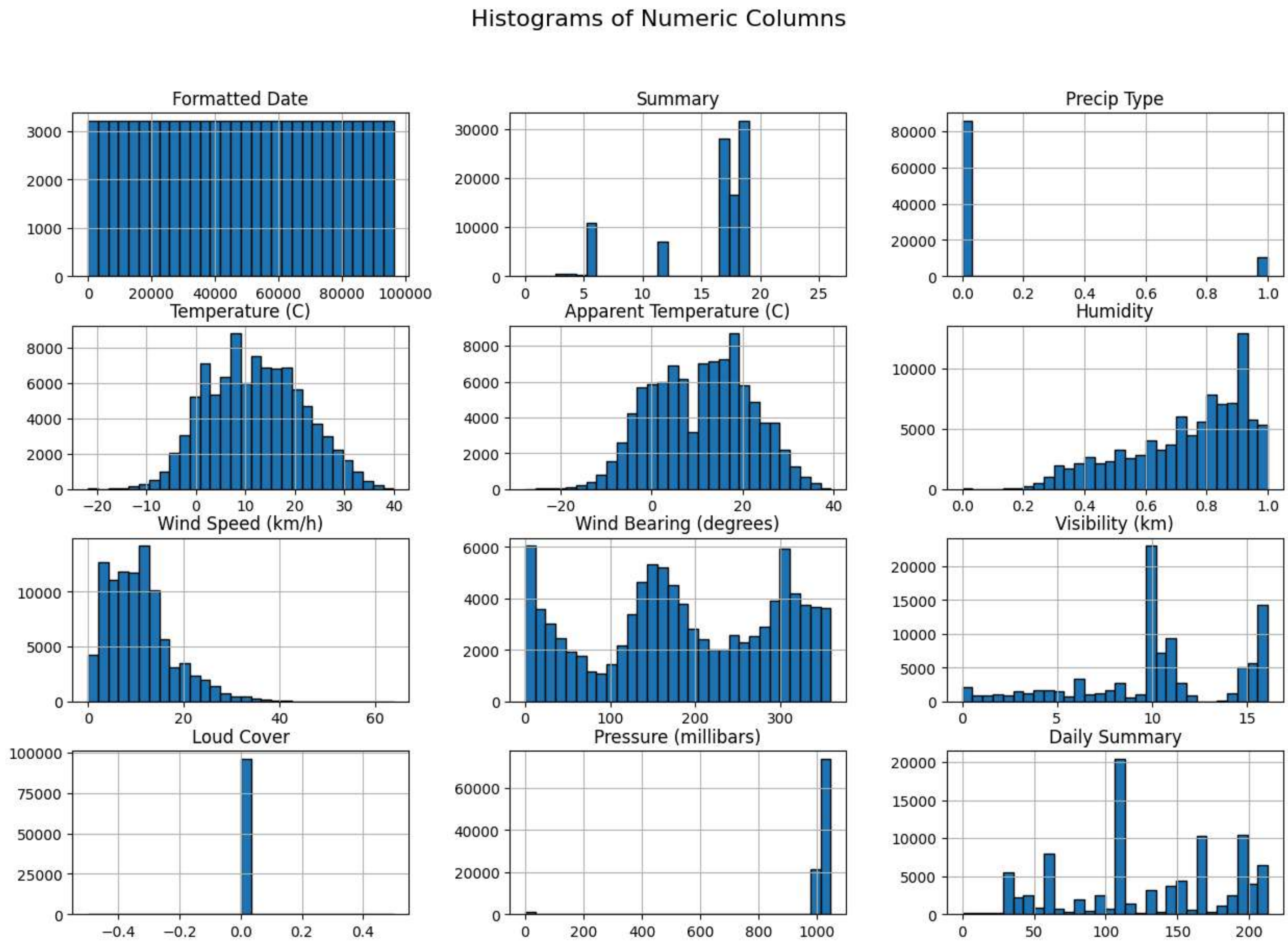
```
First 5 rows after Label Encoding:
   Formatted Date  Summary  Precip Type  Temperature (C)  \
0            2159       19            0         9.472222
1            2160       19            0         9.355556
2            2161       17            0         9.377778
3            2162       19            0         8.288889
4            2163       17            0         8.755556

   Apparent Temperature (C)  Humidity  Wind Speed (km/h)  \
0                  7.388889      0.89            14.1197
1                  7.227778      0.86            14.2646
2                  9.377778      0.89             3.9284
3                  5.944444      0.83            14.1036
4                  6.977778      0.83            11.0446

   Wind Bearing (degrees)  Visibility (km)  Loud Cover  Pressure (millibars)  \
0                   251.0          15.8263         0.0               1015.13
1                   259.0          15.8263         0.0               1015.63
2                   204.0          14.9569         0.0               1015.94
3                   269.0          15.8263         0.0               1016.41
4                   259.0          15.8263         0.0               1016.51

   Daily Summary
0            197
1            197
2            197
3            197
4            197
```

4. Exploratory Data Analysis (EDA)

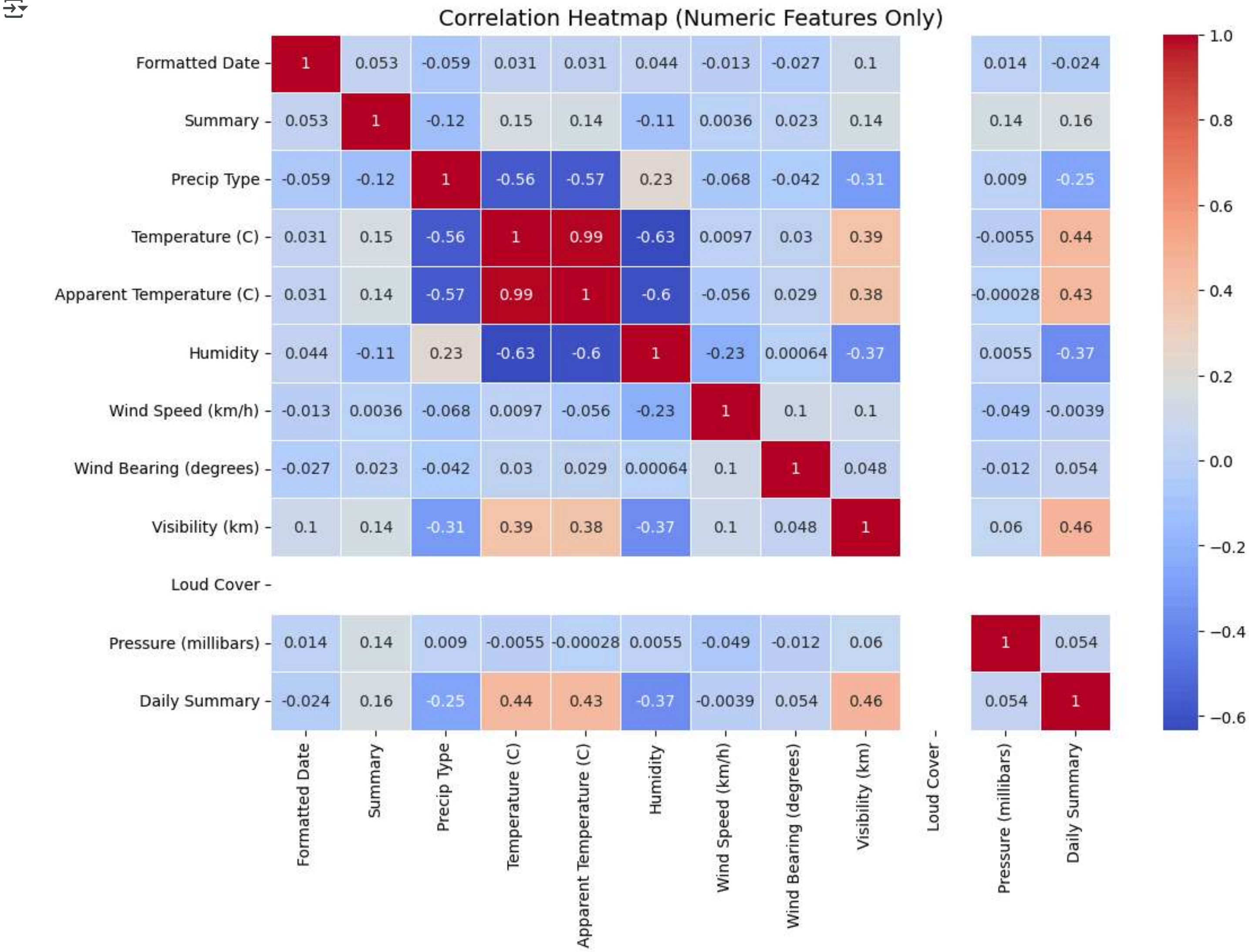
```
# 1) Boxplot for numeric columns
# -------------------------------
plt.figure(figsize=(12,6))
data.boxplot(rot=90)
plt.title("Boxplots of Numeric Columns", fontsize=14)
plt.show()
```



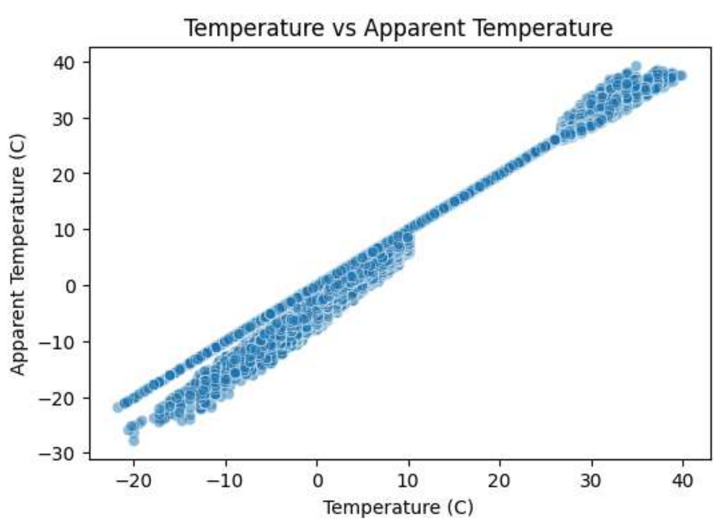

```
# 2) Histogram for numeric columns
# -------------------------------
data.hist(figsize=(15,10), bins=30, edgecolor="black")
plt.suptitle("Histograms of Numeric Columns", fontsize=16)
plt.show()
```




```
# 3) Heatmap of correlations
import matplotlib.pyplot as plt
import seaborn as sns

# Select only numeric columns
numeric_data = data.select_dtypes(include=['int64', 'float64'])

# Compute correlation
corr = numeric_data.corr()

# Plot heatmap
plt.figure(figsize=(12,8))
sns.heatmap(corr, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Correlation Heatmap (Numeric Features Only)", fontsize=14)
plt.show()
```

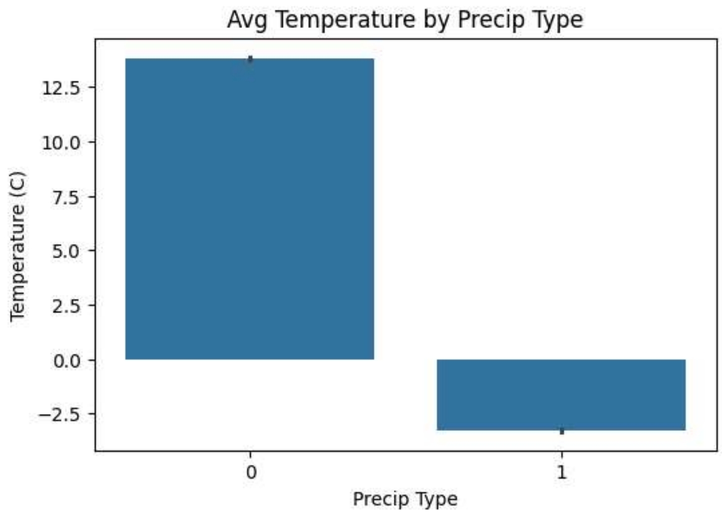



```
# 4) Scatter plot: Temperature vs Apparent Temperature
plt.figure(figsize=(6,4))
sns.scatterplot(x="Temperature (C)", y="Apparent Temperature (C)", data=data, alpha=0.5)
plt.title("Temperature vs Apparent Temperature")
plt.show()
```



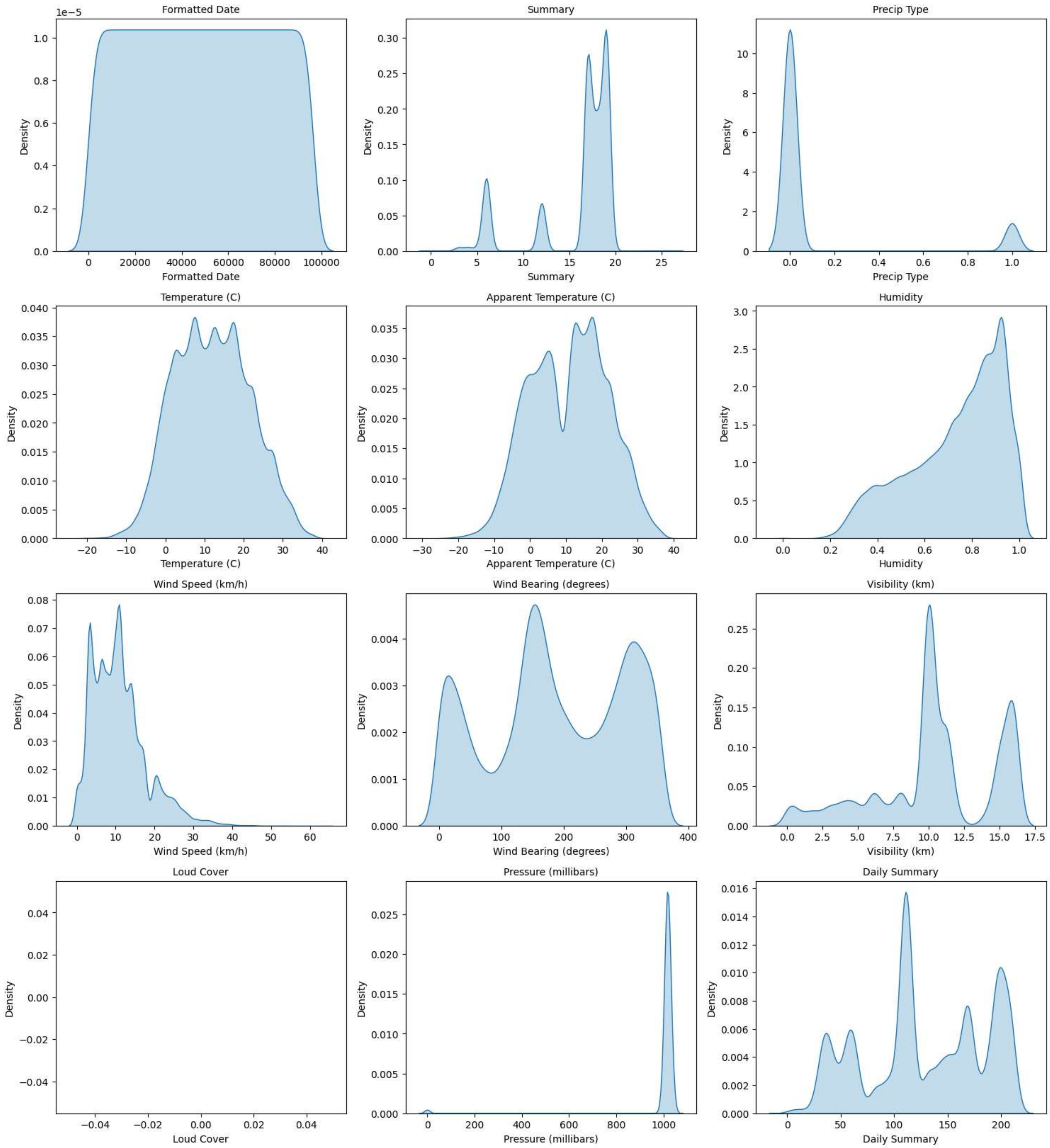

```
# 5) Barplot: Average Temperature per Precip Type
plt.figure(figsize=(6,4))
```

```
sns.barplot(x="Precip Type", y="Temperature (C)", data=data)
plt.title("Avg Temperature by Precip Type")
plt.show()
```

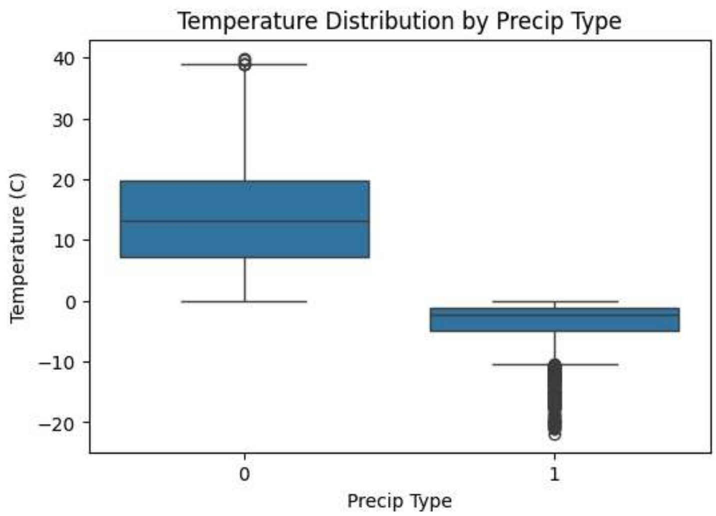



```
# 6) KDE (density) plots for numeric variables
# Select numeric columns
numeric_cols = data.select_dtypes(include=['int64','float64']).columns

# Set up subplot grid
n = len(numeric_cols)
rows = (n // 3) + 1   # 3 plots per row
cols = 3

plt.figure(figsize=(15, 4*rows))

for i, col in enumerate(numeric_cols, 1):
    plt.subplot(rows, cols, i)
    sns.kdeplot(data[col], fill=True)
    plt.title(col, fontsize=10)

plt.tight_layout()
plt.show()
```

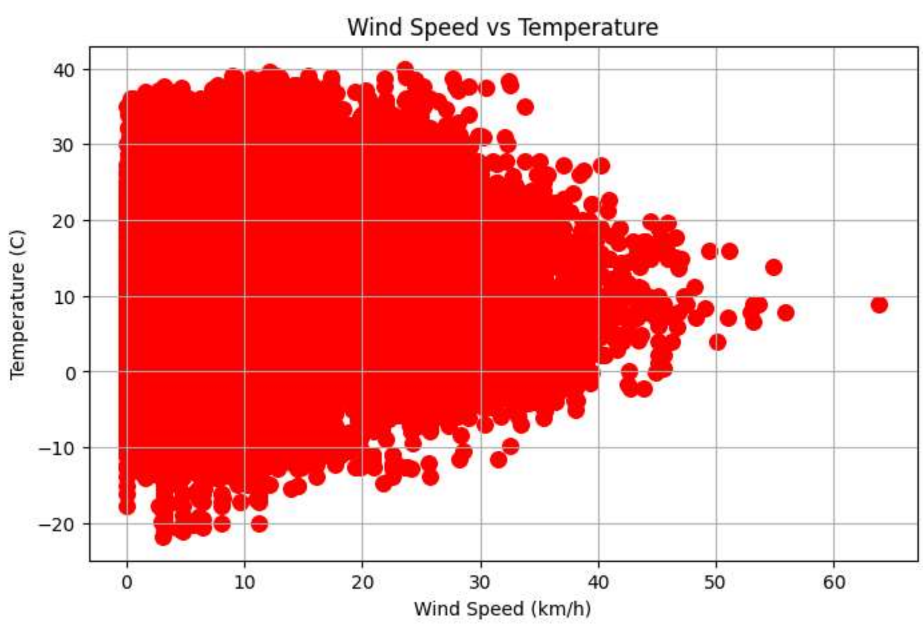
```
/tmp/ipython-input-2831935248.py:14: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_singular=False` to disat
  sns.kdeplot(data[col], fill=True)
```



```
# 7) Boxplot: Temperature by Precip Type
plt.figure(figsize=(6,4))
sns.boxplot(x="Precip Type", y="Temperature (C)", data=data)
plt.title("Temperature Distribution by Precip Type")
plt.show()
```

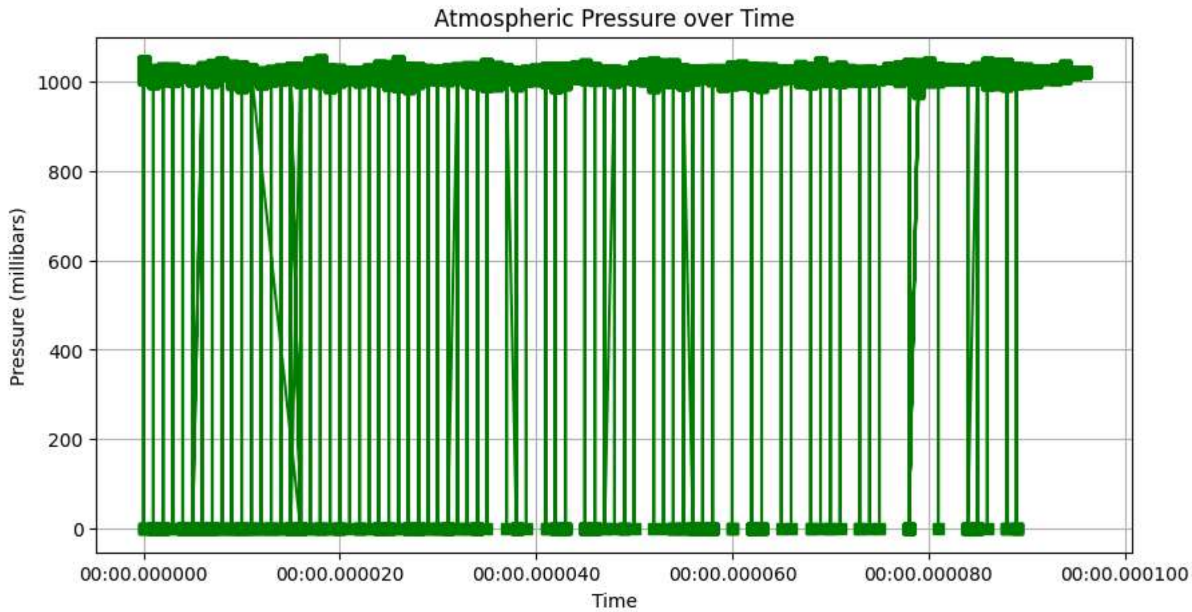



```
# 8) Scatter plot: Wind Speed vs Temperature
plt.figure(figsize=(8, 5))
plt.scatter(data["Wind Speed (km/h)"], data["Temperature (C)"], c="red", s=70)
plt.xlabel("Wind Speed (km/h)")
plt.ylabel("Temperature (C)")
plt.title("Wind Speed vs Temperature")
plt.grid(True)
plt.show()
```

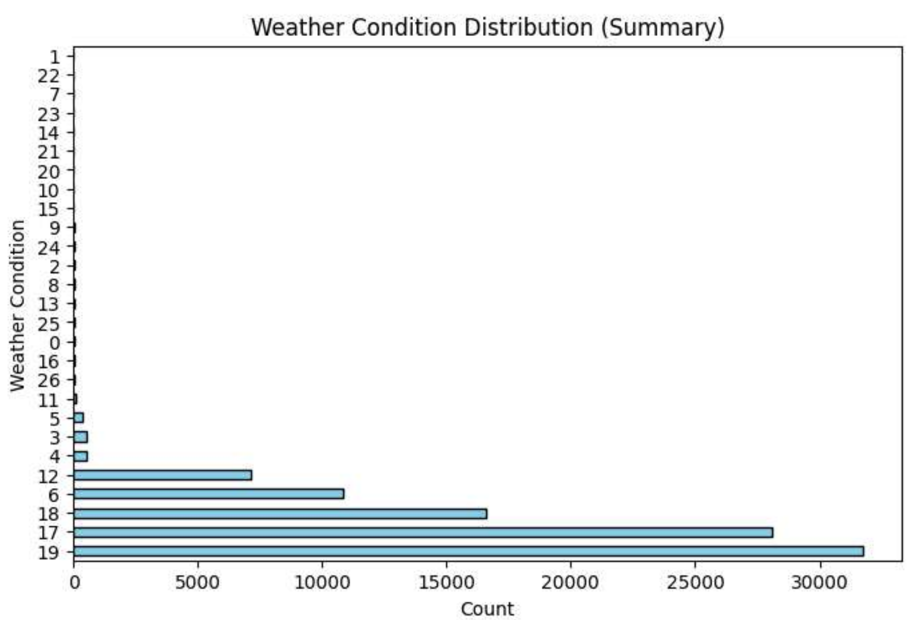



```
# 9) Line chart: Pressure over Time
plt.figure(figsize=(10, 5))
```

```
plt.plot(data["Formatted Date"], data["Pressure (millibars)"], marker="s", color="green")
plt.xlabel("Time")
plt.ylabel("Pressure (millibars)")
plt.title("Atmospheric Pressure over Time")
plt.grid(True)
plt.show()
```




```
# 5. Pie chart: Weather summary distribution
plt.figure(figsize=(8, 5))
data["Summary"].value_counts().plot.barh(color="skyblue", edgecolor="black")
plt.xlabel("Count")
plt.ylabel("Weather Condition")
plt.title("Weather Condition Distribution (Summary)")
plt.show()
```

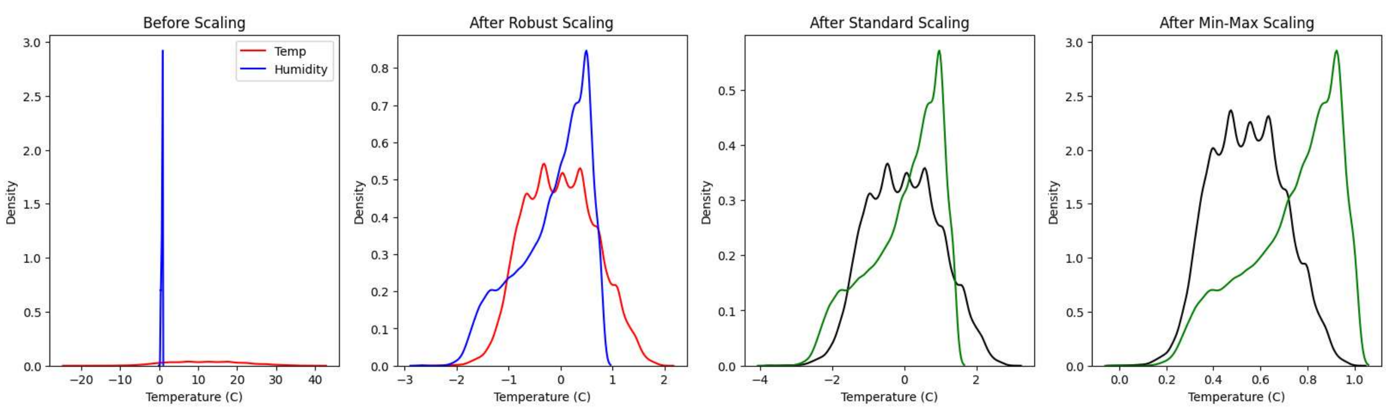


5. Apply scaling on data

```
 # [1]- Select two numeric columns from weatherHistory
x = data[['Temperature (C)', 'Humidity']]
type(x)
```

```
pandas.core.frame.DataFrame
def __init__(data=None, index: Axes | None=None, columns: Axes | None=None, dtype: Dtype |
None=None, copy: bool | None=None) -> None

Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns).
Arithmetic operations align on both row and column labels. Can be
thought of as a dict-like container for Series objects. The primary
pandas data structure.
```

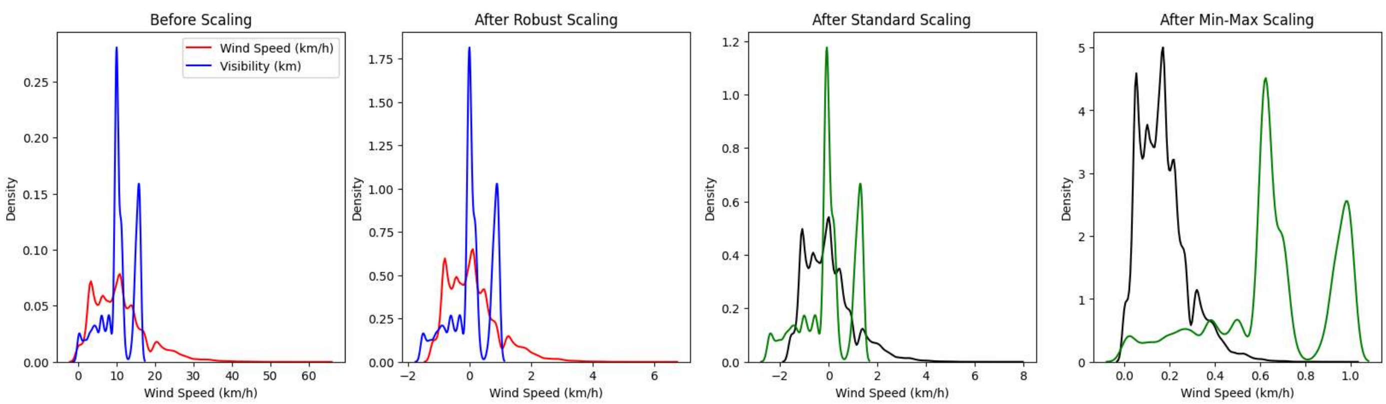
```
# Robust Scaling
scaler = preprocessing.RobustScaler()
robust_df = scaler.fit_transform(x)
robust_df = pd.DataFrame(robust_df, columns=['Temperature (C)', 'Humidity'])

# Standard Scaling
scaler = preprocessing.StandardScaler()
standard_df = scaler.fit_transform(x)
standard_df = pd.DataFrame(standard_df, columns=['Temperature (C)', 'Humidity'])

# Min-Max Scaling
scaler = preprocessing.MinMaxScaler()
minmax_df = scaler.fit_transform(x)
minmax_df = pd.DataFrame(minmax_df, columns=['Temperature (C)', 'Humidity'])


fig, (ax1, ax2, ax3, ax4) = plt.subplots(ncols=4, figsize=(20, 5))

ax1.set_title('Before Scaling')
sns.kdeplot(x['Temperature (C)'], ax=ax1, color='r', label="Temp")
sns.kdeplot(x['Humidity'], ax=ax1, color='b', label="Humidity")
ax1.legend()

ax2.set_title('After Robust Scaling')
sns.kdeplot(robust_df['Temperature (C)'], ax=ax2, color='red')
sns.kdeplot(robust_df['Humidity'], ax=ax2, color='blue')

ax3.set_title('After Standard Scaling')
sns.kdeplot(standard_df['Temperature (C)'], ax=ax3, color='black')
sns.kdeplot(standard_df['Humidity'], ax=ax3, color='g')

ax4.set_title('After Min-Max Scaling')
sns.kdeplot(minmax_df['Temperature (C)'], ax=ax4, color='black')
sns.kdeplot(minmax_df['Humidity'], ax=ax4, color='g')

plt.show()
```




```
 # [2]- Select two numeric columns from weatherHistory
x = data[['Wind Speed (km/h)', 'Visibility (km)']]
type(x)
```

```
pandas.core.frame.DataFrame
def __init__(data=None, index: Axes | None=None, columns: Axes | None=None, dtype: Dtype |
None=None, copy: bool | None=None) -> None
dtype: object

To enforce a single dtype:

>>> df = pd.DataFrame(data=d, dtype=np.int8)
>>> df.dtypes
col1    int8
```

```
# Robust Scaling
scaler = preprocessing.RobustScaler()
robust_df = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)

# Standard Scaling
scaler = preprocessing.StandardScaler()
standard_df = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)

# Min-Max Scaling
scaler = preprocessing.MinMaxScaler()
minmax_df = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)



fig, (ax1, ax2, ax3, ax4) = plt.subplots(ncols=4, figsize=(20, 5))

ax1.set_title('Before Scaling')
sns.kdeplot(x.iloc[:,0], ax=ax1, color='r', label=x.columns[0])
sns.kdeplot(x.iloc[:,1], ax=ax1, color='b', label=x.columns[1])
ax1.legend()

ax2.set_title('After Robust Scaling')
sns.kdeplot(robust_df.iloc[:,0], ax=ax2, color='red')
sns.kdeplot(robust_df.iloc[:,1], ax=ax2, color='blue')

ax3.set_title('After Standard Scaling')
sns.kdeplot(standard_df.iloc[:,0], ax=ax3, color='black')
sns.kdeplot(standard_df.iloc[:,1], ax=ax3, color='g')

ax4.set_title('After Min-Max Scaling')
sns.kdeplot(minmax_df.iloc[:,0], ax=ax4, color='black')
sns.kdeplot(minmax_df.iloc[:,1], ax=ax4, color='g')

plt.show()
```




```
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt

# VarianceThreshold with threshold=0.0
```