```
import numpy as np
import pandas as pd


import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

pd.set_option('display.max_columns', None)
```

```
# ************* Misc. *************
import random

from prettytable import PrettyTable

# ************* Plotting *************
import seaborn as sns
import missingno as msno
import matplotlib.pyplot as plt

# ************* Data Manipulation *************
from sklearn.preprocessing import LabelEncoder
```

```
df = pd.read_csv("/content/weather_dataet.csv")

df.describe()
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Atmospheric Pressure
count	13200.000000	13200.000000	13200.000000	13200.000000	13200.000000
mean	19.127576	68.710833	9.832197	53.644394	1005.827896
std	17.386327	20.194248	6.908704	31.946541	37.199589
min	-25.000000	20.000000	0.000000	0.000000	800.120000
25%	4.000000	57.000000	5.000000	19.000000	994.800000
50%	21.000000	70.000000	9.000000	58.000000	1007.650000
75%	31.000000	84.000000	13.500000	82.000000	1016.772500
max	109.000000	109.000000	48.500000	109.000000	1199.210000

```
df
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	Ind
0	14	73	9.5	82	partly cloudy	1010.82	
1	39	96	8.5	71	partly cloudy	1011.43	
2	30	64	7.0	16	clear	1018.72	
3	38	83	1.5	82	clear	1026.25	
4	27	74	17.0	66	overcast	990.67	
...	...	...	...	...	...	...	
13195	10	74	14.5	71	overcast	1003.15	
13196	-1	76	3.5	23	cloudy	1067.23	
13197	30	77	5.5	28	overcast	1012.69	
13198	3	76	10.0	94	overcast	984.27	
13199	-5	38	0.0	92	overcast	1015.37	

13200 rows × 11 columns

Next steps: Generate code with df | New interactive sheet

```
df.head()
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	UV Index	S
0	14	73	9.5	82	partly cloudy	1010.82	2	
1	39	96	8.5	71	partly cloudy	1011.43	7	
2	30	64	7.0	16	clear	1018.72	5	
3	38	83	1.5	82	clear	1026.25	7	
4	27	74	17.0	66	overcast	990.67	1	

Next steps: Generate code with df | New interactive sheet

```
df.tail()
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	I Ind
13195	10	74	14.5	71	overcast	1003.15	
13196	-1	76	3.5	23	cloudy	1067.23	
13197	30	77	5.5	28	overcast	1012.69	
13198	3	76	10.0	94	overcast	984.27	
13199	-5	38	0.0	92	overcast	1015.37	

```
df.head(200)
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	UV Index
0	14	73	9.5	82	partly cloudy	1010.82	2
1	39	96	8.5	71	partly cloudy	1011.43	7
2	30	64	7.0	16	clear	1018.72	5
3	38	83	1.5	82	clear	1026.25	7
4	27	74	17.0	66	overcast	990.67	1
...	...	...	...	...	...	...	...
195	26	68	4.0	39	partly cloudy	1016.39	4
196	10	99	16.0	58	overcast	995.85	2
197	20	42	4.0	13	partly cloudy	1028.30	5
198	-2	32	1.5	17	overcast	930.32	1
199	-1	94	13.0	74	overcast	981.13	0

200 rows × 11 columns

Next steps: Generate code with df | New interactive sheet

```
df.tail(200)
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	UV Index
13000	-6	65	6.0	85	overcast	980.58	
13001	-10	86	9.5	98	overcast	985.98	
13002	-5	61	14.0	63	overcast	999.86	
13003	10	51	4.5	31	partly cloudy	1008.56	
13004	41	48	4.5	10	clear	1013.20	
...	...	...	...	...	...	...	
13195	10	74	14.5	71	overcast	1003.15	
13196	-1	76	3.5	23	cloudy	1067.23	
13197	30	77	5.5	28	overcast	1012.69	
13198	3	76	10.0	94	overcast	984.27	
13199	-5	38	0.0	92	overcast	1015.37	

200 rows × 11 columns

```
df.count()
```

	0
Temperature	13200
Humidity	13200
Wind Speed	13200
Precipitation (%)	13200
Cloud Cover	13200
Atmospheric Pressure	13200
UV Index	13200
Season	13200
Visibility (km)	13200
Location	13200
Weather Type	13200

dtype: int64

```
df.shape
```

```
(13200, 11)
```

```
df.describe()
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Atmospheric Pressure
count	13200.000000	13200.000000	13200.000000	13200.000000	13200.000000
mean	19.127576	68.710833	9.832197	53.644394	1005.827896
std	17.386327	20.194248	6.908704	31.946541	37.199589
min	-25.000000	20.000000	0.000000	0.000000	800.120000
25%	4.000000	57.000000	5.000000	19.000000	994.800000
50%	21.000000	70.000000	9.000000	58.000000	1007.650000
75%	31.000000	84.000000	13.500000	82.000000	1016.772500
max	109.000000	109.000000	48.500000	109.000000	1199.210000

```
df.count()
```

	0
Temperature	13200
Humidity	13200
Wind Speed	13200
Precipitation (%)	13200
Cloud Cover	13200
Atmospheric Pressure	13200
UV Index	13200
Season	13200
Visibility (km)	13200
Location	13200
Weather Type	13200

dtype: int64

```
df.min()
```

	0
Temperature	-25
Humidity	20
Wind Speed	0.0
Precipitation (%)	0
Cloud Cover	clear
Atmospheric Pressure	800.12
UV Index	0
Season	Autumn
Visibility (km)	0.0
Location	coastal
Weather Type	Cloudy

dtype: object

```
df.max()
```

	0
Temperature	109
Humidity	109
Wind Speed	48.5
Precipitation (%)	109
Cloud Cover	partly cloudy
Atmospheric Pressure	1199.21
UV Index	14
Season	Winter
Visibility (km)	20.0
Location	mountain
Weather Type	Sunny

dtype: object

```
df.isnull()
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	UV Index
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
13195	False	False	False	False	False	False	False
13196	False	False	False	False	False	False	False
13197	False	False	False	False	False	False	False
13198	False	False	False	False	False	False	False
13199	False	False	False	False	False	False	False

13200 rows × 11 columns

```
df.notnull()
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	UV Index
0	True	True	True	True	True	True	True
1	True	True	True	True	True	True	True
2	True	True	True	True	True	True	True
3	True	True	True	True	True	True	True
4	True	True	True	True	True	True	True
...	...	...	...	...	...	...	...
13195	True	True	True	True	True	True	True
13196	True	True	True	True	True	True	True
13197	True	True	True	True	True	True	True
13198	True	True	True	True	True	True	True
13199	True	True	True	True	True	True	True

13200 rows × 11 columns

```
df.dropna()
```

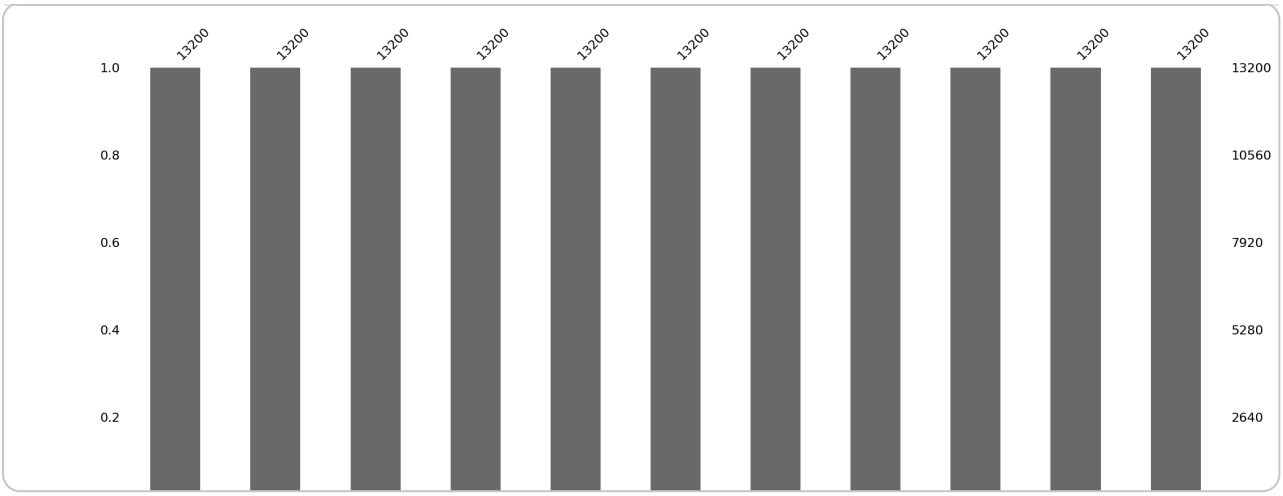
	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	I Ind
0	14	73	9.5	82	partly cloudy	1010.82	
1	39	96	8.5	71	partly cloudy	1011.43	
2	30	64	7.0	16	clear	1018.72	
3	38	83	1.5	82	clear	1026.25	
4	27	74	17.0	66	overcast	990.67	
...	...	...	...	...	...	...	
13195	10	74	14.5	71	overcast	1003.15	
13196	-1	76	3.5	23	cloudy	1067.23	
13197	30	77	5.5	28	overcast	1012.69	
13198	3	76	10.0	94	overcast	984.27	
13199	-5	38	0.0	92	overcast	1015.37	

13200 rows × 11 columns

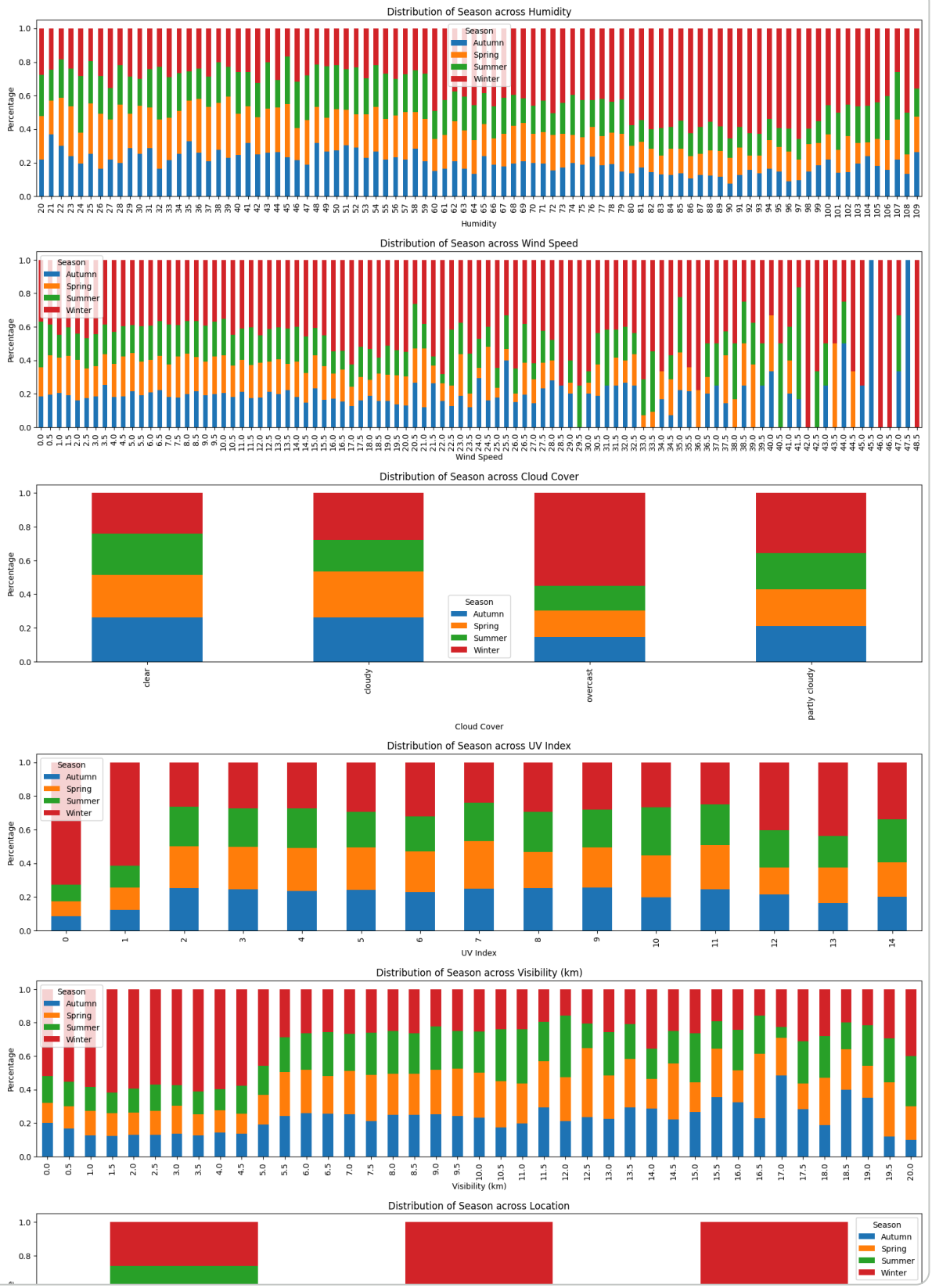
```
table = PrettyTable()
table.field_names = ['Feature', 'Data Type']

for column in df.columns:
    column_dtype = str(df[column].dtype)
    table.add_row([column, column_dtype])

print(table)
```

```
+----------------------+-----------+
|       Feature        | Data Type |
+----------------------+-----------+
|     Temperature      |   int64   |
|       Humidity       |   int64   |
|      Wind Speed      |  float64  |
|  Precipitation (%)   |   int64   |
|     Cloud Cover      |   object  |
| Atmospheric Pressure |  float64  |
|       UV Index       |   int64   |
|        Season        |   object  |
|   Visibility (km)    |  float64  |
|       Location       |   object  |
|     Weather Type     |   object  |
+----------------------+-----------+
```

```
msno.bar(df)

plt.show()
```




```
def distribution_of_target(target, dataframe):
    cat_cols = [feature
                for feature in dataframe.columns
                if (dataframe[feature].dtype != 'O' and dataframe[feature].n
                or (dataframe[feature].dtype == 'O' and feature not in [targ
                ]

    for column in cat_cols:
        contingency_table = pd.crosstab(dataframe[column], dataframe[target]
        contingency_table.plot(kind="bar", stacked=True, figsize=(20, 4))

        plt.title(f"Distribution of {target} across {column}")
        plt.xlabel(column)
        plt.ylabel("Percentage")
        plt.legend(title=target)

        plt.show()

distribution_of_target("Season", df)
```

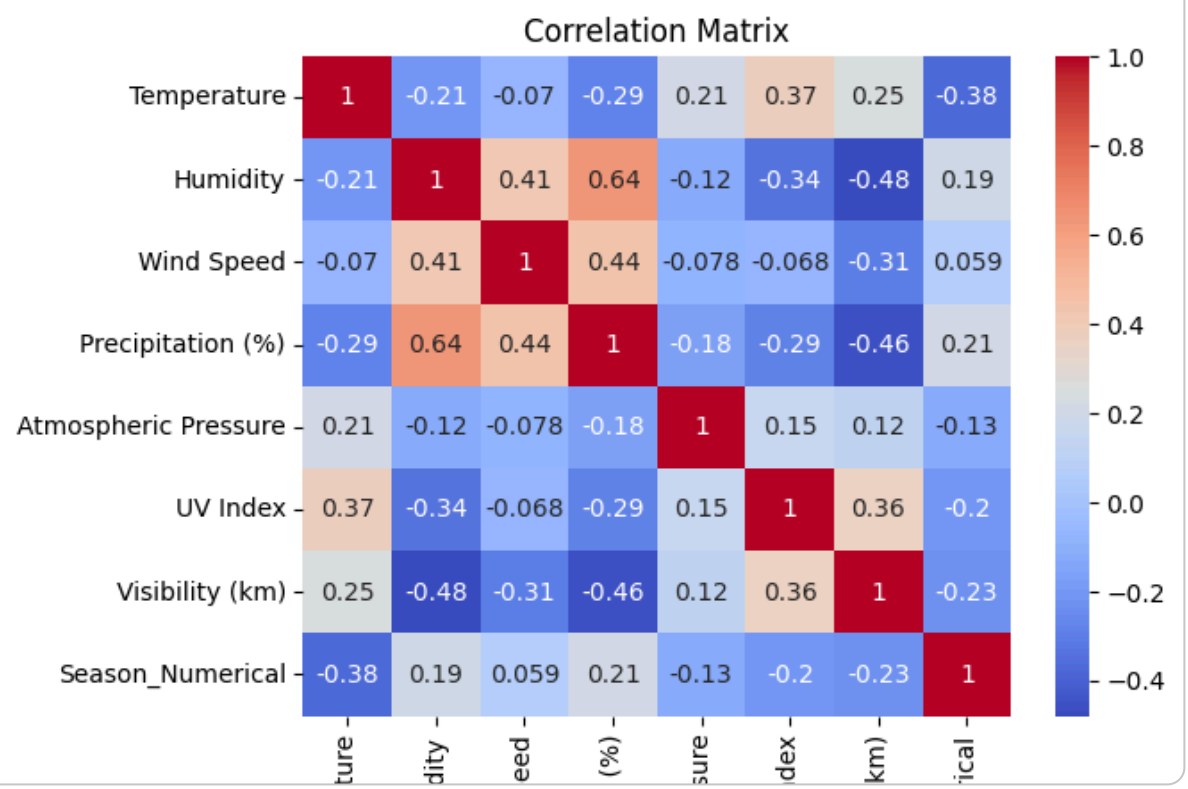



```
Start coding or generate with AI.
```

Correlation MAtrix

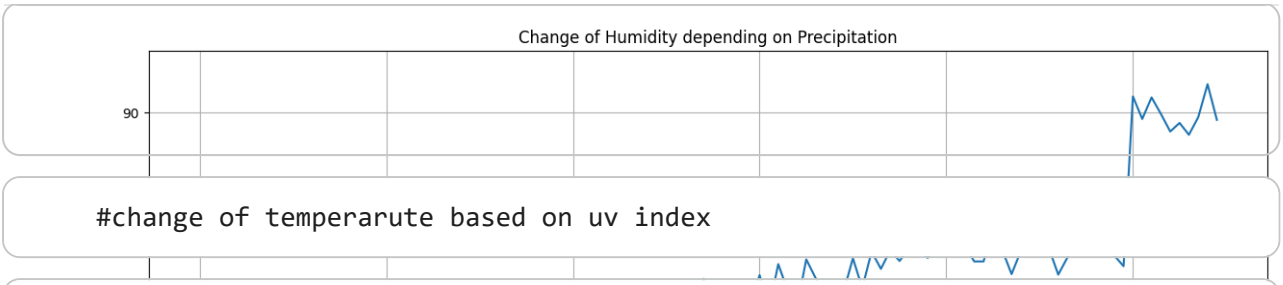
```
label_encoder = LabelEncoder()
df["Season_Numerical"] = label_encoder.fit_transform(df["Season"])
numerical_df = df.select_dtypes(include=["int", "float"])
corr_matrix = numerical_df.corr()
```

```
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')
plt.show()
```




```
#relationships like change of humiditydepending on various factors like prci
```

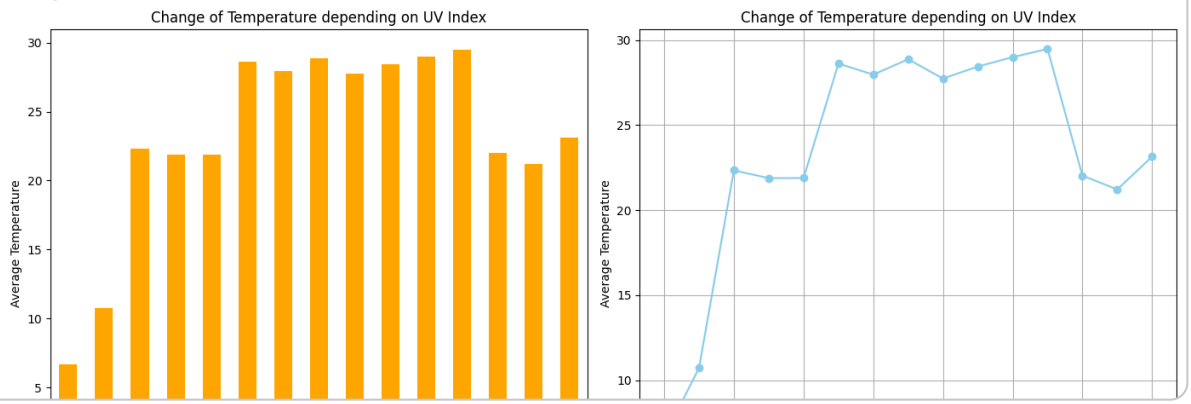
```
plt.figure(figsize=(15, 10))

precipitation_on_humidity = df.groupby("Precipitation ( )

precipitation_on_humidity.plot(kind="line")

plt.title('Change of Humidity depending on Precipitatic
plt.xlabel('Precipitation')
plt.ylabel('Average Humidity')
plt.grid(True)

plt.show()
```




```
#change of temperarute based on uv index
```

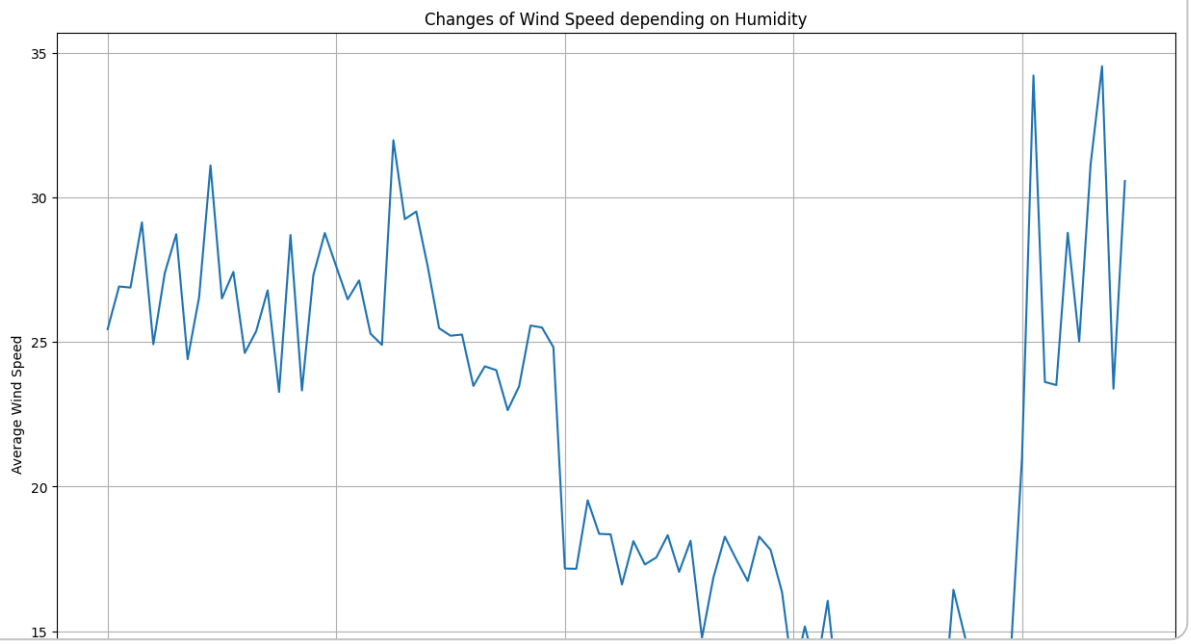
```
plt.figure(figsize=(15, 10))

temperature_on_uv = df.groupby("UV Index")["Temperature"].mean()

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))

# Bar Chart
temperature_on_uv.plot(kind='bar', ax=axes[0], color='orange')
axes[0].set_title('Change of Temperature depending on UV Index')
axes[0].set_xlabel('UV Index')
axes[0].set_ylabel('Average Temperature')

# Line Chart
temperature_on_uv.plot(kind='line', ax=axes[1], color='skyblue', marker='o')
axes[1].set_title('Change of Temperature depending on UV Index')
axes[1].set_xlabel('UV Index')
axes[1].set_ylabel('Average Temperature')
axes[1].grid(True)

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

```
<Figure size 1500x1000 with 0 Axes>
```

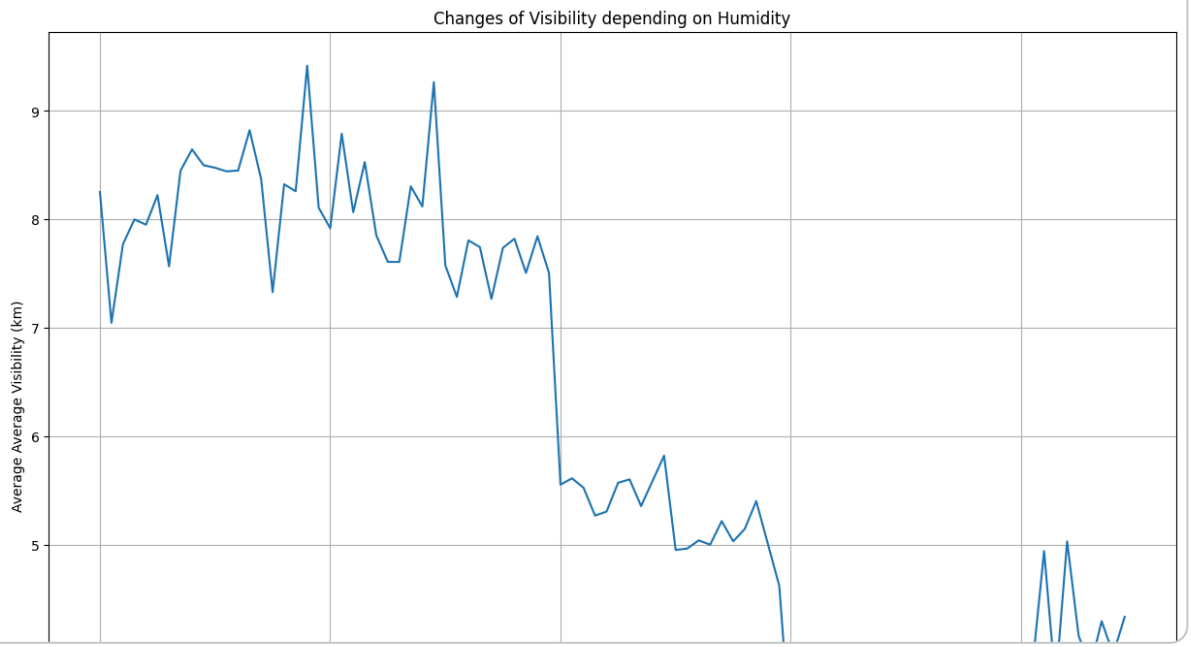



```
#change od wind speed depending on humidity
plt.figure(figsize=(15, 10))

wind_on_humidity = df.groupby("Humidity")["Temperature"].mean()

wind_on_humidity.plot(kind="line")

plt.title('Changes of Wind Speed depending on Humidity')
plt.xlabel('Humidity')
plt.ylabel('Average Wind Speed')
plt.grid(True)
```

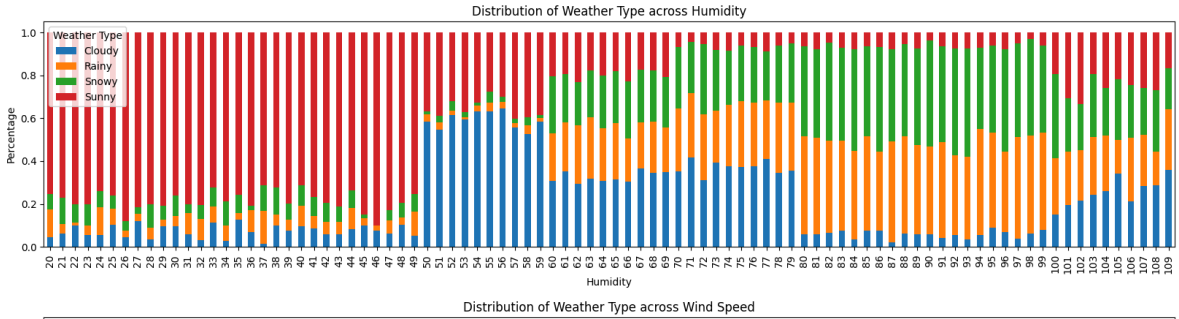
```
plt.show()
```

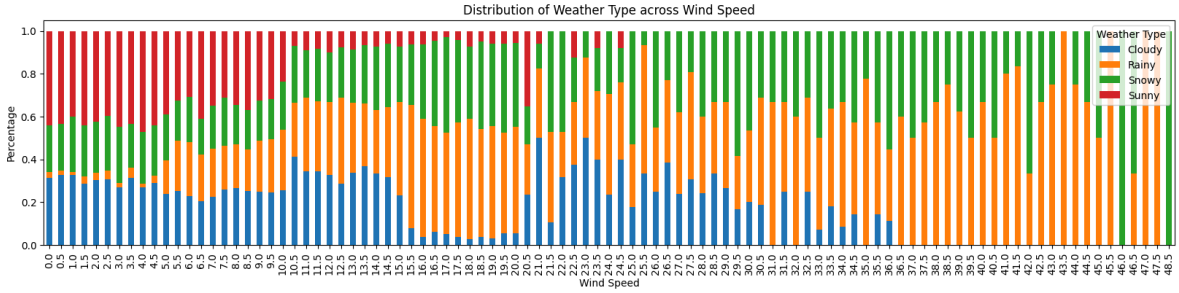


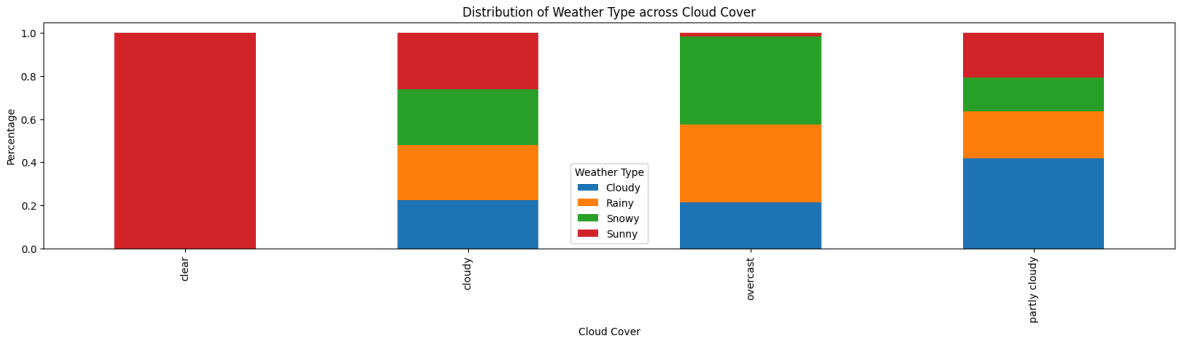

```
#Change of visibility based on humidity
plt.figure(figsize=(15, 10))

visibility_on_humidity = df.groupby("Humidity")["Visibility (km)"].mean()

visibility_on_humidity.plot(kind="line")

plt.title('Changes of Visibility depending on Humidity')
plt.xlabel('Humidity')
plt.ylabel('Average Average Visibility (km)')
plt.grid(True)

plt.show()
```

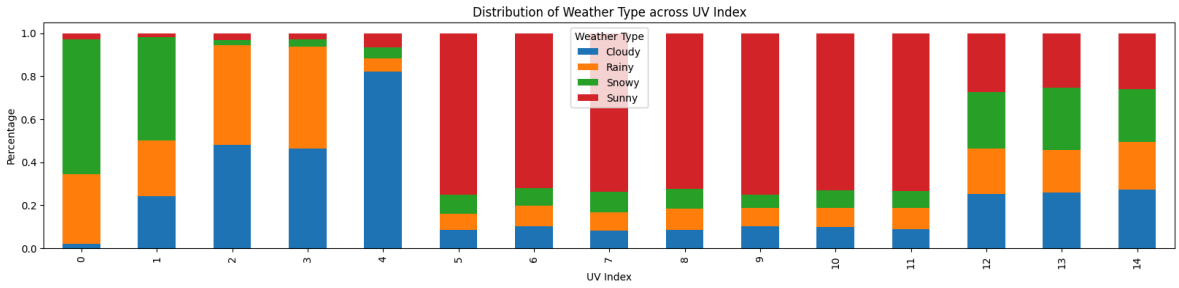


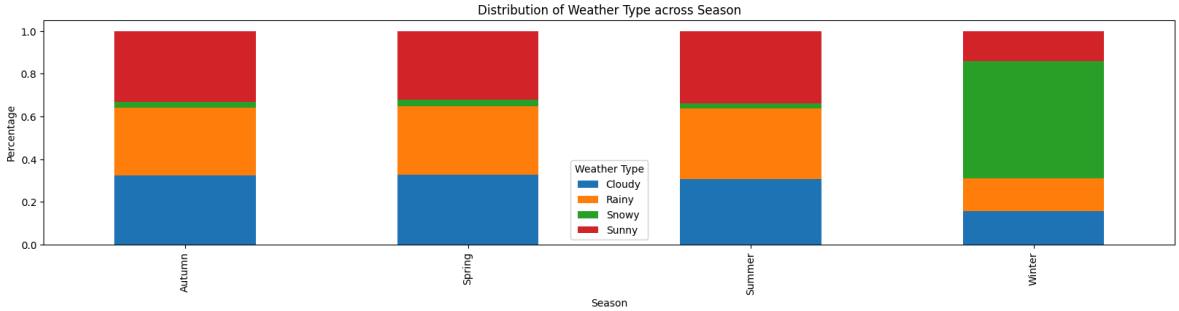
```
#distribution of weather type over all columns
distribution_of_target("Weather Type", df)
```

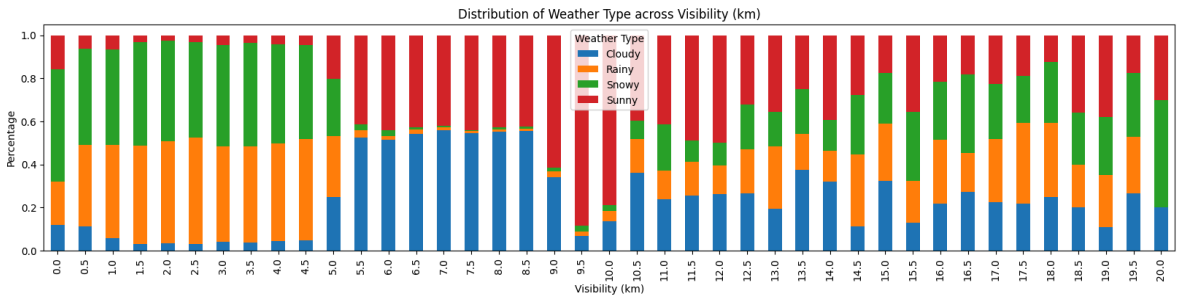












```
#Distribution of cloud cover over all coulmns
distribution_of_target("Cloud Cover", df)
```


```
df['Temp_Humidity_Interaction'] = df['Temperature'] * df['Humidity']
df['Wind_Speed_Squared'] = df['Wind Speed']**2
```

```
# Using the Steadman's apparent temperature formula for "feels like" tempera
# Formula: AT = 0.885 * Temperature - 22.4 + (1.20 * Humidity + 0.13 * Tempe
# The formula is simplified and may not be perfectly accurate for all condit
df['Feels_Like_Temp'] = 0.885 * df['Temperature'] - 22.4 + (1.20 * df['Humid
display(df.head())
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	UV Index	S
0	14	73	9.5	82	partly cloudy	1010.82	2	
1	39	96	8.5	71	partly cloudy	1011.43	7	
2	30	64	7.0	16	clear	1018.72	5	
3	38	83	1.5	82	clear	1026.25	7	
4	27	74	17.0	66	overcast	990.67	1	

Task

Analyze the provided weather data to classify different weather conditions.

Double-click (or enter) to edit

Data preprocessing

Subtask:

Handle categorical variables by encoding them into numerical representations.

```
for column in df.columns:
    if df[column].dtype == 'object':
        le = LabelEncoder()
        df[column] = le.fit_transform(df[column])

df.head()
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	UV Index	Sea
0	14	73	9.5	82	3	1010.82	2	
1	39	96	8.5	71	3	1011.43	7	
2	30	64	7.0	16	0	1018.72	5	
3	38	83	1.5	82	0	1026.25	7	
4	27	74	17.0	66	2	990.67	1	

Next steps: Generate code with df | New interactive sheet

```
for column in df.columns:
    if df[column].dtype == 'object':
        le = LabelEncoder()
        df[column] = le.fit_transform(df[column])

df.head()
```

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	UV Index	Sea
0	14	73	9.5	82	3	1010.82	2	
1	39	96	8.5	71	3	1011.43	7	
2	30	64	7.0	16	0	1018.72	5	
3	38	83	1.5	82	0	1026.25	7	
4	27	74	17.0	66	2	990.67	1	

Next steps: Generate code with df | New interactive sheet

Feature engineering

Subtask:

Create new features that might be useful for analysis or modeling.

```
df['Temp_Humidity_Interaction'] = df['Temperature'] * df['Humidity']
df['Wind_Speed_Squared'] = df['Wind Speed']**2
# Using the Steadman's apparent temperature formula for "feels like" tempera
# Formula: AT = 0.885 * Temperature - 22.4 + (1.20 * Humidity + 0.13 * Tempe
# The formula is simplified and may not be perfectly accurate for all condit
df['Feels_Like_Temp'] = 0.885 * df['Temperature'] - 22.4 + (1.20 * df['Humid
display(df.head())
```

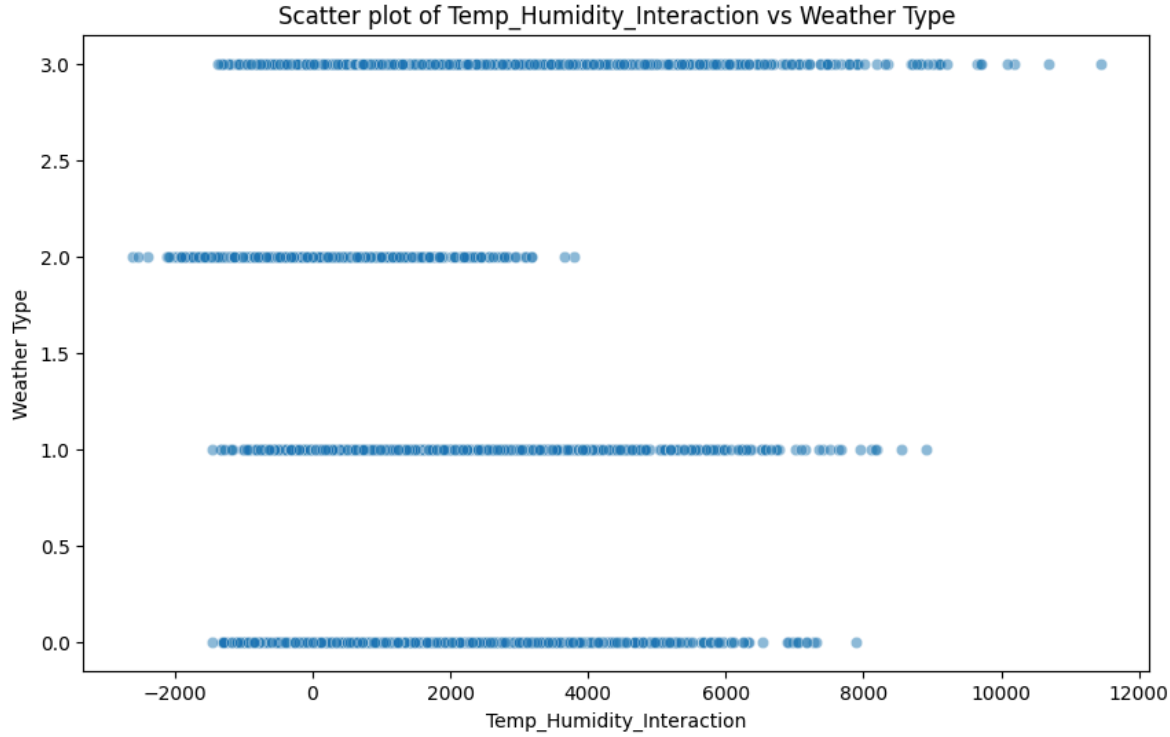
	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	UV Index	Sea
0	14	73	9.5	82	3	1010.82	2	
1	39	96	8.5	71	3	1011.43	7	
2	30	64	7.0	16	0	1018.72	5	
3	38	83	1.5	82	0	1026.25	7	
4	27	74	17.0	66	2	990.67	1	

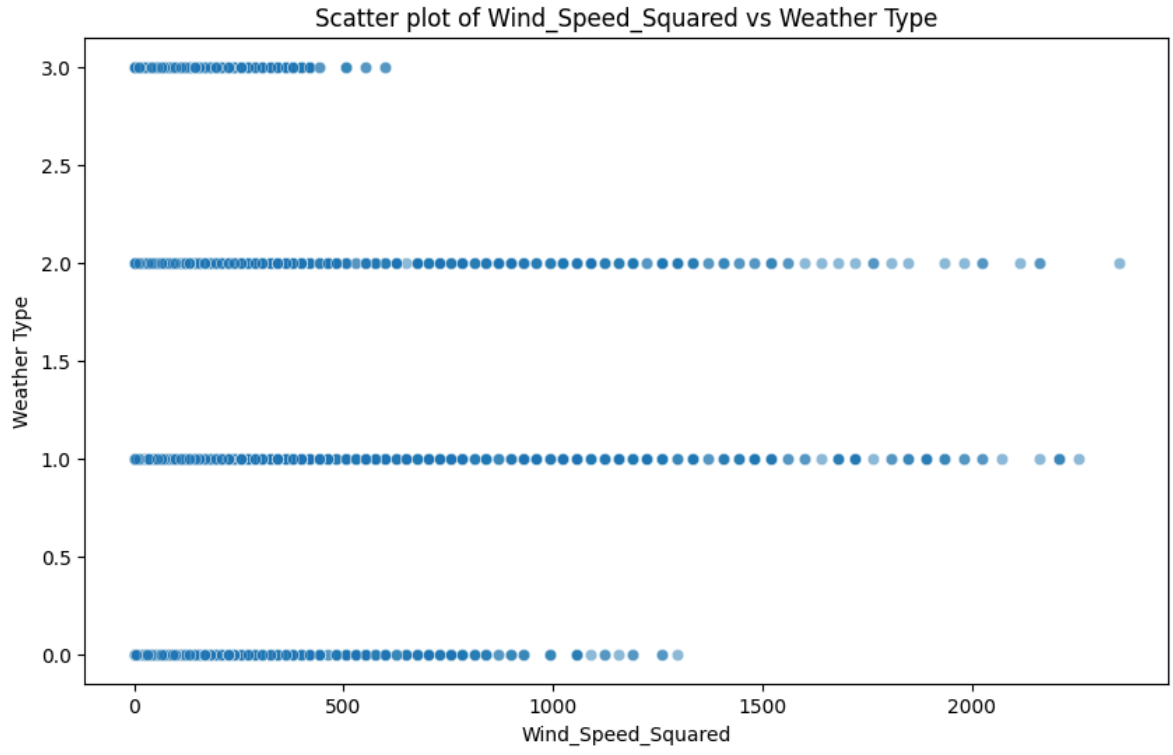
Exploratory data analysis (eda)

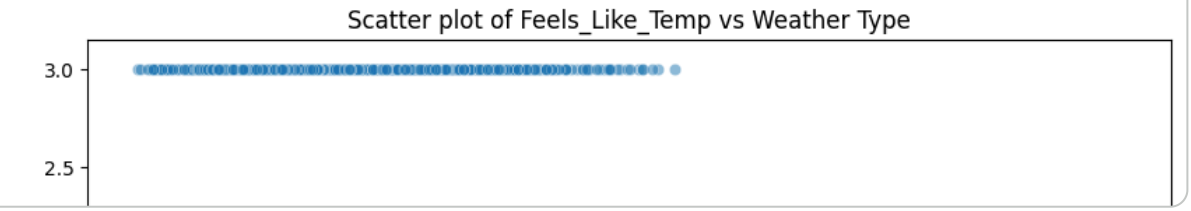
Subtask:

Continue exploring relationships between features and the target variable with visualizations and statistical tests.

```
engineered_features = ['Temp_Humidity_Interaction', 'Wind_Speed_Squared', 'F
target_variable = 'Weather Type'

for feature in engineered_features:
    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=df, x=feature, y=target_variable, alpha=0.5)
    plt.title(f'Scatter plot of {feature} vs {target_variable}')
    plt.xlabel(feature)
    plt.ylabel(target_variable)
    plt.show()
```

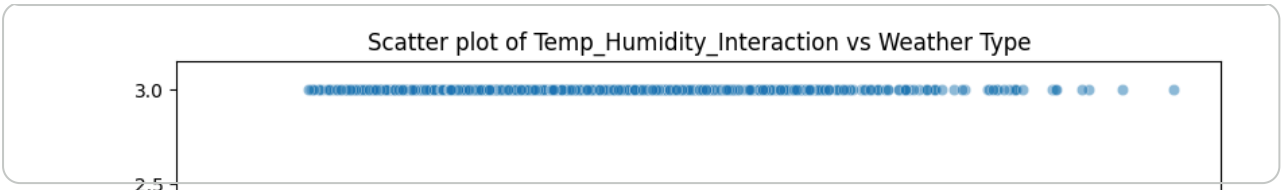







```
engineered_features = ['Temp_Humidity_Interaction', 'Wind_Speed_Squared', 'F
target_variable = 'Weather Type'

for feature in engineered_features:
    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=df, x=feature, y=target_variable, alpha=0.5)
    plt.title(f'Scatter plot of {feature} vs {target_variable}')
    plt.xlabel(feature)
```

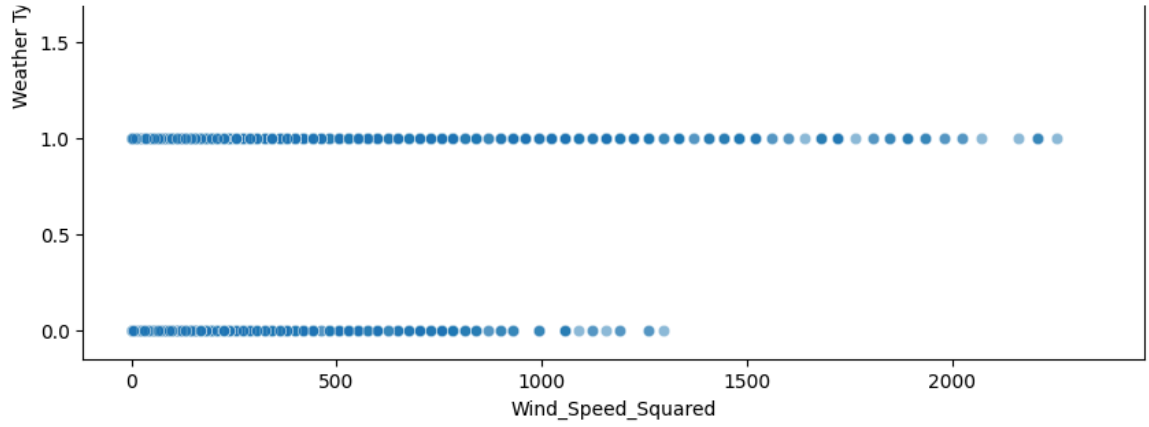
```
        plt.ylabel(target_variable)
        plt.show()
```

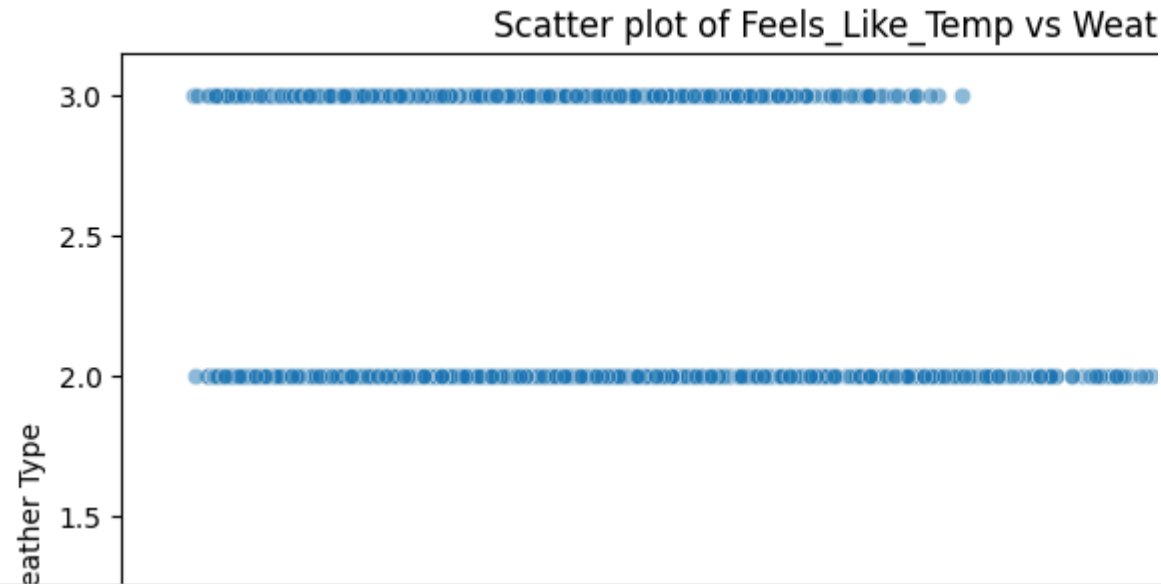


Reasoning: Generate box plots or violin plots to examine the distribution of the engineered numerical features across different categories of the original categorical columns to understand how these features vary with different weather conditions, seasons, and locations.

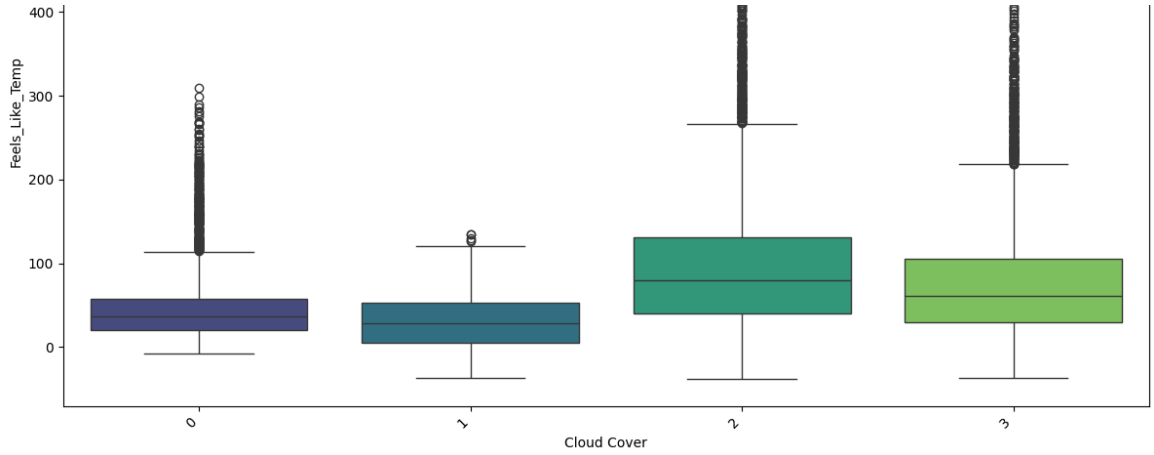
```
Start coding or generate with AI.
```

```
categorical_features = ['Cloud Cover', 'Season', 'Location', 'Weather Type']
engineered_features = ['Temp_Humidity_Interaction', 'Wind_Speed_Squared', 'F

for cat_feature in categorical_features:
    for eng_feature in engineered_features:
        plt.figure(figsize=(12, 7))
        sns.boxplot(data=df, x=cat_feature, y=eng_feature, palette='viridis'
        plt.title(f'Distribution of {eng_feature} by {cat_feature}')
        plt.xlabel(cat_feature)
        plt.ylabel(eng_feature)
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()
```

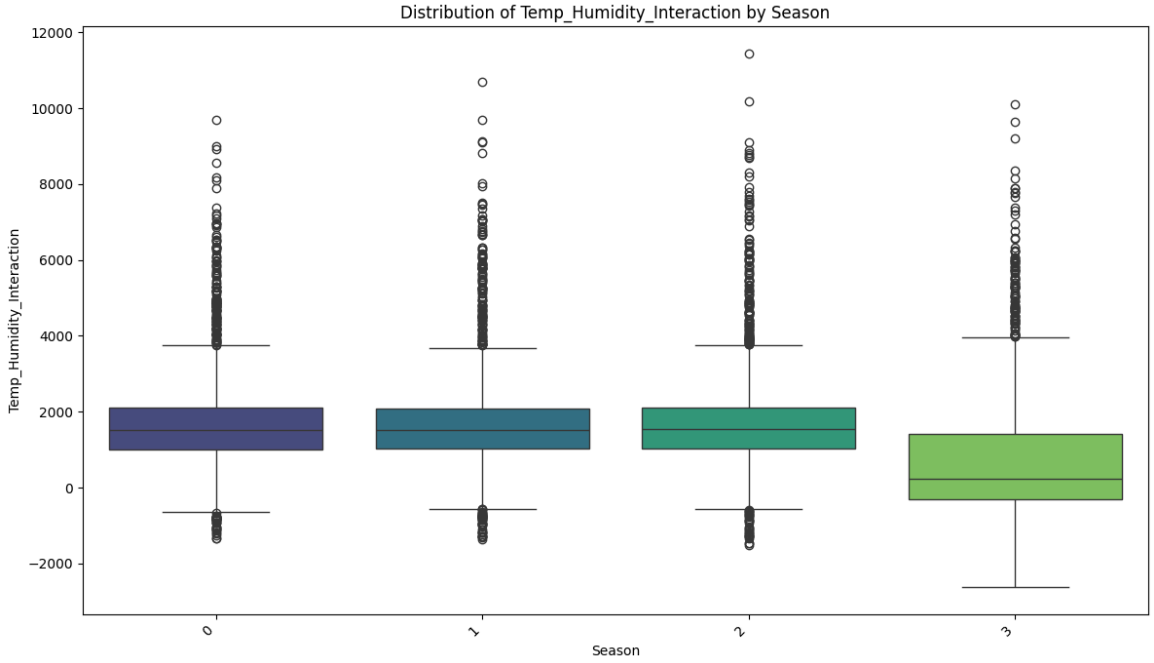




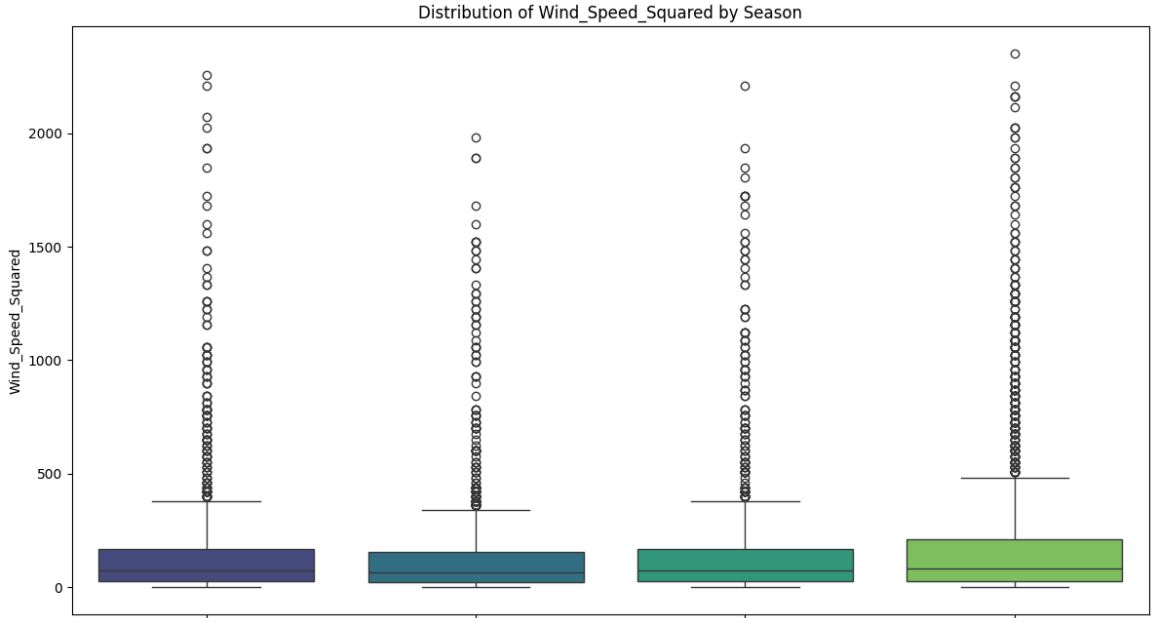

W
1.0



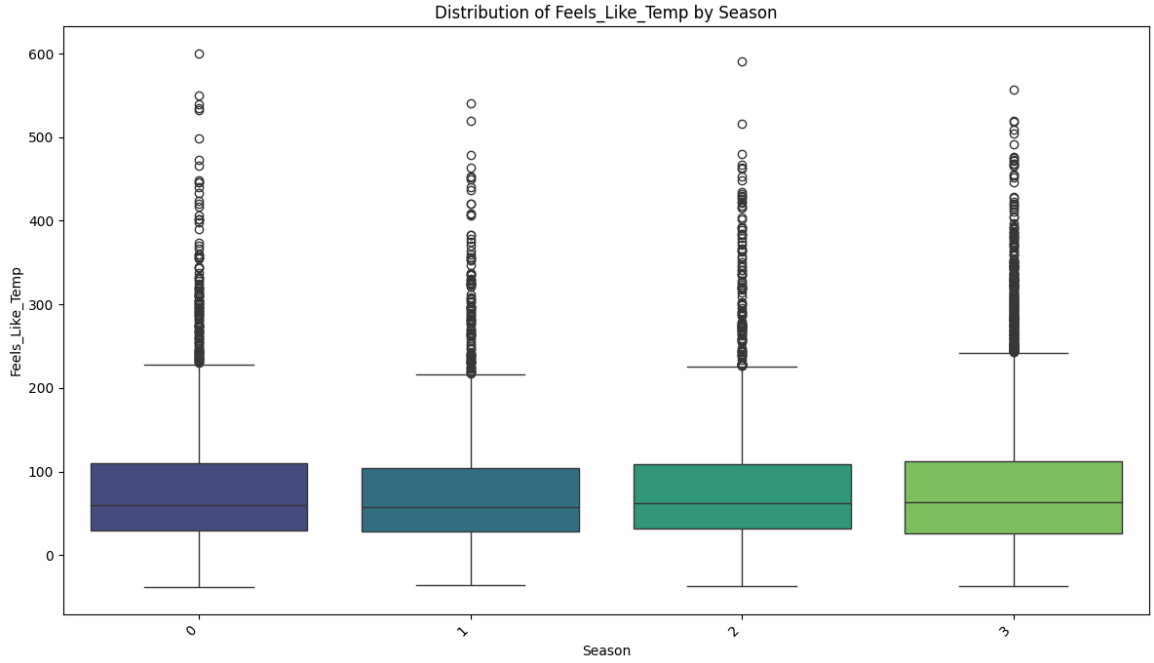

```
/tmp/ipython-input-2787046026.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed

  sns.boxplot(data=df, x=cat_feature, y=eng_feature, palette='viridis')
```



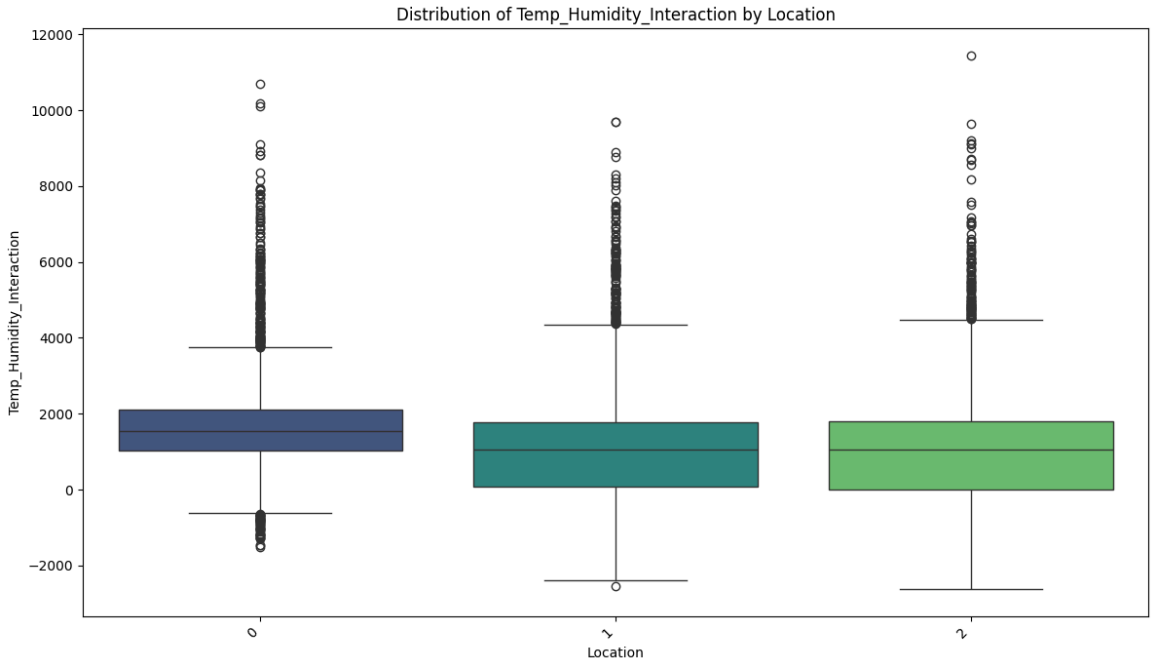

```
/tmp/ipython-input-2787046026.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed

  sns.boxplot(data=df, x=cat_feature, y=eng_feature, palette='viridis')
```



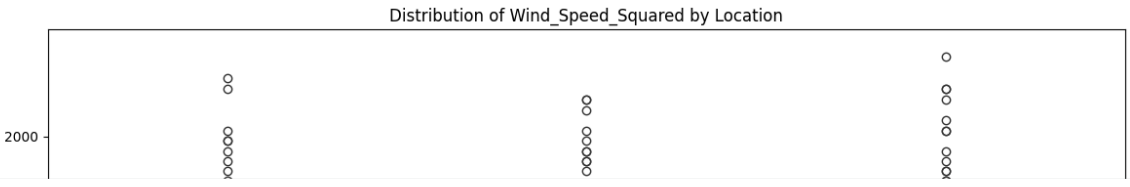

```
/tmp/ipython-input-2787046026.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed

  sns.boxplot(data=df, x=cat_feature, y=eng_feature, palette='viridis')
```




```
/tmp/ipython-input-2787046026.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed

  sns.boxplot(data=df, x=cat_feature, y=eng_feature, palette='viridis')
```




```
/tmp/ipython-input-2787046026.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed

  sns.boxplot(data=df, x=cat_feature, y=eng_feature, palette='viridis')
```



Reasoning: Calculate and display the correlation coefficients between the new engineered features and the 'Weather Type' column to quantify their linear relationships.

```
engineered_features = ['Temp_Humidity_Interaction', 'Wind_Speed_Squared', 'F
target_variable = 'Weather Type'

# Calculate correlation coefficients
correlations = df[engineered_features + [target_variable]].corr()

# Display correlations with the target variable
print("Correlation of Engineered Features with Weather Type:")
print(correlations[target_variable].drop(target_variable))
```

```
Correlation of Engineered Features with Weather Type:
Temp_Humidity_Interaction   -0.122754
Wind_Speed_Squared          -0.113527
Feels_Like_Temp             -0.168210
Name: Weather Type, dtype: float64
```

```
  sns.boxplot(data=df, x=cat_feature, y=eng_feature, palette='viridis')
```

Model selection

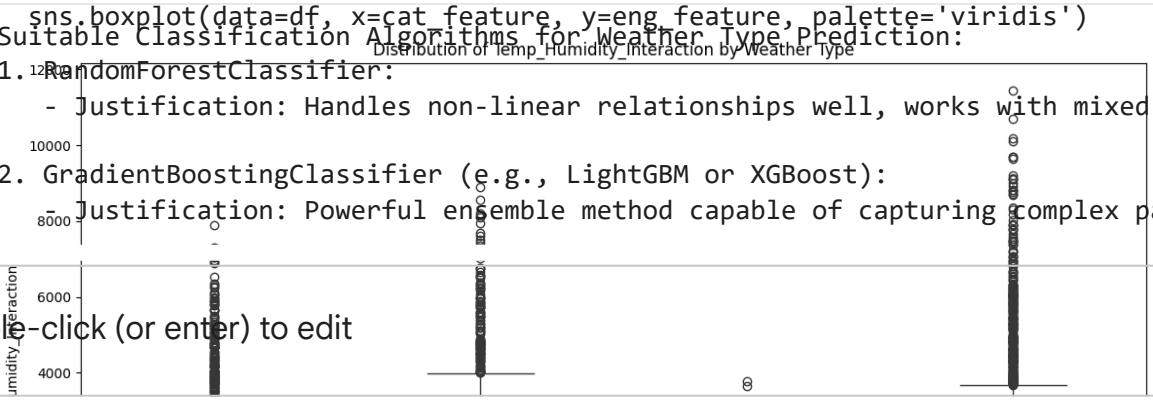
Subtask:

Choose appropriate machine learning models for weather classification.

```
# Listing two distinct algorithms and justifications:

print("Suitable Classification Algorithms for Weather Type Prediction:")
print("1. RandomForestClassifier:")
print("   - Justification: Handles non-linear relationships well, works with
print("\n2. GradientBoostingClassifier (e.g., LightGBM or XGBoost):")
print("   - Justification: Powerful ensemble method capable of capturing com
```

```
Suitable Classification Algorithms for Weather Type Prediction:
1. RandomForestClassifier:
   - Justification: Handles non-linear relationships well, works with mixed f

2. GradientBoostingClassifier (e.g., LightGBM or XGBoost):
   - Justification: Powerful ensemble method capable of capturing complex pat
```

```
  sns.boxplot(data=df, x=cat_feature, y=eng_feature, palette='viridis')
```



Double-click (or enter) to edit

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

```
from sklearn.preprocessing import StandardScaler

# Define features (X) and target (y)
X = df.drop(['Weather Type', 'Season_Numerical'], axis=1) # Drop target and
y = df['Weather Type']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Scale numerical features (important for models like Logistic Regression, S
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'Support Vector Machine': SVC(random_state=42)
}

print("Models initialized and data split/scaled successfully. Ready for trai
```

```
Models initialized and data split/scaled successfully. Ready for training and
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f

results_table = PrettyTable()
results_table.field_names = ["Model", "Accuracy", "Precision", "Recall", "F1

for name, model in models.items():
    print(f"\n--- Training {name} ---")
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted', zero_div
    recall = recall_score(y_test, y_pred, average='weighted', zero_division=
    f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)

    results_table.add_row([name, f"{accuracy:.4f}", f"{precision:.4f}", f"{r

    print(f"Evaluation for {name}:")
    print(f"  Accuracy: {accuracy:.4f}")
    print(f"  Precision: {precision:.4f}")
    print(f"  Recall: {recall:.4f}")
    print(f"  F1-Score: {f1:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred, zero_division=0))
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
```

```
print("\n--- Model Performance Summary ---")
print(results_table)
```

```
  Precision: 0.9160
```