In [1]:
```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix , classification_report,f1_score,accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, f1_score, confusion_matrix, precision_recall
from sklearn.model_selection import GridSearchCV
```

In [4]:
```
df = pd.read_csv('weather_classification_data.csv')
df.head()
```

Out[4]:

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	UV Index	Season	Visibility (km)	Location	Weather Type
0	14.0	73	9.5	82.0	partly cloudy	1010.82	2	Winter	3.5	inland	Rainy
1	39.0	96	8.5	71.0	partly cloudy	1011.43	7	Spring	10.0	inland	Cloudy
2	30.0	64	7.0	16.0	clear	1018.72	5	Spring	5.5	mountain	Sunny
3	38.0	83	1.5	82.0	clear	1026.25	7	Spring	1.0	coastal	Sunny
4	27.0	74	17.0	66.0	overcast	990.67	1	Winter	2.5	mountain	Rainy

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Temperature             0
Humidity                0
Wind Speed              0
Precipitation (%)       0
Cloud Cover             0
Atmospheric Pressure    0
UV Index                0
Season                  0
Visibility (km)         0
Location                0
Weather Type            0
dtype: int64
```

In [6]:
```
df.duplicated().sum()
```

Out[6]:
```
0
```

In [7]:
```
df = df.drop_duplicates()
```

In [8]:
```
df.shape
```

Out[8]:
```
(13200, 11)
```

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
0
```

In [10]:
```
df['Weather Type'].value_counts()
```

Out[10]:
```
Rainy     3300
Cloudy    3300
Sunny     3300
Snowy     3300
Name: Weather Type, dtype: int64
```

In [11]:
```
numeric_columns = df._get_numeric_data().columns
numeric_columns
```

Out[11]:
```
Index(['Temperature', 'Humidity', 'Wind Speed', 'Precipitation (%)',
       'Atmospheric Pressure', 'UV Index', 'Visibility (km)'],
      dtype='object')
```

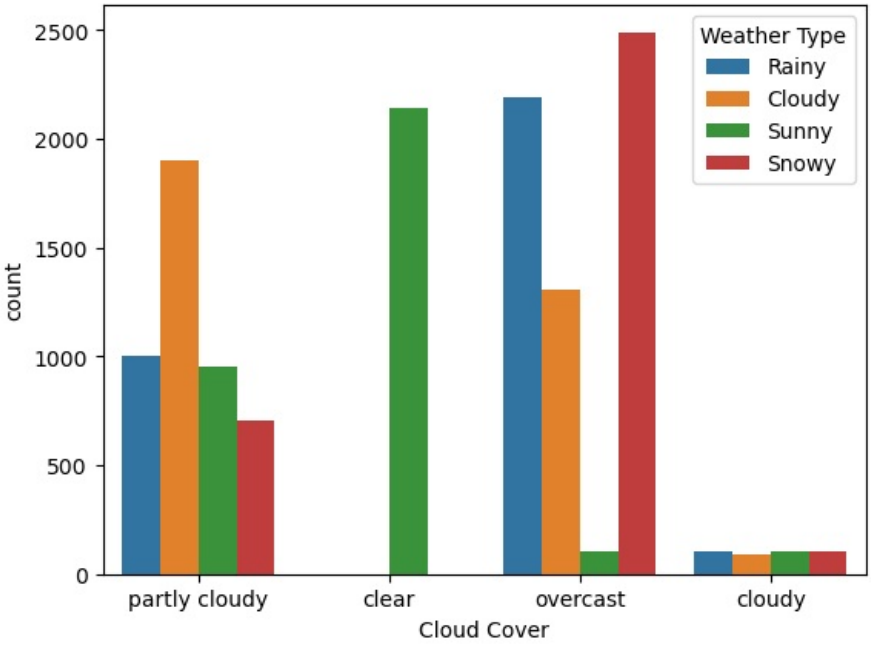
In [12]:
```
categorical_columns = df.drop(numeric_columns,axis=1).columns
categorical_columns[0]
```

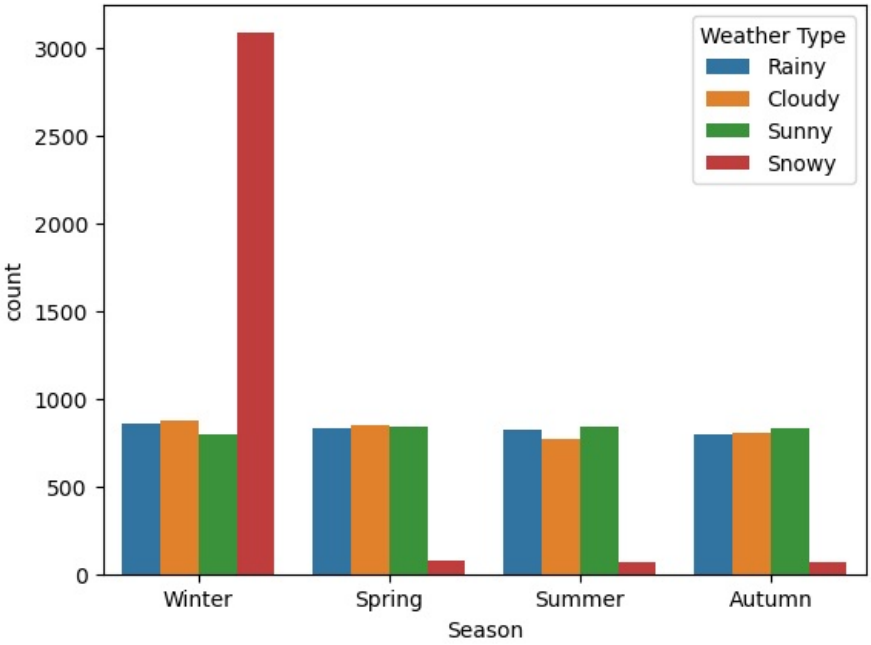
Out[12]:
```
'Cloud Cover'
```

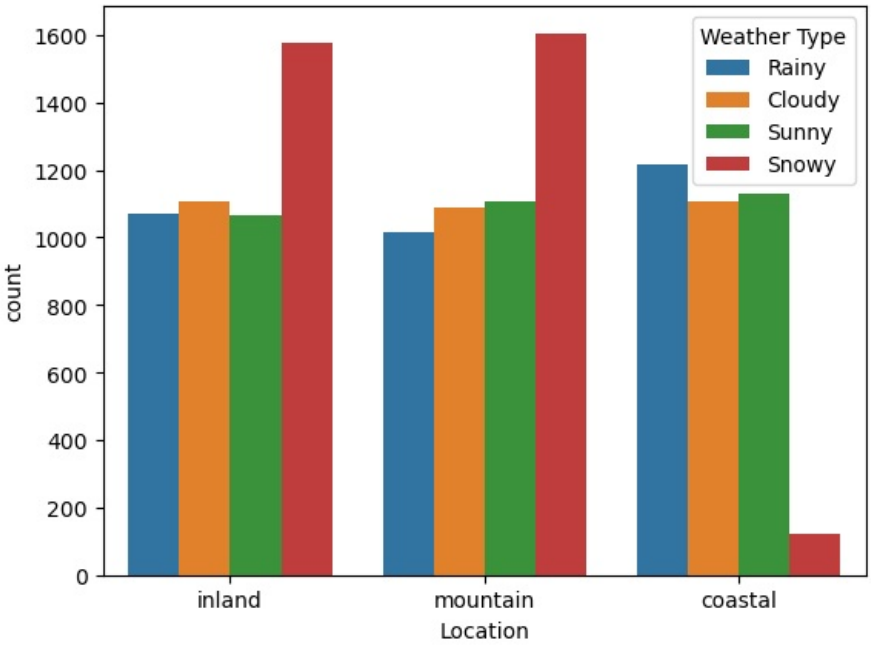
In [13]:
```
numeric_columns = df._get_numeric_data().columns
```

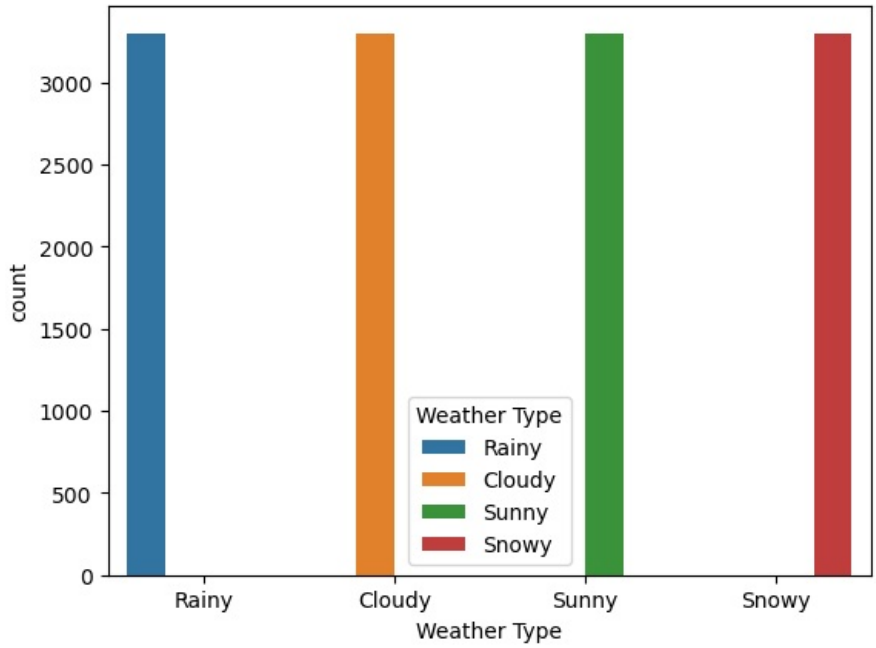
```
categorical_columns = df.drop(numeric_columns,axis=1).columns

for i in categorical_columns:
    sns.countplot(x=i,hue='Weather Type',data=df)
    plt.show()
```

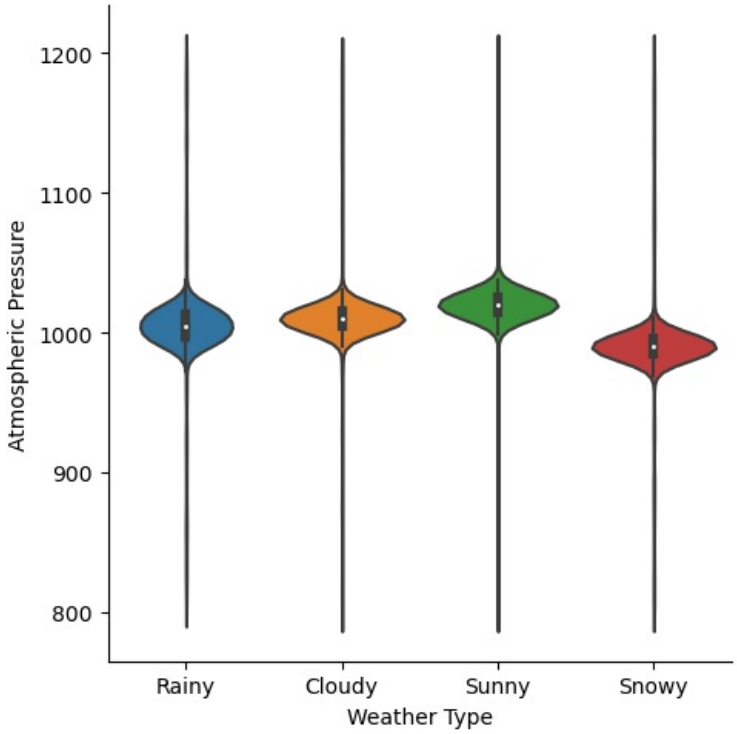




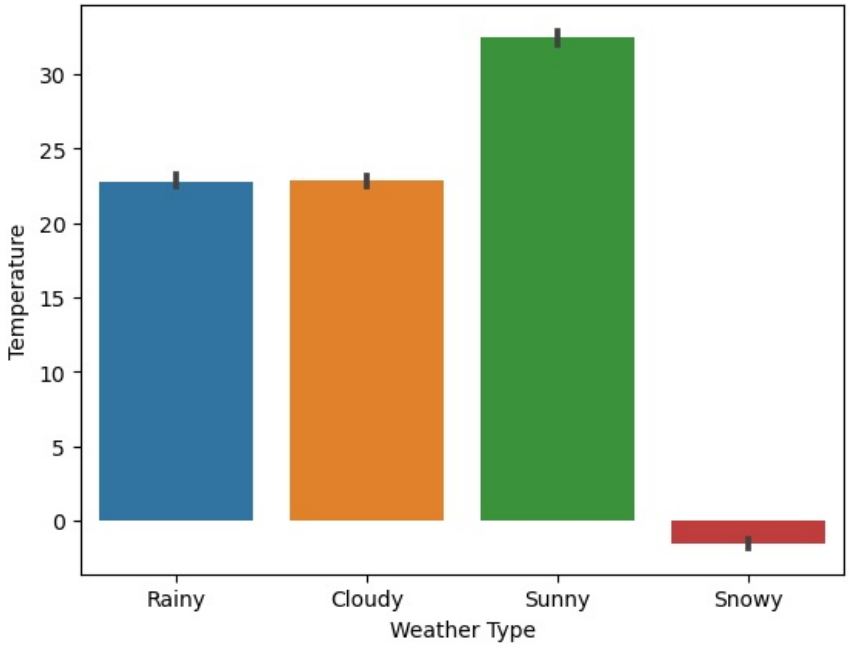




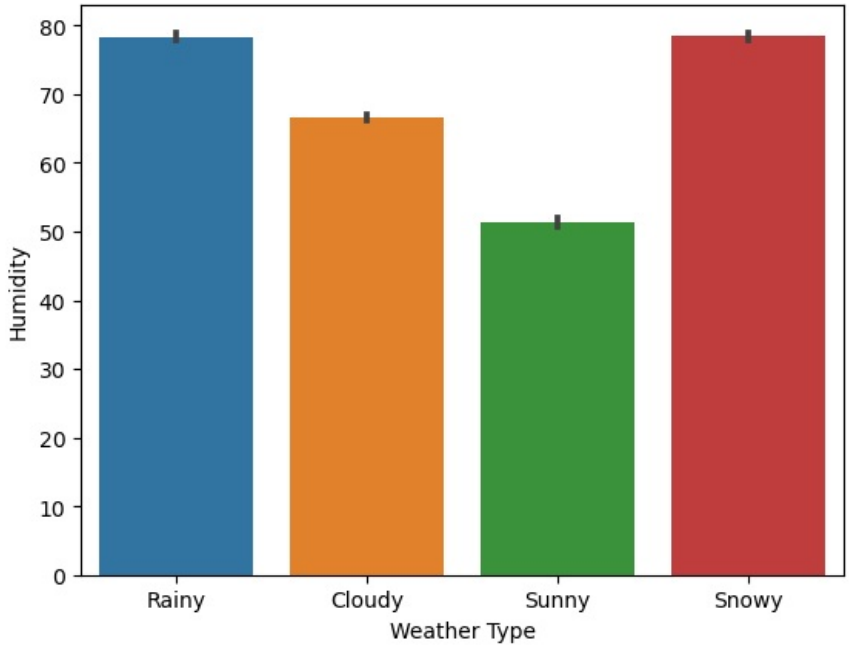
In [14]:
```
sns.catplot(x='Weather Type',y='Atmospheric Pressure',kind='violin',data=df);
```



In [15]:
```
sns.barplot(y='Temperature',x='Weather Type',data=df)
plt.show()
```



In [16]:
```
sns.barplot(y='Humidity',x='Weather Type',data=df)
plt.show()
```



In [17]:
```
df.dtypes
```

Out[17]:
```
Temperature             float64
Humidity                  int64
Wind Speed              float64
Precipitation (%)       float64
Cloud Cover              object
Atmospheric Pressure    float64
UV Index                  int64
Season                   object
Visibility (km)         float64
Location                 object
Weather Type             object
dtype: object
```

In [18]:
```
l = LabelEncoder()
```

In [19]:
```
cols = ['Cloud Cover','Location','Weather Type','Season']
for column in cols:
    df[column] = l.fit_transform(df[column])
```

In [20]:
```
x = df.drop('Weather Type',axis = 1)
y = df['Weather Type']
```

In [21]:
```
print(x.shape)
print(y.shape)
print(type(x))
print(type(y))
```

```
(13200, 10)
(13200,)
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

In [ ]:

```
```

In [23]:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 42)
```

In [24]:

```
m1 = DecisionTreeClassifier(random_state = 42)
```

In [25]:

```
m1.fit(x_train,y_train)
```

Out[25]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [26]:

```
preds = m1.predict(x_test)
```

In [27]:

```
def evaluate_metrics(yt, yp):
    results = {}
    results['accuracy'] = accuracy_score(yt, yp)

    # Determine if the classification is binary or multiclass
    unique_classes = set(yt)

    if len(unique_classes) == 2:
        average_method = 'binary'
    else:
        average_method = 'weighted'  # You can use 'micro', 'macro', or 'weighted'

    precision, recall, f_beta, _ = precision_recall_fscore_support(yt, yp, average=average_method)
    results['recall'] = recall
    results['precision'] = precision
    results['f1score'] = f_beta

    return results
```

In [52]:

```
evaluate_metrics(y_test, preds)
```

Out[52]:

```
{'accuracy': 0.9056818181818181,
 'recall': 0.9056818181818181,
 'precision': 0.9058387082362442,
 'f1score': 0.9056950158173532}
```

In [54]:

```
rs = 123
```

In [56]:

```
params_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 5]
}
```

In [58]:

```
model = DecisionTreeClassifier(random_state=rs)
```

In [60]:

```
grid_search = GridSearchCV(estimator = model,
                        param_grid = params_grid,
                        scoring='f1',
                        cv = 5, verbose = 1)
grid_search.fit(x_train, y_train.values.ravel())
best_params = grid_search.best_params_
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

In [62]:

```
best_params
```

Out[62]:

```
{'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 1}
```

In [68]:

```
m1 = DecisionTreeClassifier(criterion='gini', max_depth=5, min_samples_leaf=1)
```

In [70]:

```
m1.fit(x_train,y_train)
```

Out[70]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5)
```

In [78]:

```
y_pred = m1.predict(x_test)
```

In [80]:

```
cm = confusion_matrix(y_test,y_pred)
cf = classification_report(y_test,y_pred)
print(cm)
```

```python
print(cf)
```

```
[[585  35   9  22]
 [ 53 581   4   9]
 [ 29   9 652  11]
 [ 55  22   8 556]]
              precision    recall  f1-score   support

           0       0.81      0.90      0.85       651
           1       0.90      0.90      0.90       647
           2       0.97      0.93      0.95       701
           3       0.93      0.87      0.90       641

    accuracy                           0.90      2640
   macro avg       0.90      0.90      0.90      2640
weighted avg       0.90      0.90      0.90      2640
```

In [ ]: