## Display the first few rows of each dataframe to understand the structure

```
import pandas as pd

# Load all the uploaded CSV files
file_paths = {
    'deliveries_data': '/content/deliveries_data.csv',
    'food_delivery_weather_data': '/content/food_delivery_weather_data.csv',
    'food_delivery_weather_data_with_time': '/content/food_delivery_weather_data_with_time.csv',
    'toronto_food_delivery_2021_2023': '/content/toronto_food_delivery_2021_2023.csv',
    'toronto_food_delivery_2021_2023_with_counts': '/content/toronto_food_delivery_2021_2023_with_counts.csv',
    'toronto_weather_2021_2022': '/content/toronto_weather_2021_2022.csv',
    'toronto_weather_2021_2022_with_location': '/content/toronto_weather_2021_2022_with_location.csv',
    'weather_data': '/content/weather_data.csv'
}

# Load the CSV files
dfs = {name: pd.read_csv(path) for name, path in file_paths.items()}

# Display the first few rows of each dataframe to understand the structure
dfs_overview = {name: df.head() for name, df in dfs.items()}
dfs_overview
```

```
0  2021-01-01       Taco Bell     Fast Food           7.794540
1  2021-01-02         Dominos        Indian           8.336749
2  2021-01-03       Starbucks       Italian           5.238387
3  2021-01-04     Burger King      Japanese           7.232606
4  2021-01-05       Taco Bell      American           4.833890

   Total Cost (CAD)  Delivery Time (Minutes)  Customer Rating Payment Method  \
0         35.893632                38.176126         3.817341           Cash
1         49.234974                51.207571         4.714388      Gift Card
2         91.044638                59.904913         3.954974           Cash
3         15.037636                27.645284         2.225072    Credit Card
4         91.617563                38.386033         2.060325     Debit Card

   Delivery Count Location
0             154  Toronto
1             131  Toronto
2             116  Toronto
3              59  Toronto
4              98  Toronto  ,
'toronto_weather_2021_2022':          Date  Temperature_High_C  Temperature_Low_C  Precipitation_mm  \
0  2021-01-01           6.854305           0.211647         7.877288
1  2021-01-02          32.782144          19.636397        13.024660
2  2021-01-03          22.939727          11.287755         2.131861
3  2021-01-04          16.939632           6.708978        13.156906
4  2021-01-05          -2.979161         -11.567466        19.988275

   Humidity_%  Wind_Speed_kmh
0   55.617211       13.132378
1   94.431702       22.815533
2   80.243939       25.419487
3   73.626438        9.540349
4   46.659368       14.273525  ,
'toronto_weather_2021_2022_with_location':          Date Location  Temperature_High_C  Temperature_Low_C  \
0  2021-01-01  Toronto           6.854305           0.211647
1  2021-01-02  Toronto          32.782144          19.636397
2  2021-01-03  Toronto          22.939727          11.287755
3  2021-01-04  Toronto          16.939632           6.708978
4  2021-01-05  Toronto          -2.979161         -11.567466

   Precipitation_mm  Humidity_%  Wind_Speed_kmh
0          7.877288   55.617211       13.132378
1         13.024660   94.431702       22.815533
2          2.131861   80.243939       25.419487
3         13.156906   73.626438        9.540349
4         19.988275   46.659368       14.273525  ,
'weather_data':           Date_Time  Temperature_High_C  Temperature_Low_C  Precipitation_mm  \
0   2021-01-01 8:03            6.854305           0.211647          7.877288
1  2021-01-02 11:06           32.782144          19.636397         13.024660
2  2021-01-03 18:16           22.939727          11.287755          2.131861
3  2021-01-04 20:20           16.939632           6.708978         13.156906
4  2021-01-05 21:06           -2.979161         -11.567466         19.988275

   Humidity_%  Wind_Speed_kmh Location
0   55.617211       13.132378  Toronto
1   94.431702       22.815533  Toronto
2   80.243939       25.419487  Toronto
3   73.626438        9.540349  Toronto
4   46.659368       14.273525  Toronto  }
```

## Analytics type of each attribute

```
# Check for missing values in the datasets before merging
missing_values = {name: df.isnull().sum() for name, df in dfs.items()}

missing_values
```

```
Precipitation_mm     0
WindSpeed_kmh        0
Weather_Condition    0
Delivery_Orders      0
dtype: int64,
'food_delivery_weather_data_with_time': Date                 0
Temperature_C        0
Precipitation_mm     0
WindSpeed_kmh        0
Weather_Condition    0
Delivery_Orders      0
Time_of_Day          0
Day_of_Week          0
dtype: int64,
'toronto_food_delivery_2021_2023': Date                      0
Restaurant Name            0
Food Category              0
Delivery Fee (CAD)         0
Total Cost (CAD)           0
Delivery Time (Minutes)    0
Customer Rating            0
Payment Method             0
Location                   0
dtype: int64,
'toronto_food_delivery_2021_2023_with_counts': Date                       0
Restaurant Name            0
Food Category              0
Delivery Fee (CAD)         0
Total Cost (CAD)           0
Delivery Time (Minutes)    0
Customer Rating            0
Payment Method             0
Delivery Count             0
Location                   0
dtype: int64,
'toronto_weather_2021_2022': Date                  0
Temperature_High_C    0
Temperature_Low_C     0
Precipitation_mm      0
Humidity_%            0
Wind_Speed_kmh        0
dtype: int64,
'toronto_weather_2021_2022_with_location': Date                  0
Location              0
Temperature_High_C    0
Temperature_Low_C     0
Precipitation_mm      0
Humidity_%            0
Wind_Speed_kmh        0
dtype: int64,
'weather_data': Date_Time             0
Temperature_High_C    0
Temperature_Low_C     0
Precipitation_mm      0
Humidity_%            0
Wind_Speed_kmh        0
Location              0
dtype: int64}
```

## Merging the deliveries data with weather data on Date and Location

```
import pandas as pd

# Load the datasets
deliveries_data = pd.read_csv('/content/deliveries_data.csv')
weather_data = pd.read_csv('/content/weather_data.csv')

# Convert 'Date _Time' and 'Date_Time' to datetime format if not already in datetime
deliveries_data['Date _Time'] = pd.to_datetime(deliveries_data['Date _Time'], errors='coerce')
weather_data['Date_Time'] = pd.to_datetime(weather_data['Date_Time'], errors='coerce')

# Extract only the date part from datetime columns
deliveries_data['Date'] = deliveries_data['Date _Time'].dt.date
weather_data['Date'] = weather_data['Date_Time'].dt.date
```

```
# Merging the deliveries data with weather data on Date and Location
merged_data_by_date = pd.merge(deliveries_data, weather_data, on=['Date', 'Location'], how='inner')

# Saving the merged data to a new CSV file
merged_data_by_date.to_csv('/content/merged_deliveries_weather_data.csv', index=False)

# Displaying the first few rows of the merged dataset
merged_data_by_date.head()
```

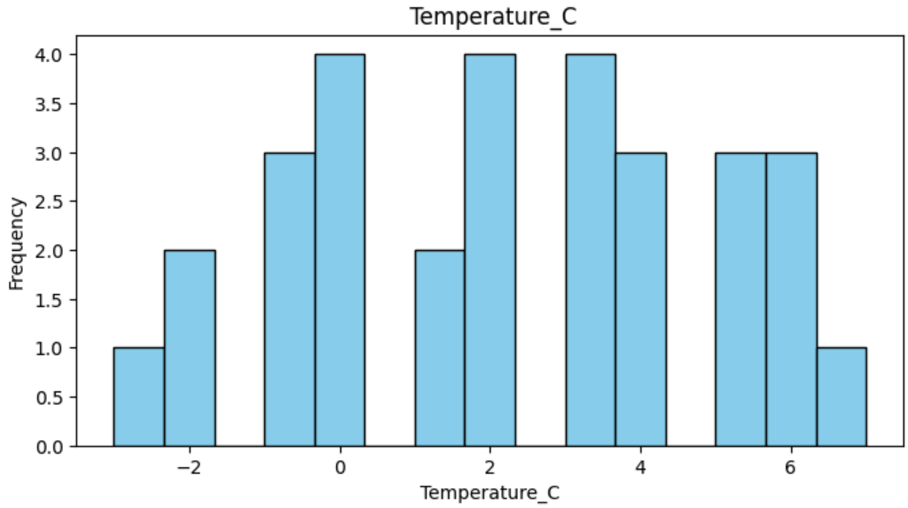
| | Date _Time | Restaurant Name | Food Category | Delivery Fee (CAD) | Total Cost (CAD) | Delivery Time (Minutes) | Customer Rating | Payment Method | Delivery Count | Location | Date | Date_Time | Tempera |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2021-01-01 18:26:00 | McDonalds | Mediterranean | 4.099880 | 35.247802 | 52.985054 | 4.526698 | Debit Card | 92 | Toronto | 2021-01-01 | 2021-01-01 08:03:00 | |
| 1 | 2021-01-02 00:55:00 | McDonalds | Mediterranean | 6.649724 | 48.037354 | 44.334002 | 2.578640 | Gift Card | 56 | Toronto | 2021-01-02 | 2021-01-02 11:06:00 | |
| 2 | 2021-01-03 17:25:00 | Burger King | Vegan | 7.745344 | 52.714623 | 37.774088 | 3.746297 | Debit Card | 141 | Toronto | 2021-01-03 | 2021-01-03 18:16:00 | |
| 3 | 2021-01-04 09:39:00 | Burger King | Indian | 2.995375 | 90.761685 | 57.759836 | 2.666369 | PayPal | 132 | Toronto | 2021-01-04 | 2021-01-04 20:20:00 | |
| 4 | 2021-01-05 03:04:00 | Taco Bell | Italian | 6.796778 | 49.963075 | 21.913553 | 3.594108 | Debit Card | 133 | Toronto | 2021-01-05 | 2021-01-05 21:06:00 | |

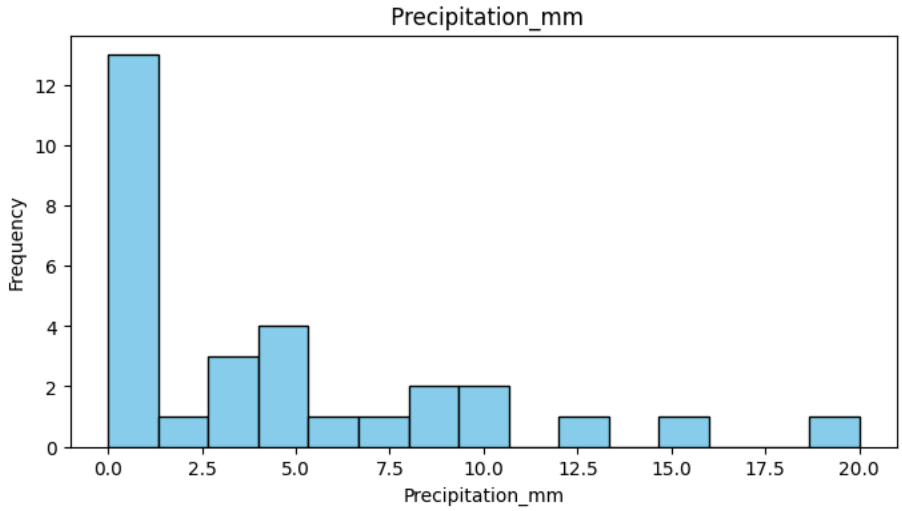
Next steps: Generate code with merged_data_by_date | View recommended plots | New interactive sheet

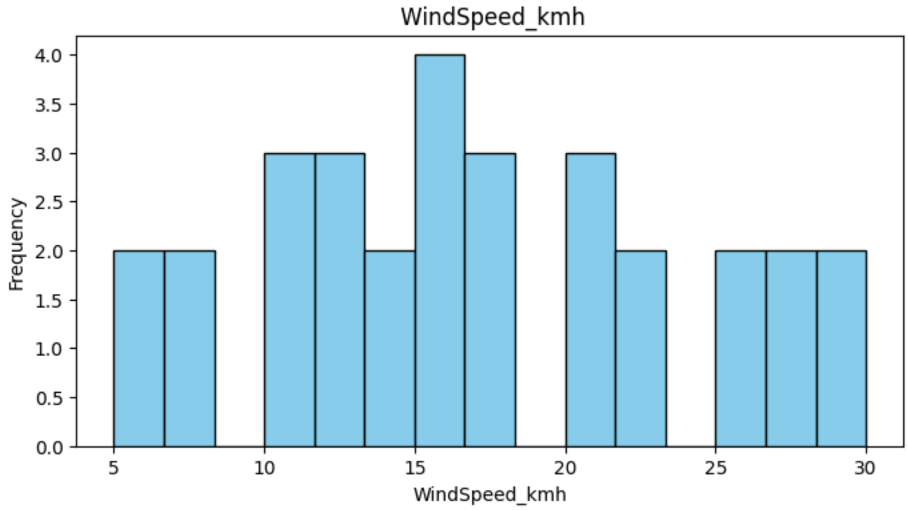
## Plotting Attributes:

### Histograms show the distribution of each numerical attribute.

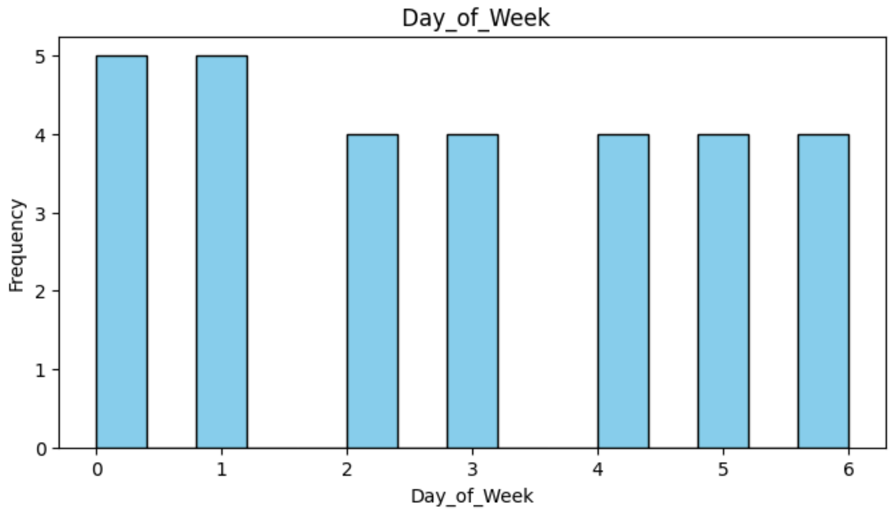
```
import matplotlib.pyplot as plt
import pandas as pd

data = pd.read_csv('food_delivery_weather_data_with_time.csv')
numerical_attributes = data.select_dtypes(include=[np.number]).columns.tolist()

# Histogram for each numerical attribute
for col in numerical_attributes:
    plt.figure(figsize=(8, 4))
    plt.hist(data[col].dropna(), bins=15, color='skyblue', edgecolor='black')
    plt.title(f' {col}')
    plt.xlabel(col)  # X-axis: Attribute values
    plt.ylabel('Frequency')  # Y-axis: Frequency of occurrences
    plt.show()
```


Temperature_C
Frequency
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2
0
2
4
6
Temperature_C


Precipitation_mm
Frequency
12
10
8
6
4
2
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Precipitation_mm

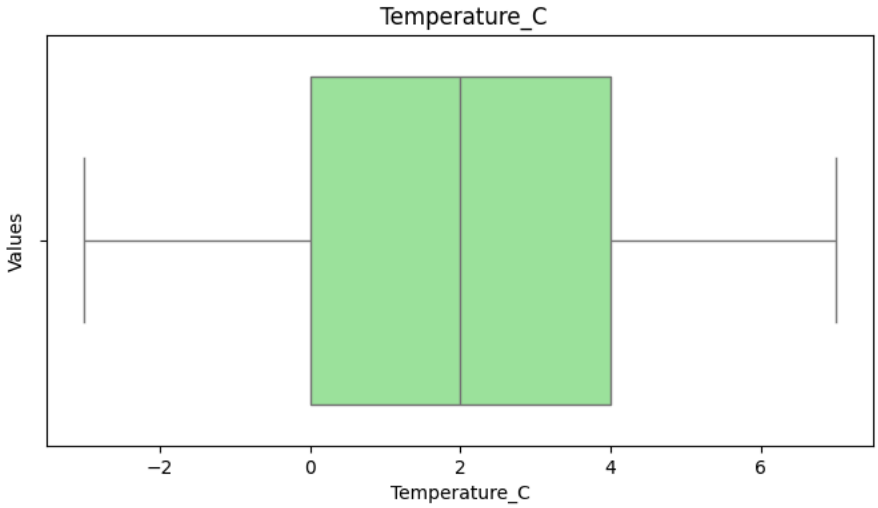

WindSpeed_kmh
Frequency
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
5
10
15
20
25
30
WindSpeed_kmh

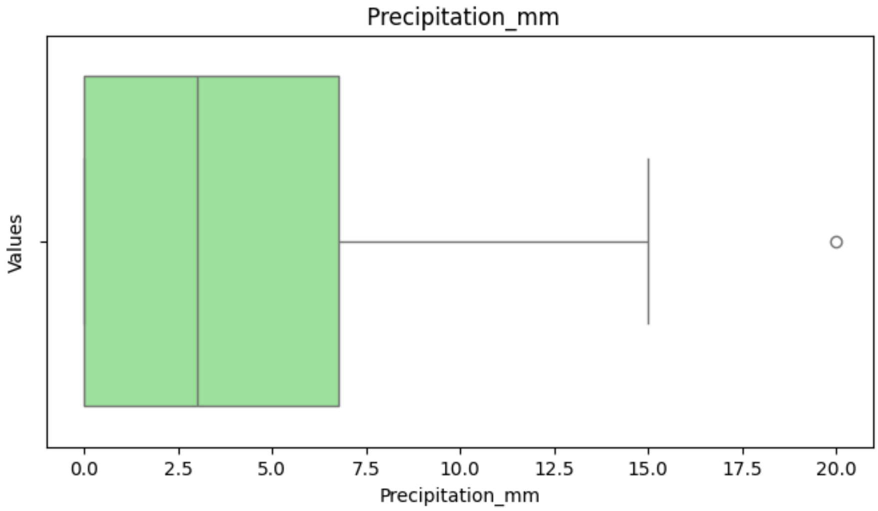

Delivery_Orders
Frequency
6
5
4
3
2
1
0
100
110
120
130
140
150
160
170
180
Delivery_Orders




## Box plots help visualize the distribution, quartiles, and outliers in each numerical attribute.

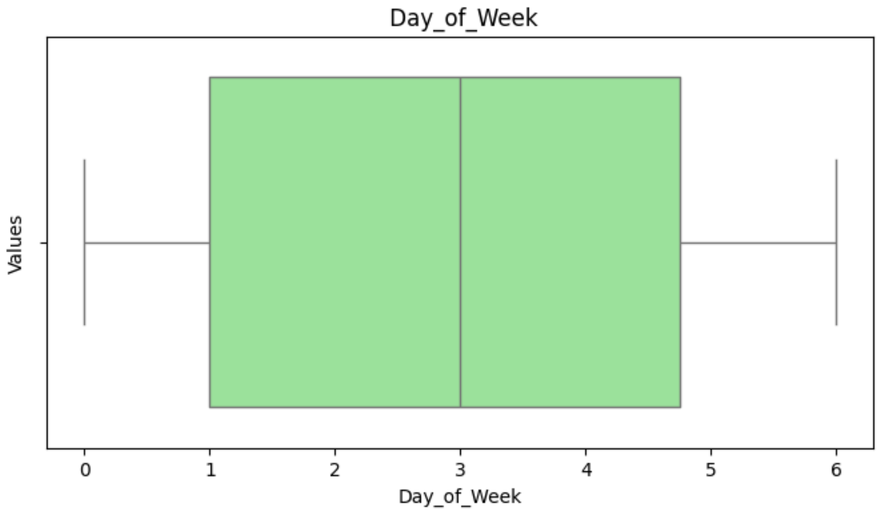
```
import seaborn as sns

# Box Plot for each numerical attribute
for col in numerical_attributes:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=data[col].dropna(), color='lightgreen')
    plt.title(f' {col}')
    plt.xlabel(col)  # X-axis: Attribute name
    plt.ylabel('Values')  # Y-axis: Attribute values
    plt.show()
```


Temperature_C
Values
−2
0
2
4
6
Temperature_C


Precipitation_mm
Values
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Precipitation_mm

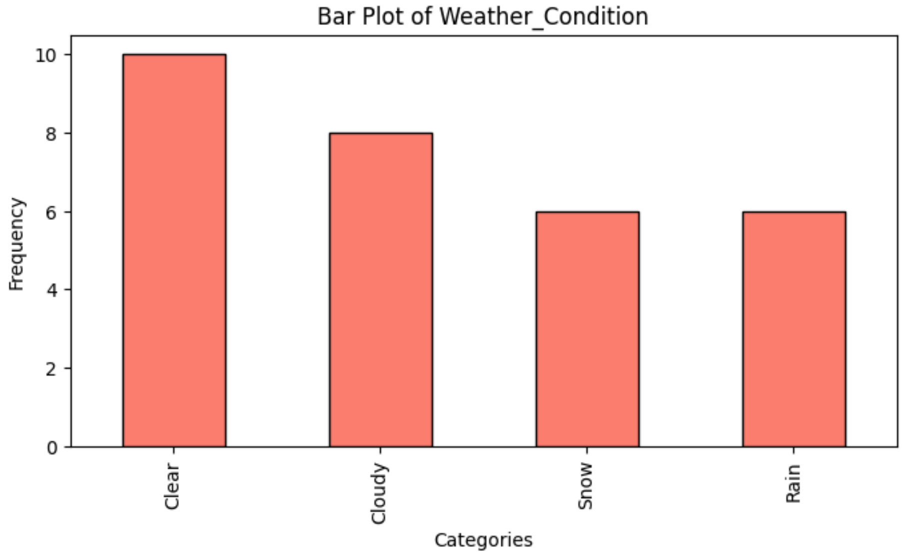

WindSpeed_kmh
Values
5
10
15
20
25
30
WindSpeed_kmh

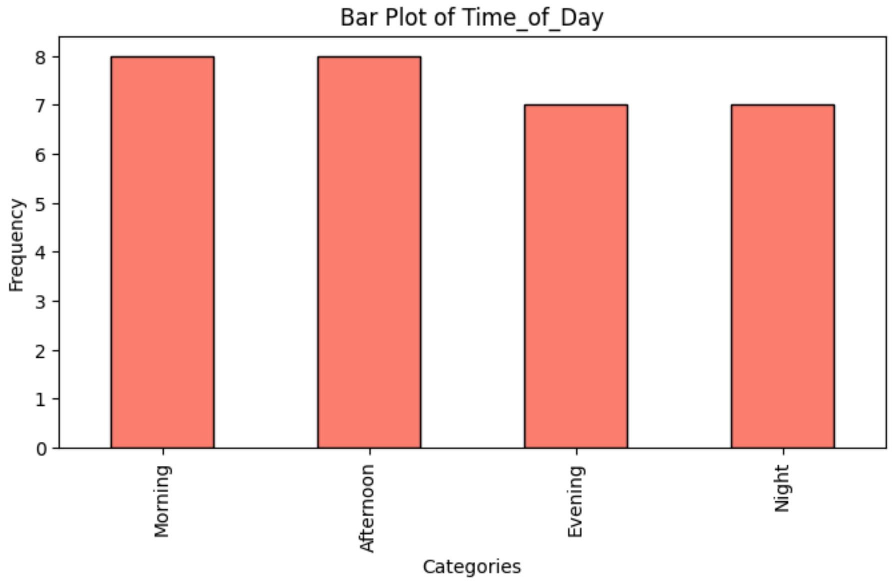

Delivery_Orders
Values
100
110
120
130
140
150
160
170
180
Delivery_Orders




## Bar plots show the frequency of each category in categorical attributes.

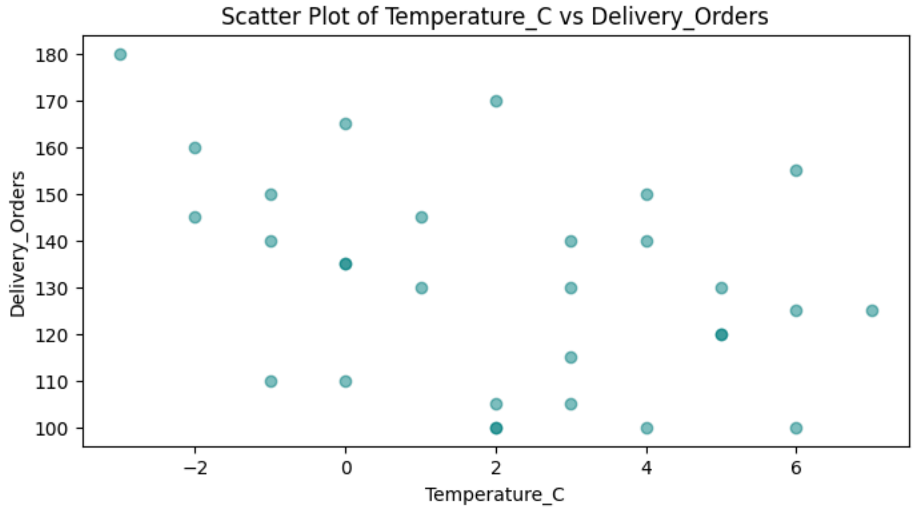
```
# Load the data
data = pd.read_csv('food_delivery_weather_data_with_time.csv')

# Identify categorical attributes and exclude 'Date'
categorical_attributes = data.select_dtypes(include=['object']).columns.tolist()
if 'Date' in categorical_attributes:
    categorical_attributes.remove('Date')  # Remove 'Date' from the list

# Bar Plot for each categorical attribute (excluding Date)
for col in categorical_attributes:
    plt.figure(figsize=(8, 4))
    data[col].value_counts().plot(kind='bar', color='salmon', edgecolor='black')
    plt.title(f'Bar Plot of {col}')
    plt.xlabel('Categories')  # X-axis: Categories in the attribute
    plt.ylabel('Frequency')  # Y-axis: Frequency of each category
    plt.show()
```

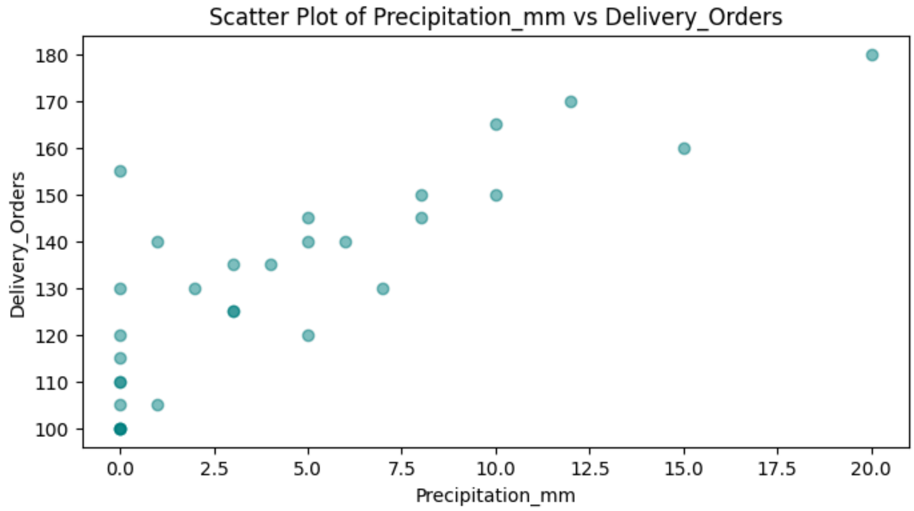


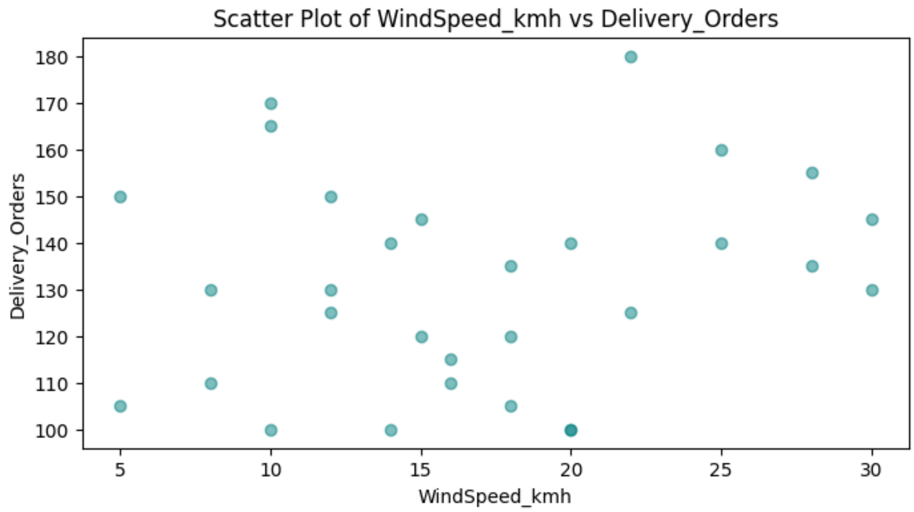



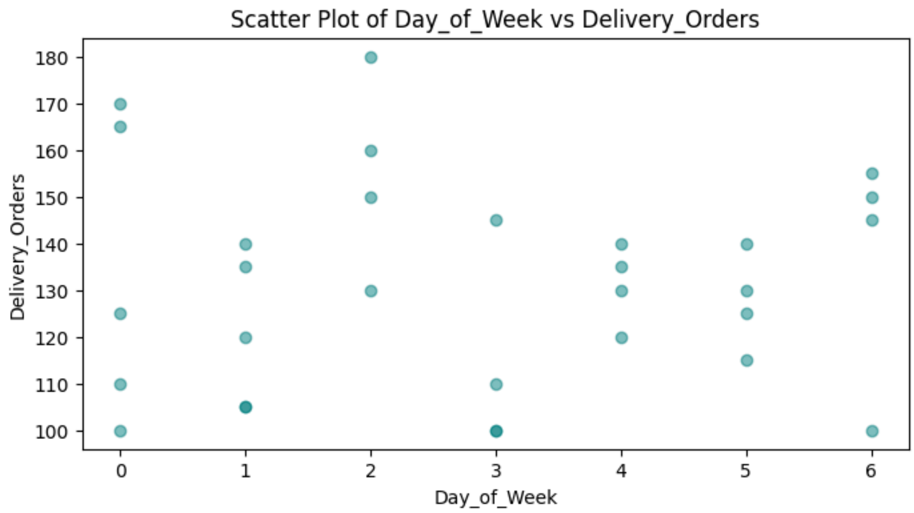

## Scatter plots visualize the relationship between each numerical attribute and the target variable (Delivery_Orders).

```
target = 'Delivery_Orders'
numerical_attributes.remove(target)  # Remove target from numerical attributes

# Scatter Plot for each numerical attribute vs. target variable
for col in numerical_attributes:
    plt.figure(figsize=(8, 4))
    plt.scatter(data[col], data[target], alpha=0.5, color='teal')
    plt.title(f'Scatter Plot of {col} vs {target}')
    plt.xlabel(col)  # X-axis: Attribute values
    plt.ylabel(target)  # Y-axis: Target variable values
    plt.show()
```


Scatter Plot of Temperature_C vs Delivery_Orders
Delivery_Orders
Temperature_C
180
170
160
150
140
130
120
110
100
−2
0
2
4
6


Scatter Plot of Precipitation_mm vs Delivery_Orders
Delivery_Orders
Precipitation_mm
180
170
160
150
140
130
120
110
100
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

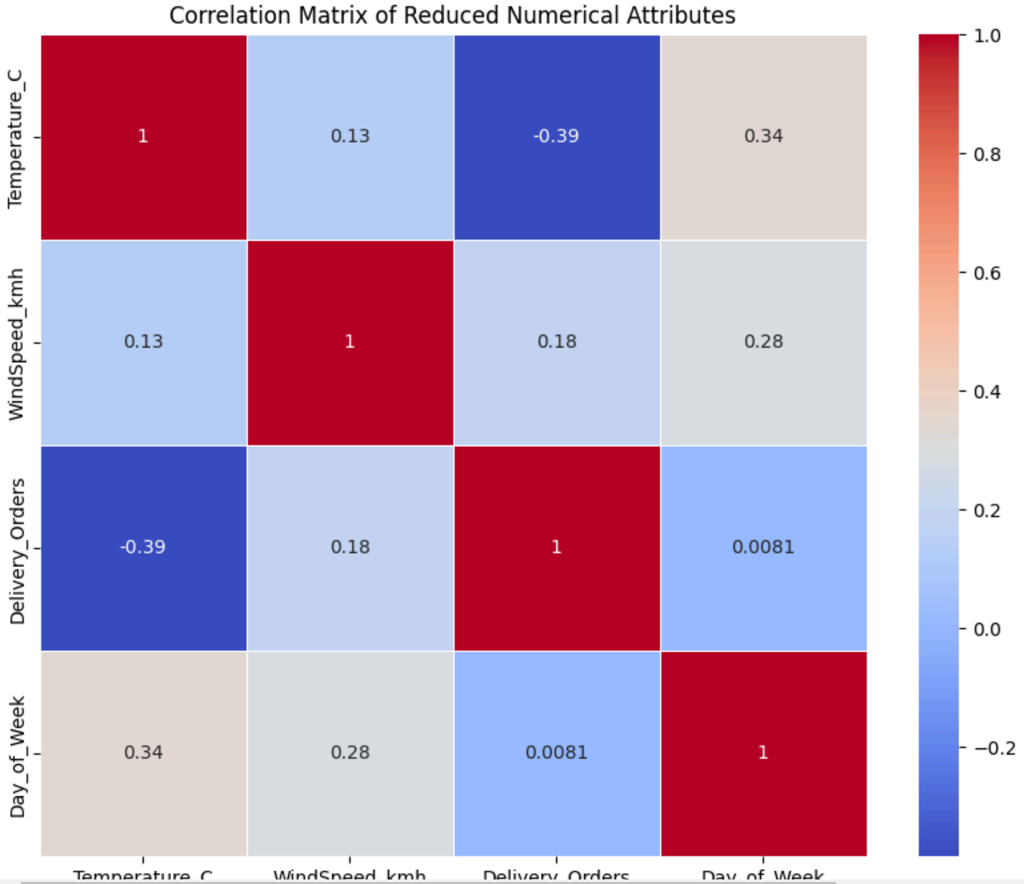

Scatter Plot of WindSpeed_kmh vs Delivery_Orders
Delivery_Orders
WindSpeed_kmh
180
170
160
150
140
130
120
110
100
5
10
15
20
25
30


Scatter Plot of Day_of_Week vs Delivery_Orders
Delivery_Orders
Day_of_Week
180
170
160
150
140
130
120
110
100
0
1
2
3
4
5
6

## Identify and Remove Redundant Attributes

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

# Load the data
data = pd.read_csv('food_delivery_weather_data_with_time.csv')

# Convert Date column to datetime format
data['Date'] = pd.to_datetime(data['Date'], errors='coerce')

# Select only numerical columns for correlation analysis
numerical_data = data.select_dtypes(include=[np.number])

# 1. Identify Highly Correlated Attributes
# Calculate the correlation matrix for numerical columns
correlation_matrix = numerical_data.corr().abs()  # Take absolute values for easier analysis

# Set a threshold for high correlation (e.g., 0.8)
correlation_threshold = 0.8
high_corr_pairs = []

# Find attribute pairs with correlation higher than the threshold
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if correlation_matrix.iloc[i, j] > correlation_threshold:
            col1 = correlation_matrix.columns[i]
            col2 = correlation_matrix.columns[j]
            high_corr_pairs.append((col1, col2))

print("Highly Correlated Attribute Pairs:", high_corr_pairs)

# 2. Remove Constant or Low-Variance Attributes
low_variance_cols = [col for col in numerical_data.columns if numerical_data[col].nunique() <= 1]
print("\nLow Variance Attributes:", low_variance_cols)

# 3. Remove Redundant Attributes
# Drop the identified redundant attributes (keeping one of each high-correlation pair and removing low-variance columns)
attributes_to_drop = set([pair[1] for pair in high_corr_pairs] + low_variance_cols)
data_reduced = data.drop(columns=attributes_to_drop)

print("\nAttributes Dropped:", attributes_to_drop)
print("\nRemaining Attributes:", data_reduced.columns.tolist())

# Show correlation heatmap of remaining numerical attributes for verification
plt.figure(figsize=(10, 8))
sns.heatmap(data_reduced.select_dtypes(include=[np.number]).corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix of Reduced Numerical Attributes')
plt.show()
```

```
Highly Correlated Attribute Pairs: [('Delivery_Orders', 'Precipitation_mm')]

Low Variance Attributes: []

Attributes Dropped: {'Precipitation_mm'}

Remaining Attributes: ['Date', 'Temperature_C', 'WindSpeed_kmh', 'Weather_Condition', 'Delivery_Orders', 'Time_of_Day', 'Day_of_Weel
```




## Again Data Cleaning

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

# Load the data
data = pd.read_csv('food_delivery_weather_data_with_time.csv')

# Step 1: Handling Missing Values
# Check for missing values
print("Missing Values:\n", data.isnull().sum())

# Method 1: Drop columns with too many missing values (e.g., > 50% missing)
data = data.dropna(thresh=len(data) * 0.5, axis=1)

# Method 2: Fill remaining missing values
# For numerical columns, we can use mean or median imputation
for col in data.select_dtypes(include=[np.number]).columns:
    data[col].fillna(data[col].median(), inplace=True)  # Using median for numerical columns

# For categorical columns, fill with the mode (most frequent value)
for col in data.select_dtypes(include=['object']).columns:
    data[col].fillna(data[col].mode()[0], inplace=True)

# Step 2: Removing Duplicates
# Check for duplicate rows
print("\nDuplicate Rows:", data.duplicated().sum())

# Drop duplicates
data = data.drop_duplicates()

# Step 3: Handling Outliers
# Using the Interquartile Range (IQR) method to identify and cap outliers in numerical columns
for col in data.select_dtypes(include=[np.number]).columns:
    Q1 = data[col].quantile(0.25)
```

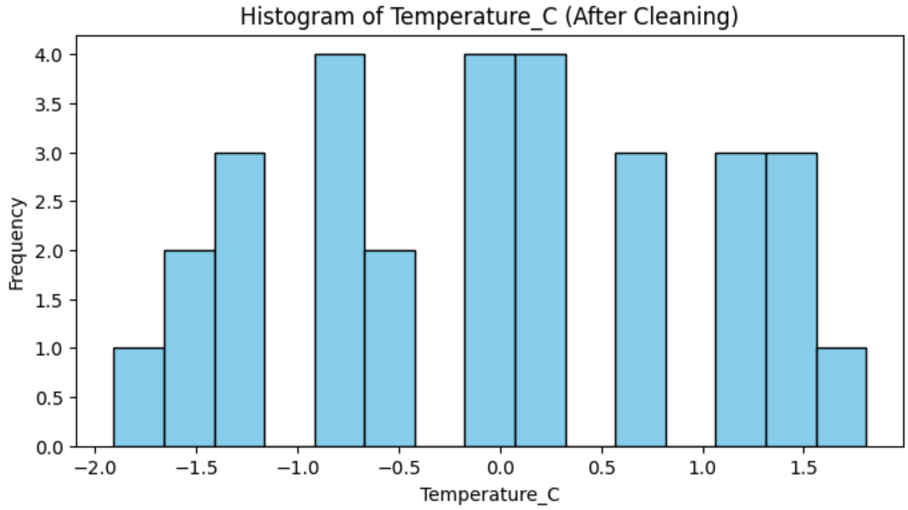
```
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        # Cap values beyond the lower and upper bounds
        data[col] = np.where(data[col] < lower_bound, lower_bound, data[col])
        data[col] = np.where(data[col] > upper_bound, upper_bound, data[col])

# Step 4: Standardizing Categorical Data
# Convert categorical columns to lowercase and strip whitespace for consistency
for col in data.select_dtypes(include=['object']).columns:
    data[col] = data[col].str.lower().str.strip()

# Step 5: Normalizing or Scaling Numerical Data
# Scale numerical columns using StandardScaler (for normalization)
scaler = StandardScaler()
numerical_cols = data.select_dtypes(include=[np.number]).columns
data[numerical_cols] = scaler.fit_transform(data[numerical_cols])

# Final Cleaned Data
print("\nCleaned Data Preview:")
print(data.head())
```

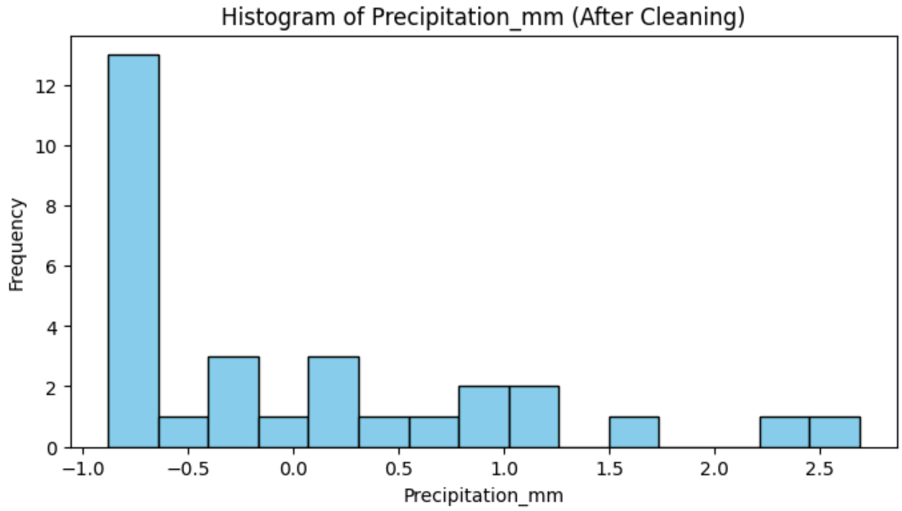
```
Missing Values:
 Date                 0
Temperature_C        0
Precipitation_mm     0
WindSpeed_kmh        0
Weather_Condition    0
Delivery_Orders      0
Time_of_Day          0
Day_of_Week          0
dtype: int64

Duplicate Rows: 0

Cleaned Data Preview:
         Date  Temperature_C  Precipitation_mm  WindSpeed_kmh  \
0  2024-01-01      -0.049523         -0.880161      -0.975176
1  2024-01-02       1.064735          0.177089      -0.265096
2  2024-01-03      -1.163780          1.234340      -1.685255
3  2024-01-04      -0.792361         -0.880161      -1.259208
4  2024-01-05       0.321897         -0.457261      -0.691144

  Weather_Condition  Delivery_Orders Time_of_Day  Day_of_Week
0             clear        -1.415976     morning    -1.392694
1            cloudy        -0.507328   afternoon    -0.901155
2              snow         0.855643     evening    -0.409616
3             clear        -0.961652       night     0.081923
4            cloudy        -0.053004     morning     0.573462
<ipython-input-54-2a8bbd30bdbf>:18: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  data[col].fillna(data[col].median(), inplace=True)  # Using median for numerical columns
<ipython-input-54-2a8bbd30bdbf>:22: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  data[col].fillna(data[col].mode()[0], inplace=True)
```

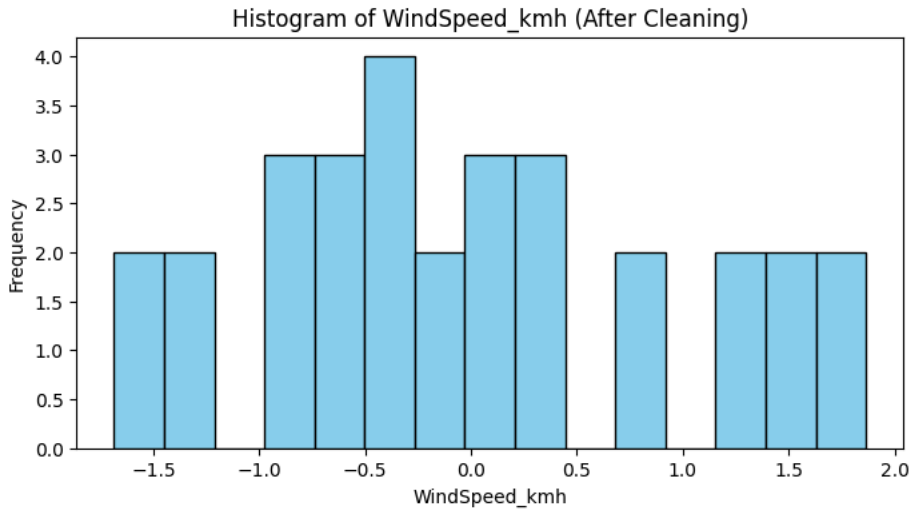
## Re-Plotting and Re-Analyzing the Data

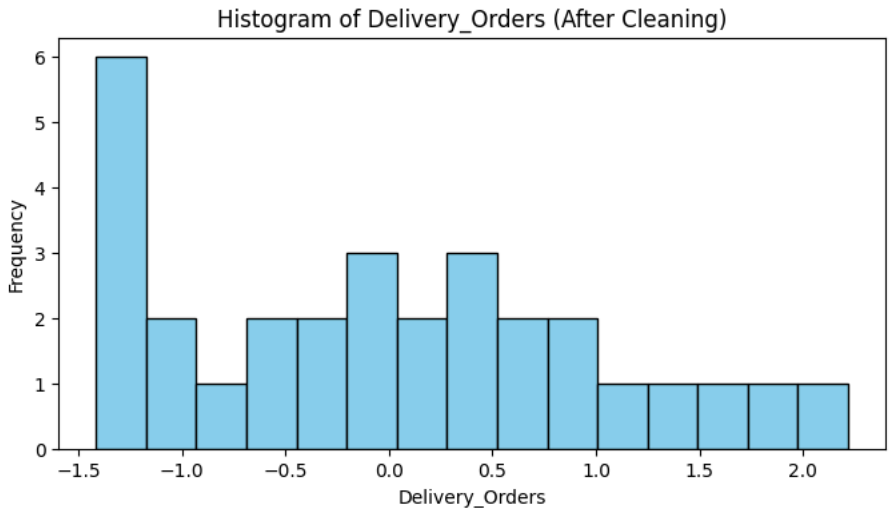
```
# Identify numerical and categorical attributes
numerical_attributes = data.select_dtypes(include=[np.number]).columns.tolist()
categorical_attributes = data.select_dtypes(include=['object']).columns.tolist()

# Step 1: Histograms for Numerical Attributes
for col in numerical_attributes:
    plt.figure(figsize=(8, 4))
    plt.hist(data[col].dropna(), bins=15, color='skyblue', edgecolor='black')
    plt.title(f'Histogram of {col} (After Cleaning)')
    plt.xlabel(col)  # X-axis: Attribute values
    plt.ylabel('Frequency')  # Y-axis: Frequency of occurrences
    plt.show()

# Step 2: Box Plots for Numerical Attributes
```

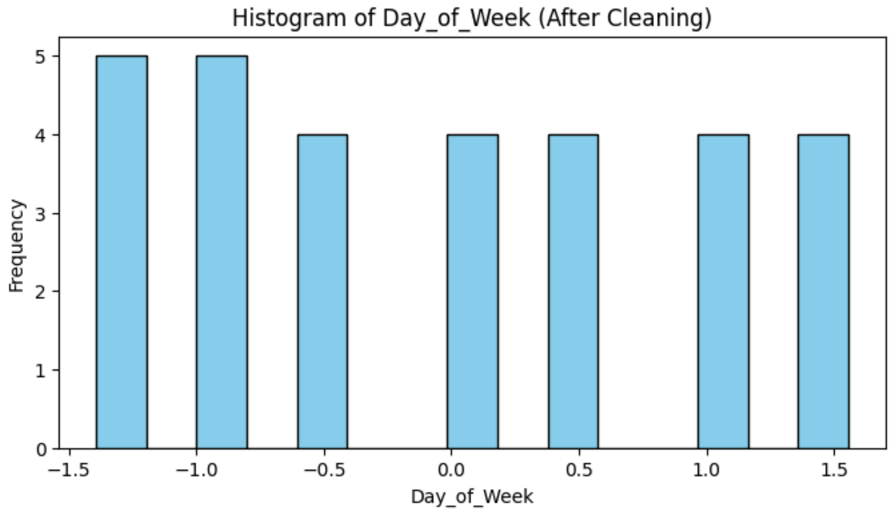
```
for col in numerical_attributes:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=data[col].dropna(), color='lightgreen')
    plt.title(f'Box Plot of {col} (After Cleaning)')
    plt.xlabel(col)  # X-axis: Attribute name
    plt.ylabel('Values')  # Y-axis: Attribute values
    plt.show()

# Step 3: Bar Plots for Categorical Attributes (excluding Date)
if 'Date' in categorical_attributes:
    categorical_attributes.remove('Date')

for col in categorical_attributes:
    plt.figure(figsize=(8, 4))
    data[col].value_counts().plot(kind='bar', color='salmon', edgecolor='black')
    plt.title(f'Bar Plot of {col} (After Cleaning)')
    plt.xlabel('Categories')  # X-axis: Categories in the attribute
    plt.ylabel('Frequency')  # Y-axis: Frequency of each category
    plt.show()

# Step 4: Scatter Plots for Numerical Attributes vs. Target Variable
target = 'Delivery_Orders'  # Define target variable if applicable
if target in numerical_attributes:
    numerical_attributes.remove(target)  # Remove target for comparison with other attributes
    for col in numerical_attributes:
        plt.figure(figsize=(8, 4))
        plt.scatter(data[col], data[target], alpha=0.5, color='teal')
        plt.title(f'Scatter Plot of {col} vs {target} (After Cleaning)')
        plt.xlabel(col)  # X-axis: Attribute values
        plt.ylabel(target)  # Y-axis: Target variable values
        plt.show()

# Step 5: Pair Plot for Numerical Attributes
# Ensures pair plot only includes available columns after cleaning
available_numerical_attributes = [col for col in numerical_attributes + [target] if col in data.columns]
sns.pairplot(data[available_numerical_attributes].dropna())
plt.suptitle('Pair Plot of Numerical Attributes (After Cleaning)', y=1.02)
plt.show()
```

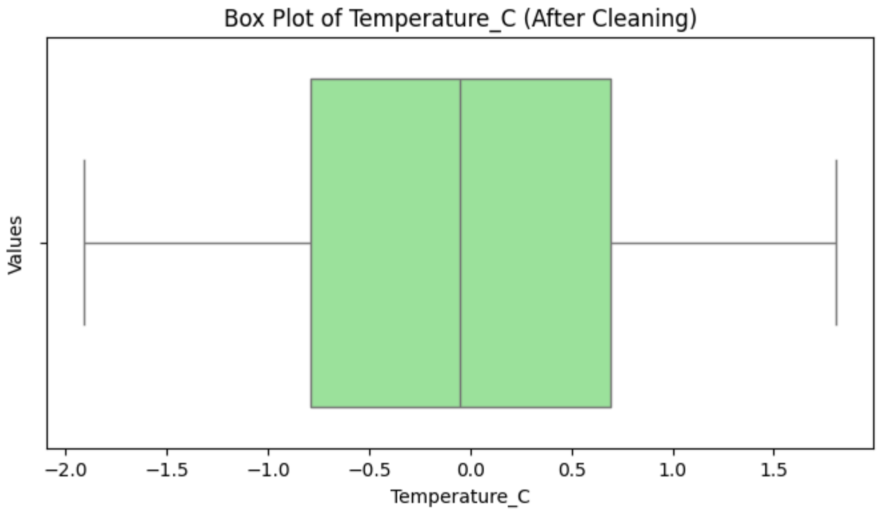

Histogram of Temperature_C (After Cleaning)
Frequency
Temperature_C

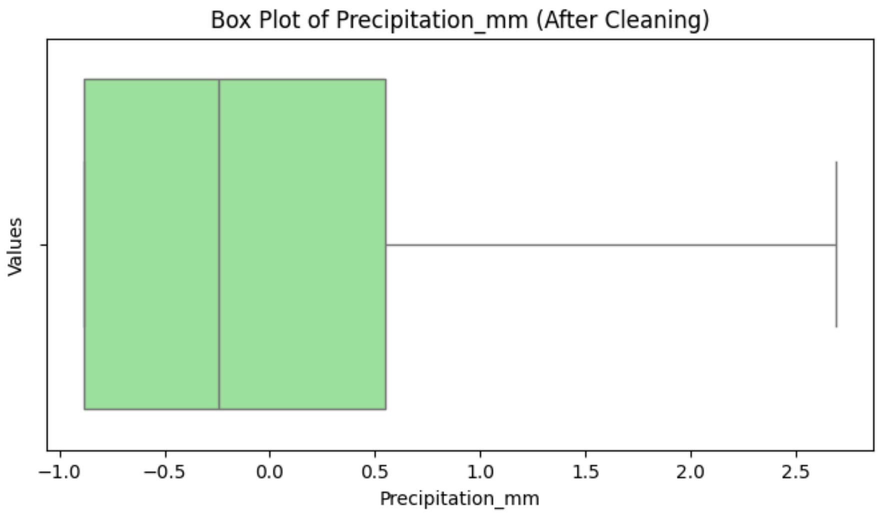

Histogram of Precipitation_mm (After Cleaning)
Frequency
Precipitation_mm


Histogram of WindSpeed_kmh (After Cleaning)
Frequency
WindSpeed_kmh


Histogram of Delivery_Orders (After Cleaning)
Frequency
Delivery_Orders

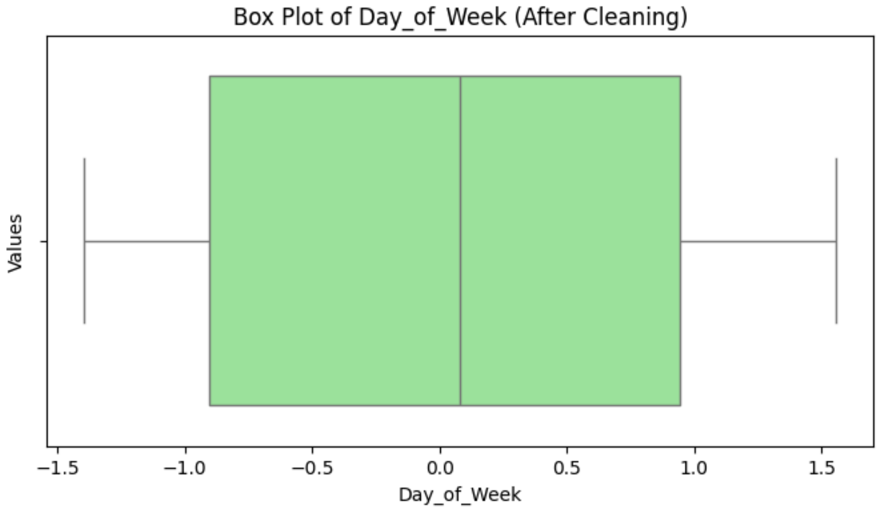

Histogram of Day_of_Week (After Cleaning)
Frequency
0
1
2
3
4
5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Day_of_Week

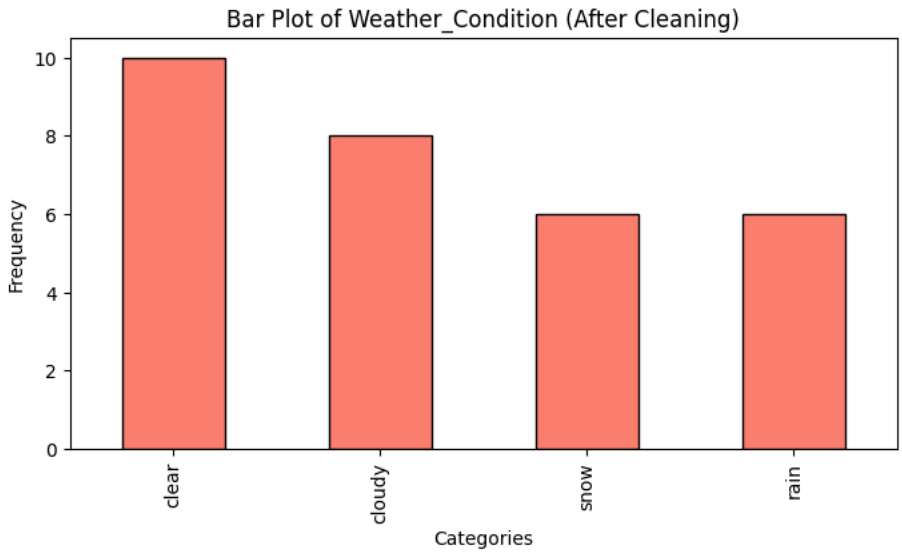

Box Plot of Temperature_C (After Cleaning)
Values
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Temperature_C

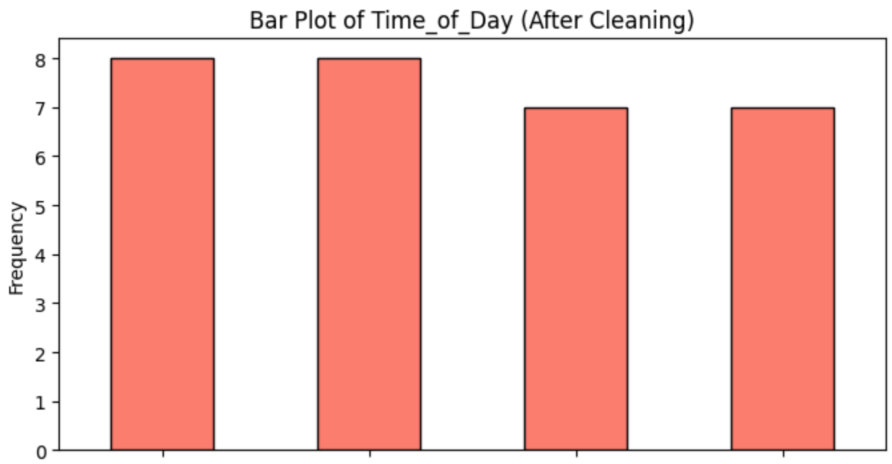

Box Plot of Precipitation_mm (After Cleaning)
Values
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Precipitation_mm

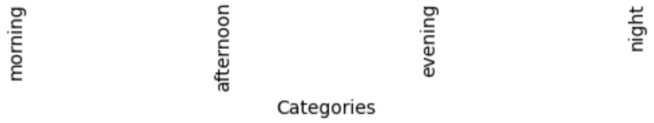

Box Plot of WindSpeed_kmh (After Cleaning)
Values
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
WindSpeed_kmh

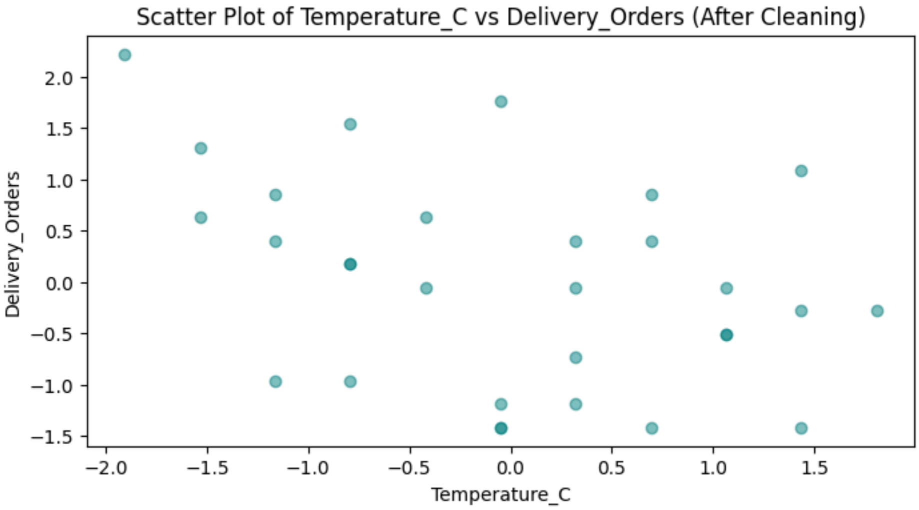

Box Plot of Delivery_Orders (After Cleaning)
Values
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0
Delivery_Orders

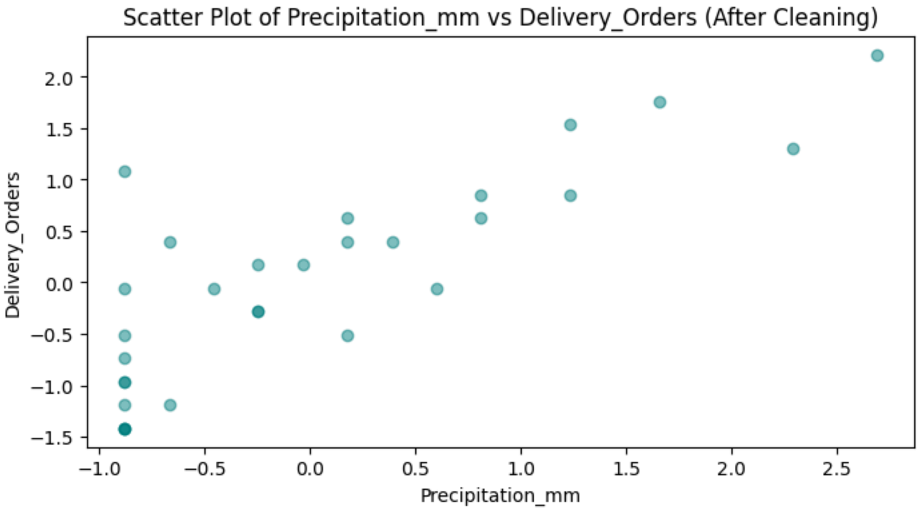

Box Plot of Day_of_Week (After Cleaning)
Values
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5
Day_of_Week

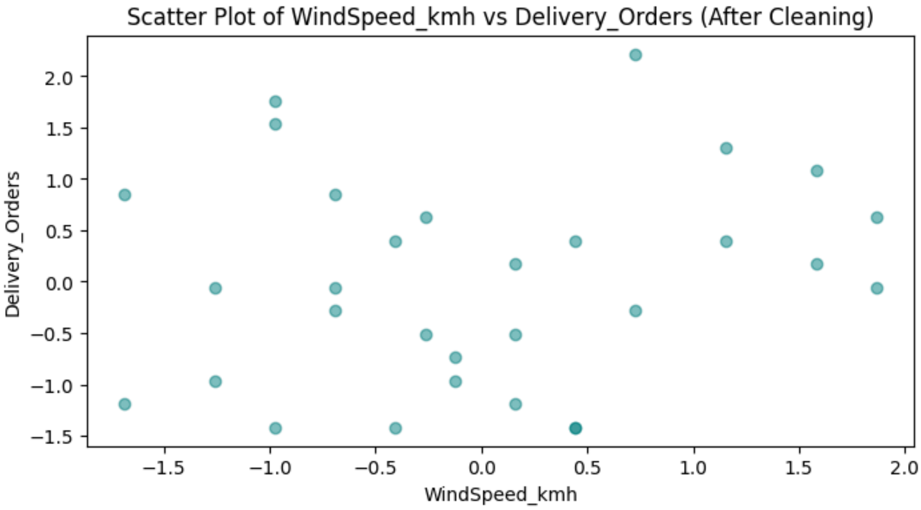

Bar Plot of Weather_Condition (After Cleaning)
Frequency
0 2 4 6 8 10
clear
cloudy
snow
rain
Categories

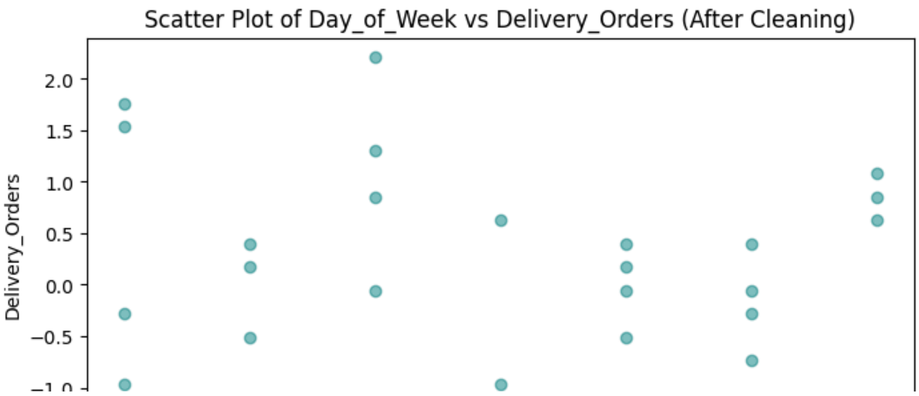

Bar Plot of Time_of_Day (After Cleaning)
Frequency
0 1 2 3 4 5 6 7 8

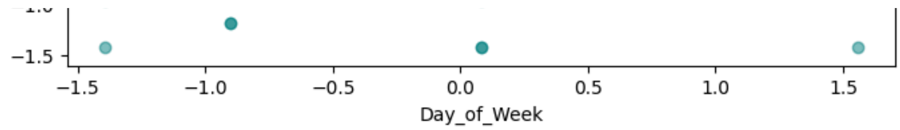

morning
afternoon
evening
night
Categories

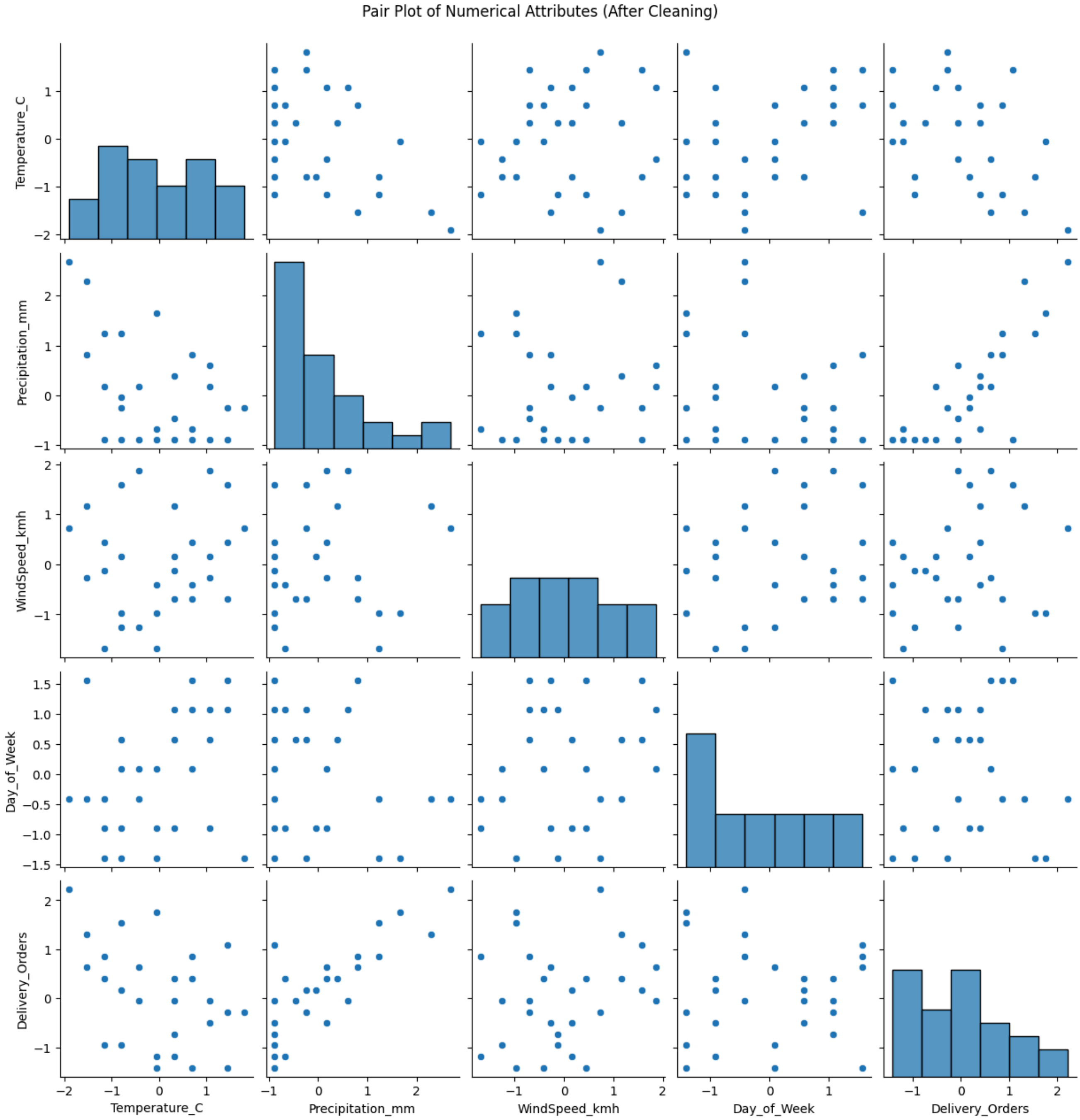

Scatter Plot of Temperature_C vs Delivery_Orders (After Cleaning)
Delivery_Orders
Temperature_C


Scatter Plot of Precipitation_mm vs Delivery_Orders (After Cleaning)
Delivery_Orders
Precipitation_mm


Scatter Plot of WindSpeed_kmh vs Delivery_Orders (After Cleaning)
Delivery_Orders
WindSpeed_kmh


Scatter Plot of Day_of_Week vs Delivery_Orders (After Cleaning)
Delivery_Orders


−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Day_of_Week


Pair Plot of Numerical Attributes (After Cleaning)
Temperature_C
Precipitation_mm
WindSpeed_kmh
Day_of_Week
Delivery_Orders

## Prepare Train and Test Datasets

To prepare the train and test datasets, I'll split the cleaned dataset into two parts:

- Training Set: Used to train the machine learning model.
- Testing Set: Used to evaluate the model's performance on unseen data.

**target:** in this case, Delivery_Orders is our Target.

- **train_test_split:** Splits the data into an 80% training set and 20% testing set.
- **random_state=42:** Ensures reproducibility of the split, providing the same split each time the code is run.

```
# show the target variable and features
target = 'Delivery_Orders'  # our target variable
features = data.drop(columns=[target]).columns.tolist()  # All columns except the target

# Separate features (X) and target (y)
X = data[features]
y = data[target]

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Display the shapes of the resulting datasets
print("Training Set (X_train):", X_train.shape)
print("Testing Set (X_test):", X_test.shape)
print("Training Set (y_train):", y_train.shape)
print("Testing Set (y_test):", y_test.shape)
```

```
Training Set (X_train): (24, 7)
Testing Set (X_test): (6, 7)
Training Set (y_train): (24,)
Testing Set (y_test): (6,)
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Reloading the merged data
merged_data_by_date = pd.read_csv('/content/merged_deliveries_weather_data.csv')

# Selecting relevant features for predicting 'Delivery Count'
features = ['Temperature_High_C', 'Temperature_Low_C', 'Precipitation_mm', 'Humidity_%', 'Wind_Speed_kmh', 'Delivery Fee (CAD)', 'Total
target = 'Delivery Count'

# Dropping rows with missing values in selected features and target
cleaned_data = merged_data_by_date.dropna(subset=features + [target])

# Splitting the data into features (X) and target (y)
X = cleaned_data[features]
y = cleaned_data[target]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scaling numeric features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Training a Random Forest Regressor model
model = RandomForestRegressor(random_state=42)
model.fit(X_train_scaled, y_train)

# Making predictions on the test data
y_pred = model.predict(X_test_scaled)

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

mse, r2
```