In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.preprocessing import LabelEncoder, MinMaxScaler, OneHotEncoder
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_m

import warnings
warnings.filterwarnings('ignore')
```

LOADING THE DATA INTO NOTEBOOK

In [2]:
```
df = pd.read_csv("C://Users//quays//Desktop//weather_classification_data.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	temperature	humidity	wind_speed	precipitation(%)	cloud_cover	atmospheric_pressure	uv_i
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	

In [4]:
```
df.nunique()
```

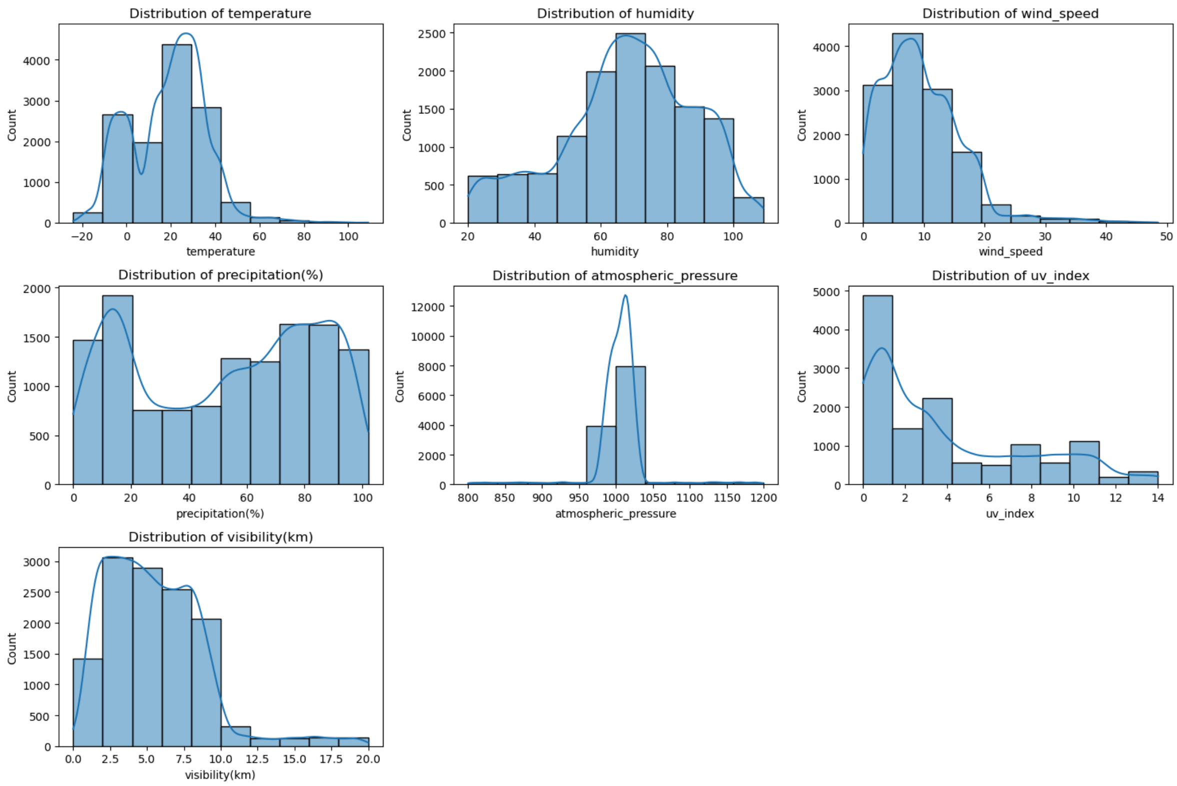
Out[4]:
```
temperature              121
humidity                  90
wind_speed                95
precipitation(%)         102
cloud_cover                4
atmospheric_pressure    5421
uv_index                  15
season                     4
visibility(km)            41
location                   3
weather_type               4
dtype: int64
```

DATA CLEANING AND PREPROCESSING

In [5]:
```
# checking for null values
df.info()
```

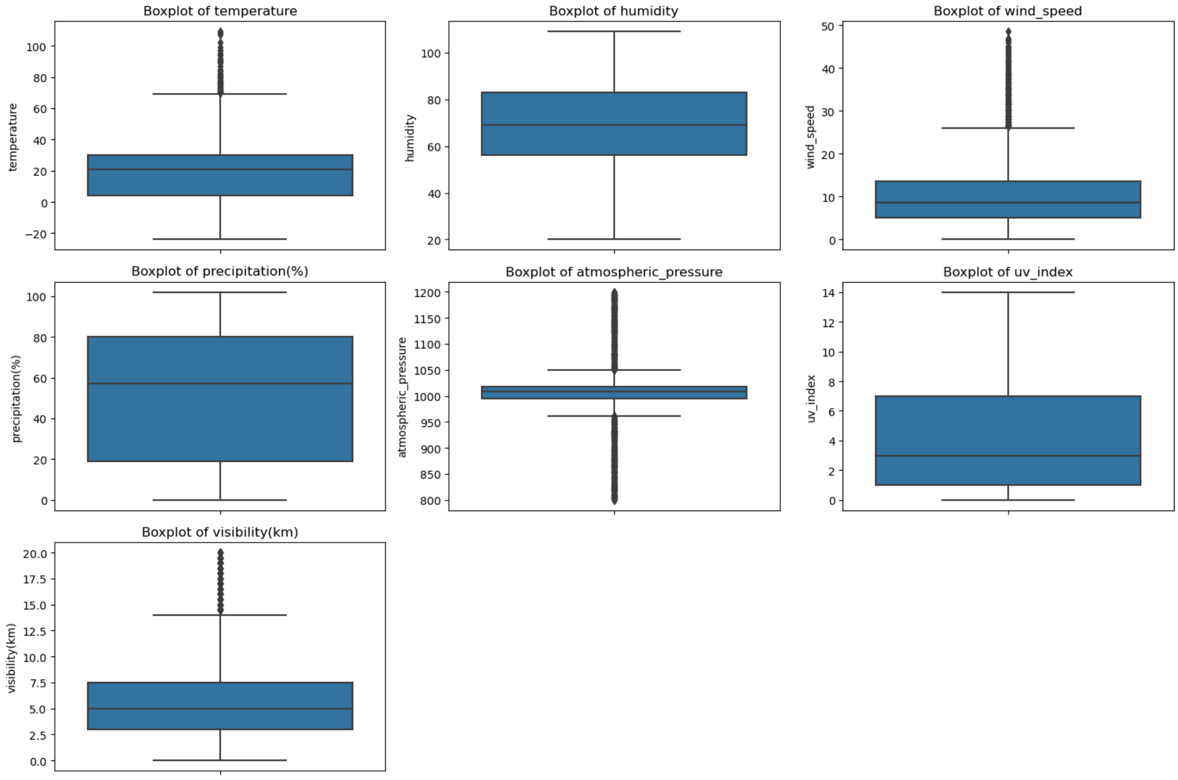
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12846 entries, 0 to 12845
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   temperature           12846 non-null  int64  
 1   humidity              12846 non-null  int64  
 2   wind_speed            12846 non-null  float64
 3   precipitation(%)      12846 non-null  int64  
 4   cloud_cover           12846 non-null  object 
 5   atmospheric_pressure  12846 non-null  float64
 6   uv_index              12846 non-null  int64  
 7   season                12846 non-null  object 
 8   visibility(km)        12846 non-null  float64
 9   location              12846 non-null  object 
 10  weather_type          12846 non-null  object 
dtypes: float64(3), int64(4), object(4)
memory usage: 1.1+ MB
```

In [6]:
```
# data distribution

num_cols = df.select_dtypes(include=["int64", "float64"]).columns # Select onl

# Plot distribution for each numerical column
plt.figure(figsize=(15, 10))
for i, col in enumerate(num_cols, 1):
    plt.subplot(3, 3, i)  # adjust grid size if more/fewer columns
    sns.histplot(df[col], kde=True, bins=10)
    plt.title(f"Distribution of {col}")

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



CHECKING FOR OUTLIERS AND REMOVING THEM

In [7]:
```
# Plot boxplots for each numerical column
plt.figure(figsize=(15, 10))
for i, col in enumerate(num_cols, 1):
    plt.subplot(3, 3, i)  # adjust grid size depending on number of columns
    sns.boxplot(y=df[col])
    plt.title(f"Boxplot of {col}")

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



In [8]:
```
def remove_outliers(df):
    num_cols = df.select_dtypes(include=["int64", "float64"]).columns
    capped_df = df.copy()

    for col in num_cols:
        Q1 = capped_df[col].quantile(0.25)
        Q3 = capped_df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        capped_df[col] = capped_df[col].clip(lower=lower_bound, upper=upper_bc

    return capped_df

# Run it
cleaned_df = remove_outliers(df)
```

In [9]:
```
cleaned_df.shape
```

Out[9]: (12846, 11)

DISPLAYING THE CLEANED DATA

In [10]:
```
cleaned_df.head()
```

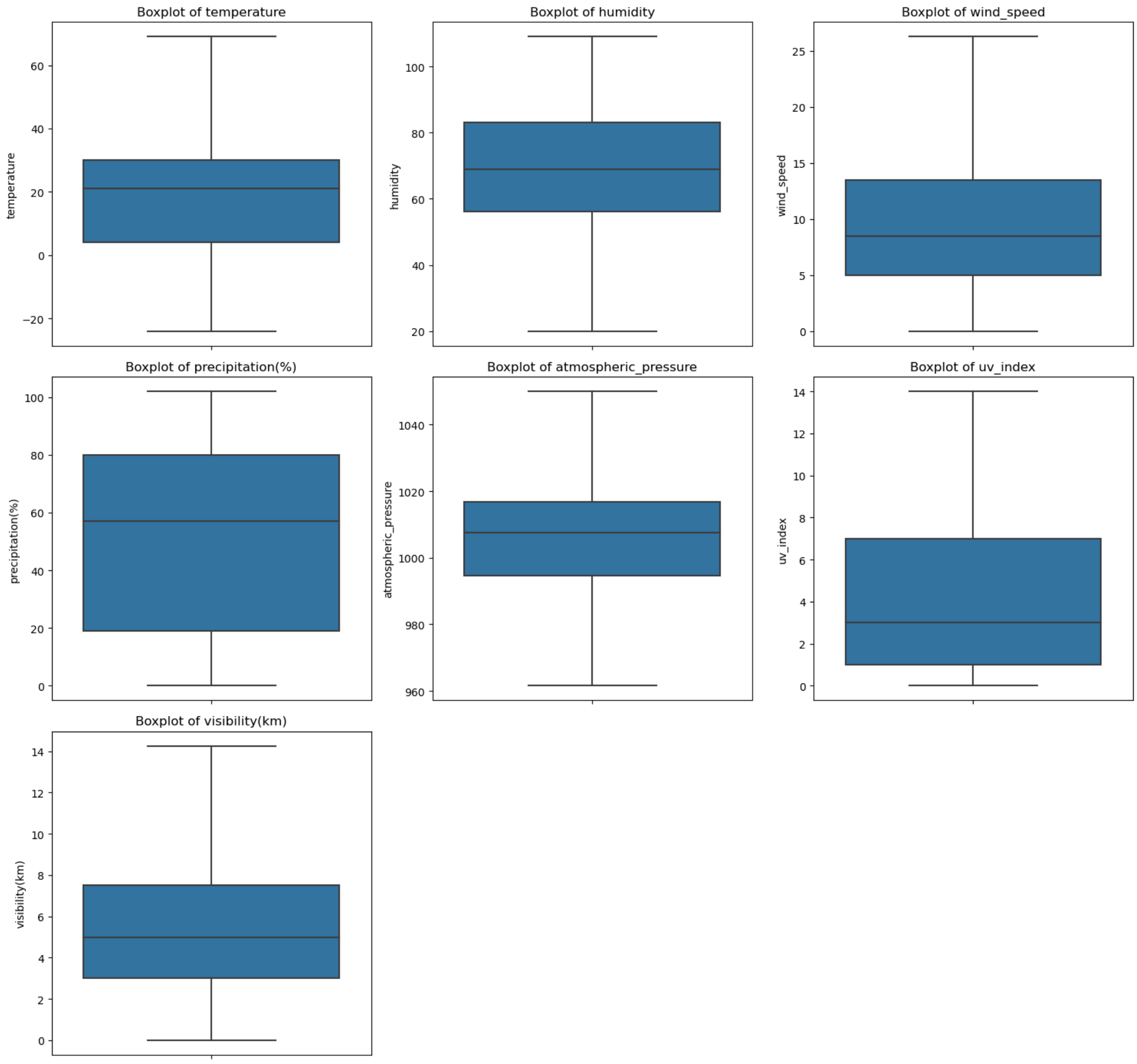
Out[10]:

	temperature	humidity	wind_speed	precipitation(%)	cloud_cover	atmospheric_pressure	uv_i
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	

REVIEWING THE DATA POINTS DISTRIBUTION

In [ ]:

In [11]:
```
num_cols = cleaned_df.select_dtypes(include=["int64", "float64"]).columns

plt.figure(figsize=(15, 14))
for i, col in enumerate(num_cols, 1):
    plt.subplot(3, 3, i)  # adjust grid size depending on number of columns
    sns.boxplot(y=cleaned_df[col])
    plt.title(f"Boxplot of {col}")

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



SPLITTING THE CLEANED DATA INTO TRAIN , VALIDATION AND TEST

In [12]:
```
train_validation_data, test_data = train_test_split(cleaned_df, test_size=0.2,
train_data, validation_data = train_test_split(train_validation_data, test_siz
```

In [13]:
```
print('test data size:', test_data.shape)
print('train data size: ',   train_data.shape)
print('validation data size:', validation_data.shape)
```

```
test data size: (2570, 11)
train data size:  (7707, 11)
validation data size: (2569, 11)
```

1. TRAIN DATASET

In [72]:
```
train_data.head(20)
```

Out[72]:

	temperature	humidity	wind_speed	precipitation(%)	cloud_cover	atmospheric_pressure
6356	24	88	17.5	58	overcast	1007.43
11321	60	108	20.5	76	partly cloudy	1010.49
8416	33	67	1.5	41	overcast	1015.13
7974	12	84	19.0	77	overcast	995.09
4376	20	99	16.5	72	overcast	1014.90
10386	-4	73	16.0	91	overcast	995.19
1140	41	20	6.5	3	partly cloudy	1024.98
11620	3	60	5.5	84	overcast	998.39
5288	1	83	1.5	67	overcast	984.98
7195	47	20	14.5	67	cloudy	961.57
9106	-14	44	8.5	55	partly cloudy	1049.97
3050	10	73	10.0	24	overcast	1011.49
12770	19	93	12.0	74	clear	1020.70
1094	11	60	9.0	47	overcast	1017.79
5443	38	63	5.5	11	clear	1013.87
6158	18	69	9.0	39	partly cloudy	1013.45
6104	32	58	2.0	20	partly cloudy	1000.98
12820	15	66	8.0	59	overcast	1006.99
4499	10	99	5.5	92	partly cloudy	1000.65
11005	31	94	15.0	67	partly cloudy	1004.99

In [15]:
```
train_data.describe()
```

Out[15]:

	temperature	humidity	wind_speed	precipitation(%)	atmospheric_pressure	uv_ind
count	7707.000000	7707.000000	7707.000000	7707.000000	7707.000000	7707.0000
mean	18.997924	68.065655	9.472038	51.796678	1006.160529	3.9003
std	16.680662	19.912209	6.078570	31.156630	17.152382	3.7718
min	-24.000000	20.000000	0.000000	0.000000	961.570000	0.0000
25%	4.000000	56.000000	5.000000	19.000000	995.085000	1.0000
50%	21.000000	69.000000	8.500000	56.000000	1007.840000	3.0000
75%	30.000000	82.000000	13.500000	80.000000	1016.870000	7.0000
max	69.000000	109.000000	26.250000	102.000000	1049.970000	14.0000

In [16]:
```
train_data_X = train_data[['temperature','humidity','wind_speed','precipitatio
                           'atmospheric_pressure','uv_index','season','visibility(km)','l

train_data_y = train_data['weather_type']
```

2. VALIDATION DATASET

In [17]:
```
validation_data.head()
```

Out[17]:

	temperature	humidity	wind_speed	precipitation(%)	cloud_cover	atmospheric_pressure
6160	18	60	16.0	62	partly cloudy	990.89
10700	34	47	1.0	4	clear	1018.05
1739	1	88	2.5	70	overcast	998.54
11774	2	99	4.5	62	overcast	992.81
1503	32	60	11.0	49	partly cloudy	1007.94

In [18]:
```
validation_data.describe()
```

Out[18]:

	temperature	humidity	wind_speed	precipitation(%)	atmospheric_pressure	uv_ind
count	2569.000000	2569.000000	2569.000000	2569.000000	2569.000000	2569.0000
mean	18.586220	67.643441	9.309459	52.367458	1005.594204	3.9739
std	17.161152	20.430486	5.979183	31.504890	17.807310	3.8645
min	-24.000000	20.000000	0.000000	0.000000	961.570000	0.0000
25%	3.000000	56.000000	5.000000	19.000000	993.830000	1.0000
50%	21.000000	69.000000	8.500000	57.000000	1007.390000	3.0000
75%	31.000000	82.000000	13.000000	81.000000	1016.660000	7.0000
max	69.000000	109.000000	26.250000	102.000000	1049.970000	14.0000

In [20]:
```
validation_data_X = validation_data[['temperature','humidity','wind_speed','pr
                    'atmospheric_pressure','uv_index','season','visibility(km)','l

validation_data_y = validation_data['weather_type']
```

3. TEST DATASET

In [21]:
```
test_data.head()
```

Out[21]:

	temperature	humidity	wind_speed	precipitation(%)	cloud_cover	atmospheric_pressure
5419	-1	93	5.0	85	partly cloudy	993.52
3830	-4	84	19.0	84	overcast	996.43
10998	-13	20	9.0	19	overcast	987.05
8119	47	48	1.5	68	partly cloudy	1049.97
9682	-7	96	4.0	76	overcast	983.19

In [22]:
```
test_data.describe()
```

Out[22]:

	temperature	humidity	wind_speed	precipitation(%)	atmospheric_pressure	uv_ind
count	2570.000000	2570.000000	2570.000000	2570.000000	2570.000000	2570.0000
mean	18.509339	68.825681	9.535895	53.345136	1006.227424	3.9058
std	16.611704	20.174423	6.083410	30.964179	17.338642	3.8073
min	-24.000000	20.000000	0.000000	0.000000	961.570000	0.0000
25%	4.000000	58.000000	5.000000	21.000000	994.737500	1.0000
50%	21.000000	70.000000	8.500000	58.000000	1007.695000	3.0000
75%	30.000000	84.000000	13.500000	81.000000	1016.752500	6.0000
max	69.000000	109.000000	26.250000	102.000000	1049.970000	14.0000

In [23]:
```
test_data_X = test_data[['temperature','humidity','wind_speed','precipitation(
                    'atmospheric_pressure','uv_index','season','visibility(km)','l

test_data_y = test_data['weather_type']
```

ONE HOT ENCODING THE COLUMNS

In [24]:
```
# categorical columns
categorical_cols = ["cloud_cover", "season", "location"]
```

creating the one hot - And the MinMax Scalar Objects

In [25]:
```
# Create OneHotEncoder object
encoder = OneHotEncoder(sparse=False, drop=None)
```

In [26]:
```
# Create MinMaxScaler object
scaler = MinMaxScaler()
```

------- TRAINING DATA ------

** ONE HOT ENCODING TRAINING CATEGORICAL DATA AND STANDARDIZING THE VALUES (0 - 1) **

ENCODING

In [27]:
```
# Separate categorical and numerical data
X_categorical = train_data_X[categorical_cols]
X_numeric = train_data_X.drop(columns=categorical_cols)
```

In [28]:
```
# Fit and transform categorical data
encoded_array = encoder.fit_transform(X_categorical)
```

In [29]:
```
encoded_array[:5]
```

Out[29]:
```
array([[0., 0., 1., 0., 0., 1., 0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 1., 0., 0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0.]])
```

In [30]:
```
# Get feature names (must match categorical_cols)
encoded_df = pd.DataFrame(
    encoded_array,
    columns=encoder.get_feature_names_out(categorical_cols),
    index=train_data_X.index
)
```

STANDARDIZING

In [31]:
```
scaled_array = scaler.fit_transform(X_numeric)
```

In [32]:
```
# Convert back to DataFrame with original column names
scaled_df = pd.DataFrame(
    scaled_array,
    columns=X_numeric.columns,
    index=train_data_X.index)
```

In [33]:
```
# Combine encoded categorical and scaled numerical
training_data_X = pd.concat([scaled_df, encoded_df], axis=1)

training_data_X.head()
```

Out[33]:

	temperature	humidity	wind_speed	precipitation(%)	atmospheric_pressure	uv_index	vis
6356	0.516129	0.764045	0.666667	0.568627	0.518778	0.071429	
11321	0.903226	0.988764	0.780952	0.745098	0.553394	0.000000	
8416	0.612903	0.528090	0.057143	0.401961	0.605882	0.142857	
7974	0.387097	0.719101	0.723810	0.754902	0.379186	0.142857	
4376	0.473118	0.887640	0.628571	0.705882	0.603281	0.142857	

In [71]:
```
training_data_X[training_data_X['temperature'] < 0]
```

Out[71]:

temperature	humidity	wind_speed	precipitation(%)	atmospheric_pressure	uv_index	visibility

In [34]:
```
training_data_y = train_data_y
training_data_y.head()
```

Out[34]:
```
6356       Rainy
11321      Sunny
8416      Cloudy
7974       Rainy
4376       Rainy
Name: weather_type, dtype: object
```

------ VALIDATION DATA ------

** ONE HOT ENCODING VALIDATION CATEGORICAL DATA AND STANDARDIZING THE VALUES (0 - 1) **

In [35]:
```
# Separate categorical and numerical data
val_X_categorical = validation_data_X[categorical_cols]
val_X_numeric = validation_data_X.drop(columns=categorical_cols)
```

In [36]:
```
# Fit and transform categorical data
val_encoded_array = encoder.fit_transform(val_X_categorical)
```

In [37]:
```
val_encoded_array[:5]
```

Out[37]:
```
array([[0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 0., 1., 0., 1., 0.],
       [0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 1.]])
```

In [38]:
```
# Get feature names (must match categorical_cols)
val_encoded_df = pd.DataFrame(
    val_encoded_array,
    columns=encoder.get_feature_names_out(categorical_cols),
    index=validation_data_X.index
)
```

STANDARDIZATION

In [39]:
```
val_scaled_array = scaler.fit_transform(val_X_numeric)
```

In [40]:
```
# Convert back to DataFrame with original column names
val_scaled_df = pd.DataFrame(
    val_scaled_array,
    columns=val_X_numeric.columns,
    index=validation_data_X.index)
```

In [41]:
```
# Combine encoded categorical and scaled numerical
validation_data_X = pd.concat([val_scaled_df, val_encoded_df], axis=1)

validation_data_X.head()
```

Out[41]:

	temperature	humidity	wind_speed	precipitation(%)	atmospheric_pressure	uv_index	vis
6160	0.451613	0.449438	0.609524	0.607843	0.331674	0.071429	
10700	0.623656	0.303371	0.038095	0.039216	0.638914	0.642857	
1739	0.268817	0.764045	0.095238	0.686275	0.418213	0.071429	
11774	0.279570	0.887640	0.171429	0.607843	0.353394	0.000000	
1503	0.602151	0.449438	0.419048	0.480392	0.524548	0.071429	

In [42]:
```
validation_data_y.head()
```

Out[42]:
```
6160       Rainy
10700      Sunny
1739       Snowy
11774      Snowy
1503      Cloudy
Name: weather_type, dtype: object
```

------ TESTING DATA ------

** ONE HOT ENCODING VALIDATION CATEGORICAL DATA AND STANDARDIZING THE VALUES (0 - 1) **

In [43]:
```
# Separate categorical and numerical data
test_X_categorical = test_data_X[categorical_cols]
test_X_numeric = test_data_X.drop(columns=categorical_cols)
```

In [44]:
```
# Fit and transform categorical data
test_encoded_array = encoder.fit_transform(test_X_categorical)
```

In [45]:
```
test_encoded_array[:5]
```

Out[45]:
```
array([[0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0., 1., 0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1.]])
```

In [46]:
```
# Get feature names (must match categorical_cols)
test_encoded_df = pd.DataFrame(
    test_encoded_array,
    columns=encoder.get_feature_names_out(categorical_cols),
    index=test_data_X.index
)
```

STANDARDIZATION

In [47]:
```
test_scaled_array = scaler.fit_transform(test_X_numeric)
```

In [48]:
```
# Convert back to DataFrame with original column names
test_scaled_df = pd.DataFrame(
    test_scaled_array,
    columns=test_X_numeric.columns,
    index=test_data_X.index)
```

In [49]:
```
# Combine encoded categorical and scaled numerical
test_data_X = pd.concat([test_scaled_df, test_encoded_df], axis=1)

test_data_X.head()
```

Out[49]:

	temperature	humidity	wind_speed	precipitation(%)	atmospheric_pressure	uv_index	vis
5419	0.247312	0.820225	0.190476	0.833333	0.361425	0.071429	
3830	0.215054	0.719101	0.723810	0.823529	0.394344	0.071429	
10998	0.118280	0.000000	0.342857	0.186275	0.288235	0.500000	
8119	0.763441	0.314607	0.057143	0.666667	1.000000	0.928571	
9682	0.182796	0.853933	0.152381	0.745098	0.244570	0.071429	

In [50]:
```
test_data_y.head()
```

Out[50]:
```
5419       Snowy
3830       Snowy
10998      Snowy
8119      Cloudy
9682       Snowy
Name: weather_type, dtype: object
```

DECISION TREE CLASSIFIER MODEL

CREATING THE MODEL

In [51]:
```
model = DecisionTreeClassifier(max_depth= 7)
```

TRAINING

In [52]:
```
model.fit(training_data_X, training_data_y)
```

Out[52]:
```
▾       DecisionTreeClassifier
DecisionTreeClassifier(max_depth=7)
```

MAKING PREDICTIONS

In [53]:
```
# model predictions
model_pred = model.predict(training_data_X)
model_pred
```

Out[53]:
```
array(['Rainy', 'Cloudy', 'Cloudy', ..., 'Snowy', 'Sunny', 'Snowy'],
      dtype=object)
```

In [54]:
```
# accuracy
acc = accuracy_score(training_data_y, model_pred)
acc
```

Out[54]:
```
0.9375892046191774
```

MODEL VALIDATION

In [55]:
```
# VALIDATION
```

In [56]:
```
y_val = model.predict(validation_data_X)
y_val
```

Out[56]:
```
array(['Rainy', 'Sunny', 'Snowy', ..., 'Sunny', 'Snowy', 'Snowy'],
      dtype=object)
```

In [57]:
```
# Accuracy
acc_val = accuracy_score(y_val, validation_data_y)
acc_val
```

Out[57]:
```
0.9077462047489295
```

In [58]:
```
conf_mat_val = confusion_matrix(y_val, validation_data_y)
conf_mat_val
```

Out[58]:
```
array([[555,  28,  26,  29],
       [ 26, 557,  10,  20],
       [  3,   2, 615,   5],
       [ 27,  34,  27, 605]], dtype=int64)
```

In [59]:
```
# TESTING
y_test_pred = model.predict(test_data_X)
y_test_pred
```

Out[59]:
```
array(['Snowy', 'Snowy', 'Sunny', ..., 'Snowy', 'Snowy', 'Snowy'],
      dtype=object)
```

In [60]:
```
acc_test = accuracy_score(test_data_y, y_test_pred)
acc_test
```

Out[60]:
```
0.9046692607003891
```

SAVING THE MODEL TO BE DEPLOYED

In [61]:
```
import pickle
```

In [62]:
```
filename = 'dt_model.sav'
pickle.dump(model, open(filename, 'wb'))
```

In [63]:
```
load_model = pickle.load(open(filename, 'rb'))
```

In [64]:
```
# scaled part
inputs_to_transform = scaler.transform([[18,60,16,62,990.89,1,5]])

# flatten from [[...]] → [...]
inputs_to_transform = inputs_to_transform.flatten()

# categorical part (not scaled)
inputs_not_to_transform = [0,0,0,1,0,0,1,0,0,0,1]

# join them into one list inside another list [[...]]
inputs = [list(inputs_to_transform) + inputs_not_to_transform]

print(inputs)
```

```
[[0.4516129032258065, 0.449438202247191, 0.6095238095238096, 0.6078431372549
019, 0.331674208144797, 0.07142857142857142, 0.3508771929824561, 0, 0, 0, 1,
0, 0, 1, 0, 0, 0, 1]]
```

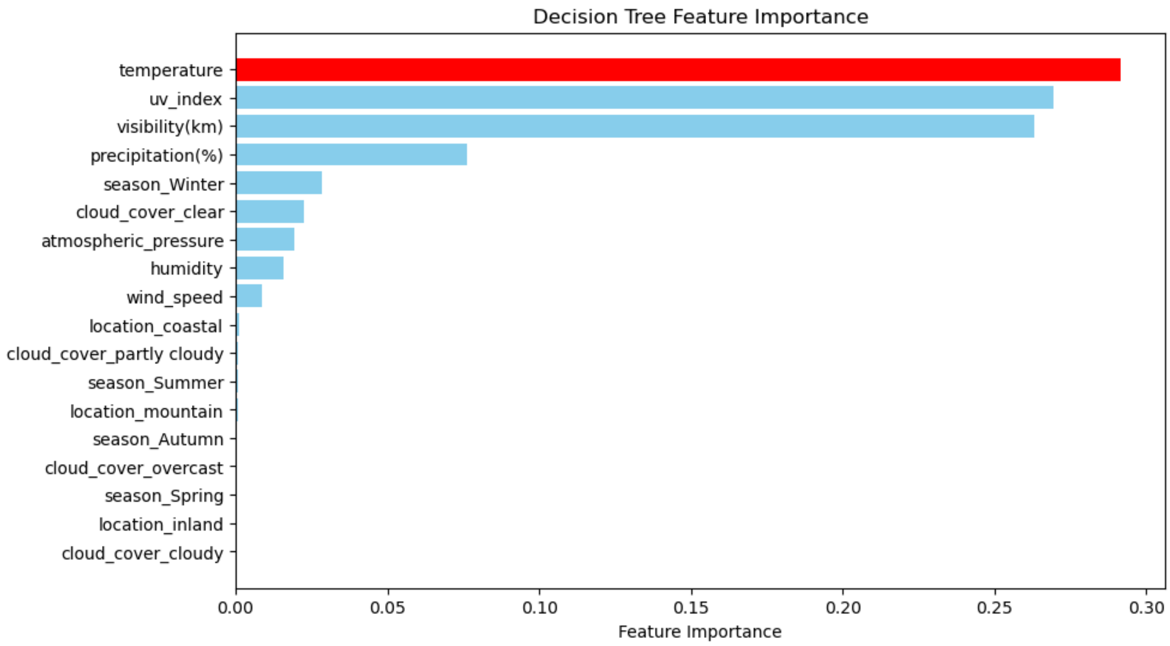
In [65]:
```
load_model.predict(inputs)[0]
```

Out[65]:
```
'Rainy'
```

FEATURE IMPORTANCE

In [66]:

```
# Feature names
feature_names = training_data_X.columns  # 1D array of column names

# Get importances
importances = model.feature_importances_
indices = np.argsort(importances)[::-1]  # descending order

# Create DataFrame
feat_importances = pd.DataFrame({
    "Feature": feature_names[indices],
    "Importance": importances[indices]
})

# Colors: red for highest, blue for the rest
colors = ['red'] + ['skyblue']*(len(feat_importances)-1)

# Plot
plt.figure(figsize=(10,6))
plt.barh(feat_importances["Feature"], feat_importances["Importance"], color=cc
plt.gca().invert_yaxis()  # highest on top
plt.xlabel("Feature Importance")
plt.title("Decision Tree Feature Importance")
plt.show()
```



In [ ]: