Individual Assignment 2 Task 1

Name: Rohit Panda

UOW ID: 8943060

Before Running the code, ensure you have the kaggle.json file to upload into the Colab.

Download the kaggle.json from this link: Download Now

In [ ]:
```
!pip install -q kaggle
```

In [ ]:
```
# 1. Upload kaggle.json
from google.colab import files
files.upload()

# 2. Move it to the correct directory and set permissions
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d nikhil7280/weather-type-classification
!unzip -o weather-type-classification.zip

import pandas as pd

df = pd.read_csv('weather_classification_data.csv')
print("Dataset loaded successfully!")
df.head()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
Dataset URL: https://www.kaggle.com/datasets/nikhil7280/weather-type-classif
ication
License(s): other
weather-type-classification.zip: Skipping, found more recently modified loca
l copy (use --force to force download)
Archive:  weather-type-classification.zip
  inflating: weather_classification_data.csv
Dataset loaded successfully!
```

Out[ ]:

	Temperature	Humidity	Wind Speed	Precipitation (%)	Cloud Cover	Atmospheric Pressure	UV Index
0	14.0	73	9.5	82.0	partly cloudy	1010.82	2
1	39.0	96	8.5	71.0	partly cloudy	1011.43	7
2	30.0	64	7.0	16.0	clear	1018.72	5
3	38.0	83	1.5	82.0	clear	1026.25	7
4	27.0	74	17.0	66.0	overcast	990.67	1

In [ ]:

```
# CSCI316 - Task 1: Naïve Bayes Weather Type Classification
# Individual Assignment 2 - 2025 Session 3 (SIM)

## 1. Import Required Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.naive_bayes import GaussianNB, MultinomialNB, CategoricalNB
from sklearn.metrics import classification_report, confusion_matrix, accurac
from sklearn.compose import ColumnTransformer
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
# 2. Load the real dataset from Kaggle
file_name = 'weather_classification_data.csv'  # Adjust if the actual file r
if os.path.exists(file_name):
    df = pd.read_csv(file_name)
    print("Dataset loaded successfully!")
else:
    raise FileNotFoundError("weather_classification_data.csv not found. Plea


    # Sample weather data
    df = pd.DataFrame({
        'Temperature': np.random.normal(25, 10, n_samples),
        'Humidity': np.random.normal(60, 20, n_samples),
        'Wind Speed': np.random.normal(15, 5, n_samples),
        'Precipitation': np.random.uniform(0, 100, n_samples),
        'Cloud Cover': np.random.choice(['Clear', 'Partly Cloudy', 'Overcast
        'Atmospheric Pressure': np.random.normal(1013, 20, n_samples),
        'UV Index': np.random.uniform(0, 11, n_samples),
        'Season': np.random.choice(['Spring', 'Summer', 'Autumn', 'Winter'],
        'Visibility': np.random.uniform(1, 25, n_samples),
        'Location': np.random.choice(['Coastal', 'Inland', 'Mountain'], n_sa
        'Weather Type': np.random.choice(['Sunny', 'Rainy', 'Cloudy', 'Snowy
    })
    print("Sample dataset created for demonstration")
```

```python
print(f"Dataset shape: {df.shape}")
print(f"Dataset columns: {df.columns.tolist()}")
```

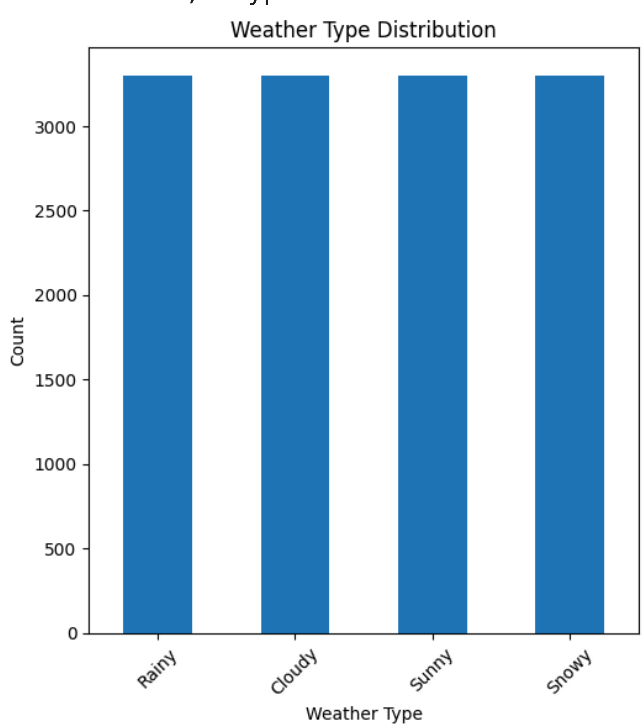
```
Dataset loaded successfully!
Dataset shape: (13200, 11)
Dataset columns: ['Temperature', 'Humidity', 'Wind Speed', 'Precipitation
(%)', 'Cloud Cover', 'Atmospheric Pressure', 'UV Index', 'Season', 'Visibili
ty (km)', 'Location', 'Weather Type']
```

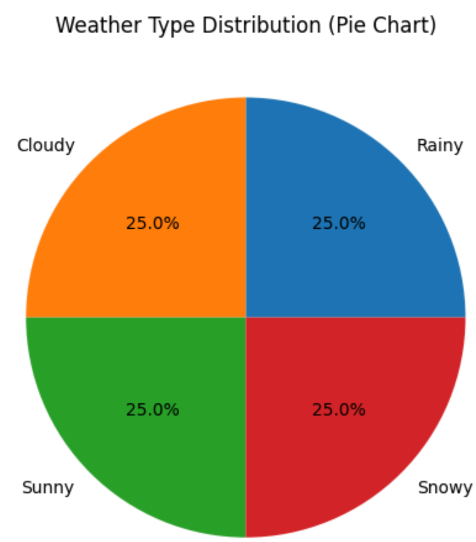
In [ ]:

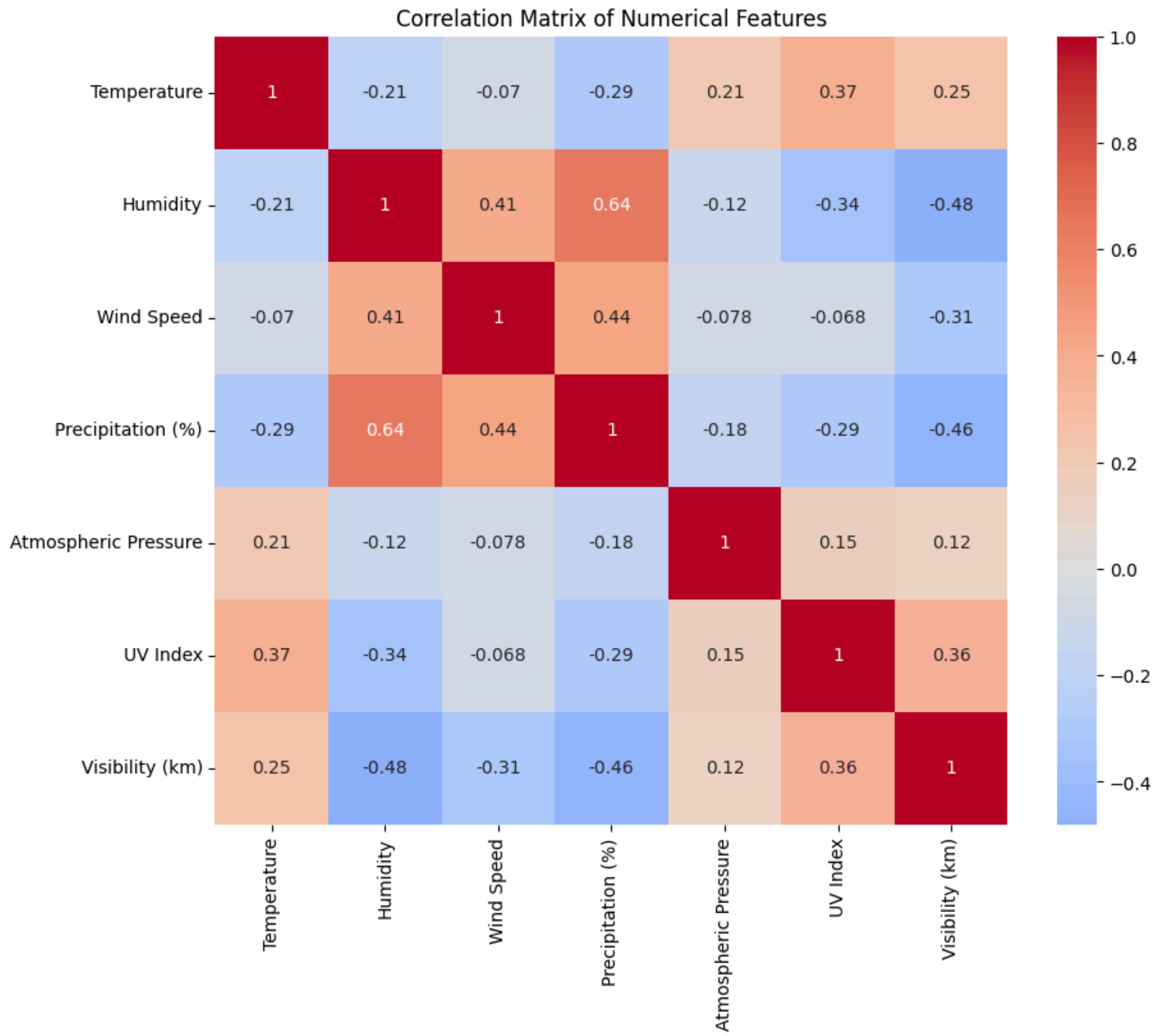
```python
## 3. Data Exploration and Visualization

# Display basic information about the dataset
print("\n=== Dataset Information ===")
print(df.info())
print("\n=== First 5 rows ===")
print(df.head())

# Check for missing values
print("\n=== Missing Values ===")
print(df.isnull().sum())

# Statistical summary
print("\n=== Statistical Summary ===")
print(df.describe())

# Target variable distribution
print("\n=== Weather Type Distribution ===")
print(df['Weather Type'].value_counts())

# Visualize target variable distribution
plt.figure(figsize=(10, 6))
plt.subplot(1, 2, 1)
df['Weather Type'].value_counts().plot(kind='bar')
plt.title('Weather Type Distribution')
plt.xlabel('Weather Type')
plt.ylabel('Count')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
df['Weather Type'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title('Weather Type Distribution (Pie Chart)')
plt.ylabel('')

plt.tight_layout()
plt.show()

# Correlation matrix for numerical features
numerical_features = df.select_dtypes(include=[np.number]).columns
if len(numerical_features) > 1:
    plt.figure(figsize=(10, 8))
    correlation_matrix = df[numerical_features].corr()
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
    plt.title('Correlation Matrix of Numerical Features')
    plt.show()
```

```
=== Dataset Information ===
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13200 entries, 0 to 13199
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Temperature           13200 non-null  float64
 1   Humidity              13200 non-null  int64  
 2   Wind Speed            13200 non-null  float64
 3   Precipitation (%)     13200 non-null  float64
 4   Cloud Cover           13200 non-null  object 
 5   Atmospheric Pressure  13200 non-null  float64
 6   UV Index              13200 non-null  int64  
 7   Season                13200 non-null  object 
 8   Visibility (km)       13200 non-null  float64
 9   Location              13200 non-null  object 
 10  Weather Type          13200 non-null  object 
dtypes: float64(5), int64(2), object(4)
memory usage: 1.1+ MB
None

=== First 5 rows ===
   Temperature  Humidity  Wind Speed  Precipitation (%)    Cloud Cover  \
0         14.0        73         9.5               82.0  partly cloudy   
1         39.0        96         8.5               71.0  partly cloudy   
2         30.0        64         7.0               16.0          clear   
3         38.0        83         1.5               82.0          clear   
4         27.0        74        17.0               66.0       overcast   

   Atmospheric Pressure  UV Index  Season  Visibility (km)  Location  \
0               1010.82         2  Winter              3.5    inland   
1               1011.43         7  Spring             10.0    inland   
2               1018.72         5  Spring              5.5  mountain   
3               1026.25         7  Spring              1.0   coastal   
4                990.67         1  Winter              2.5  mountain   

  Weather Type  
0        Rainy  
1       Cloudy  
2        Sunny  
3        Sunny  
4        Rainy  

=== Missing Values ===
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
```

```
=== Statistical Summary ===
        Temperature      Humidity    Wind Speed  Precipitation (%)  \
count  13200.000000  13200.000000  13200.000000       13200.000000
mean      19.127576     68.710833      9.832197          53.644394
std       17.386327     20.194248      6.908704          31.946541
min      -25.000000     20.000000      0.000000           0.000000
25%        4.000000     57.000000      5.000000          19.000000
50%       21.000000     70.000000      9.000000          58.000000
75%       31.000000     84.000000     13.500000          82.000000
max      109.000000    109.000000     48.500000         109.000000

       Atmospheric Pressure      UV Index  Visibility (km)
count          13200.000000  13200.000000     13200.000000
mean            1005.827896      4.005758         5.462917
std               37.199589      3.856600         3.371499
min              800.120000      0.000000         0.000000
25%              994.800000      1.000000         3.000000
50%             1007.650000      3.000000         5.000000
75%             1016.772500      7.000000         7.500000
max             1199.210000     14.000000        20.000000

=== Weather Type Distribution ===
Weather Type
Rainy     3300
Cloudy    3300
Sunny     3300
Snowy     3300
Name: count, dtype: int64
```







In [ ]:

```
## 4. Data Preprocessing

# Identify categorical and numerical features
categorical_features = df.select_dtypes(include=['object']).columns.tolist()
numerical_features = df.select_dtypes(include=[np.number]).columns.tolist()

# Remove target variable from features
if 'Weather Type' in categorical_features:
    categorical_features.remove('Weather Type')
if 'Weather Type' in numerical_features:
    numerical_features.remove('Weather Type')

print(f"Categorical features: {categorical_features}")
print(f"Numerical features: {numerical_features}")

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

# Encode target variable
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

```python
print(f"\nTarget classes: {label_encoder.classes_}")
print(f"Encoded target shape: {y_encoded.shape}")
```

```
Categorical features: ['Cloud Cover', 'Season', 'Location']
Numerical features: ['Temperature', 'Humidity', 'Wind Speed', 'Precipitation
(%)', 'Atmospheric Pressure', 'UV Index', 'Visibility (km)']

Target classes: ['Cloudy' 'Rainy' 'Snowy' 'Sunny']
Encoded target shape: (13200,)
```

In [ ]:
```python
## 5. Feature Engineering - One-Hot Encoding for Categorical Features

# Create preprocessing pipeline
if categorical_features:
    # One-hot encode categorical features
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', 'passthrough', numerical_features),
            ('cat', OneHotEncoder(drop='first', sparse_output=False), catego
        ]
    )

    # Fit and transform the features
    X_processed = preprocessor.fit_transform(X)

    # Get feature names after transformation
    feature_names = numerical_features.copy()
    if hasattr(preprocessor.named_transformers_['cat'], 'get_feature_names_c
        cat_feature_names = preprocessor.named_transformers_['cat'].get_feat
        feature_names.extend(cat_feature_names)

    print(f"Features after preprocessing: {len(feature_names)}")
    print(f"Processed data shape: {X_processed.shape}")
else:
    X_processed = X.values
    feature_names = numerical_features
```

```
Features after preprocessing: 15
Processed data shape: (13200, 15)
```

In [ ]:
```python
## 6. Stratified Train-Test Split

# Use stratified sampling to maintain class distribution
stratified_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_
train_idx, test_idx = next(stratified_split.split(X_processed, y_encoded))

X_train = X_processed[train_idx]
X_test = X_processed[test_idx]
y_train = y_encoded[train_idx]
y_test = y_encoded[test_idx]

print(f"\nTraining set shape: {X_train.shape}")
print(f"Test set shape: {X_test.shape}")

# Check class distribution in train and test sets
print(f"\nTraining set class distribution:")
unique_train, counts_train = np.unique(y_train, return_counts=True)
```

```python
for i, count in enumerate(counts_train):
    print(f"  {label_encoder.classes_[unique_train[i]]}: {count} ({count/ler
    
print(f"\nTest set class distribution:")
unique_test, counts_test = np.unique(y_test, return_counts=True)
for i, count in enumerate(counts_test):
    print(f"  {label_encoder.classes_[unique_test[i]]}: {count} ({count/len(
```

```
Training set shape: (10560, 15)
Test set shape: (2640, 15)

Training set class distribution:
  Cloudy: 2640 (25.0%)
  Rainy: 2640 (25.0%)
  Snowy: 2640 (25.0%)
  Sunny: 2640 (25.0%)

Test set class distribution:
  Cloudy: 660 (25.0%)
  Rainy: 660 (25.0%)
  Snowy: 660 (25.0%)
  Sunny: 660 (25.0%)
```

In [ ]:

```python
## 7. Naïve Bayes Model Implementation

# Try different Naïve Bayes variants
models = {
    'Gaussian Naïve Bayes': GaussianNB(),
    'Multinomial Naïve Bayes': MultinomialNB(),
}

results = {}

for name, model in models.items():
    print(f"\n=== {name} ===")
    
    try:
        # Train the model
        model.fit(X_train, y_train)
        
        # Make predictions
        y_train_pred = model.predict(X_train)
        y_test_pred = model.predict(X_test)
        
        # Calculate accuracies
        train_accuracy = accuracy_score(y_train, y_train_pred)
        test_accuracy = accuracy_score(y_test, y_test_pred)
        
        print(f"Training Accuracy: {train_accuracy:.4f}")
        print(f"Test Accuracy: {test_accuracy:.4f}")
        
        # Store results
        results[name] = {
            'model': model,
            'train_accuracy': train_accuracy,
            'test_accuracy': test_accuracy,
```

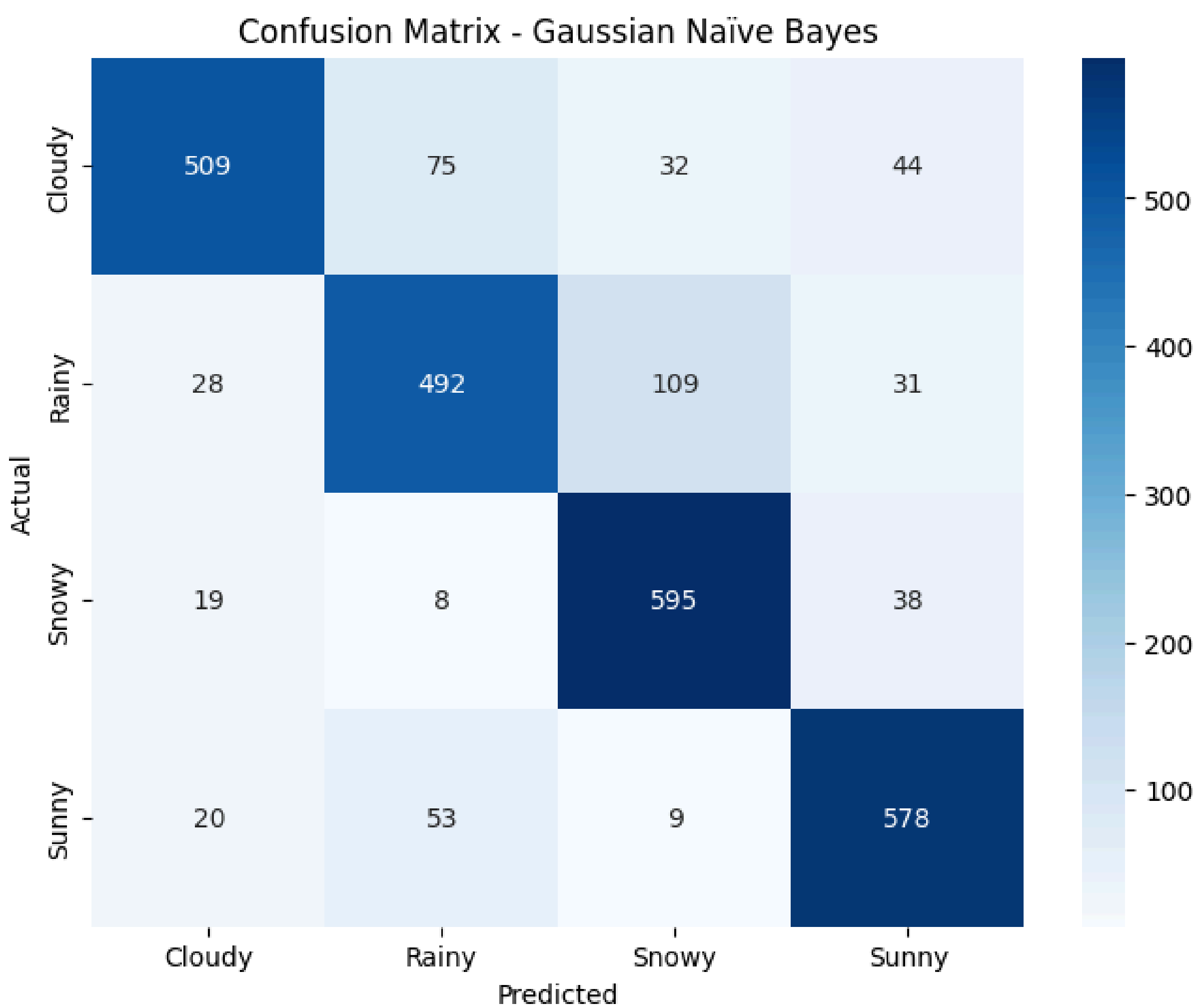
```
            'train_predictions': y_train_pred,
            'test_predictions': y_test_pred
        }

    except Exception as e:
        print(f"Error with {name}: {str(e)}")
```

```
=== Gaussian Naïve Bayes ===
Training Accuracy: 0.8290
Test Accuracy: 0.8235

=== Multinomial Naïve Bayes ===
Error with Multinomial Naïve Bayes: Negative values in data passed to Multin
omialNB (input X).
```

In [ ]:

```
## 8. Model Evaluation and Results

# Select the best performing model
best_model_name = max(results.keys(), key=lambda x: results[x]['test_accurac
best_model = results[best_model_name]['model']

print(f"\n=== Best Model: {best_model_name} ===")
print(f"Training Accuracy: {results[best_model_name]['train_accuracy']:.4f}"
print(f"Test Accuracy: {results[best_model_name]['test_accuracy']:.4f}")

# Detailed classification report
print(f"\n=== Classification Report for {best_model_name} ===")
y_test_pred_best = results[best_model_name]['test_predictions']
report = classification_report(y_test, y_test_pred_best,
                               target_names=label_encoder.classes_,
                               output_dict=True)

print(classification_report(y_test, y_test_pred_best,
                            target_names=label_encoder.classes_))

# Confusion Matrix
cm = confusion_matrix(y_test, y_test_pred_best)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)
plt.title(f'Confusion Matrix - {best_model_name}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
=== Best Model: Gaussian Naïve Bayes ===
Training Accuracy: 0.8290
Test Accuracy: 0.8235

=== Classification Report for Gaussian Naïve Bayes ===
              precision    recall  f1-score   support

      Cloudy       0.88      0.77      0.82       660
       Rainy       0.78      0.75      0.76       660
       Snowy       0.80      0.90      0.85       660
       Sunny       0.84      0.88      0.86       660

    accuracy                           0.82      2640
   macro avg       0.83      0.82      0.82      2640
weighted avg       0.83      0.82      0.82      2640
```



In [ ]:
```python
## 9. Feature Importance Analysis (for Gaussian NB)

if best_model_name == 'Gaussian Naïve Bayes':
    # For Gaussian NB, we can analyze feature importance based on variance
    print(f"\n=== Feature Analysis for Gaussian Naïve Bayes ===")

    # Get class-wise feature statistics
    n_classes = len(label_encoder.classes_)
    n_features = X_train.shape[1]
```

```
print(f"Number of classes: {n_classes}")
print(f"Number of features: {n_features}")

# Display theta (mean) and sigma (variance) for each class
if hasattr(best_model, 'theta_') and hasattr(best_model, 'var_'):
    print("\nClass-wise feature means (theta):")
    for i, class_name in enumerate(label_encoder.classes_):
        print(f"\n{class_name}:")
        if len(feature_names) == len(best_model.theta_[i]):
            for j, feature in enumerate(feature_names[:min(10, len(featu
                print(f"  {feature}: {best_model.theta_[i][j]:.4f}")
        else:
            print(f"  Mean values: {best_model.theta_[i][:5]}...")  # Sh
```

```
=== Feature Analysis for Gaussian Naïve Bayes ===
Number of classes: 4
Number of features: 15

Class-wise feature means (theta):

Cloudy:
  Temperature: 22.7985
  Humidity: 66.3155
  Wind Speed: 8.6400
  Precipitation (%): 40.3705
  Atmospheric Pressure: 1010.0687
  UV Index: 3.5742
  Visibility (km): 7.1225
  Cloud Cover_cloudy: 0.0280
  Cloud Cover_overcast: 0.3924
  Cloud Cover_partly cloudy: 0.5795

Rainy:
  Temperature: 22.7595
  Humidity: 78.2015
  Wind Speed: 13.7523
  Precipitation (%): 74.7394
  Atmospheric Pressure: 1003.5143
  UV Index: 2.6864
  Visibility (km): 3.6337
  Cloud Cover_cloudy: 0.0307
  Cloud Cover_overcast: 0.6583
  Cloud Cover_partly cloudy: 0.3110

Snowy:
  Temperature: -1.5864
  Humidity: 78.5034
  Wind Speed: 11.0536
  Precipitation (%): 74.7652
  Atmospheric Pressure: 991.2356
  UV Index: 1.9242
  Visibility (km): 3.5602
  Cloud Cover_cloudy: 0.0318
  Cloud Cover_overcast: 0.7663
  Cloud Cover_partly cloudy: 0.2019

Sunny:
  Temperature: 32.3511
  Humidity: 51.2159
  Wind Speed: 6.0852
  Precipitation (%): 24.5466
  Atmospheric Pressure: 1017.8063
  UV Index: 7.8193
  Visibility (km): 7.5525
  Cloud Cover_cloudy: 0.0330
  Cloud Cover_overcast: 0.0299
  Cloud Cover_partly cloudy: 0.2917
```

In [ ]:

```
## 10. Model Comparison Visualization
```

```python
if len(results) > 1:
    # Compare model performances
    model_names = list(results.keys())
    train_accuracies = [results[name]['train_accuracy'] for name in model_na
    test_accuracies = [results[name]['test_accuracy'] for name in model_name

    x = np.arange(len(model_names))
    width = 0.35

    plt.figure(figsize=(10, 6))
    plt.bar(x - width/2, train_accuracies, width, label='Training Accuracy',
    plt.bar(x + width/2, test_accuracies, width, label='Test Accuracy', alph

    plt.xlabel('Models')
    plt.ylabel('Accuracy')
    plt.title('Model Performance Comparison')
    plt.xticks(x, model_names, rotation=45)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

In [ ]:
```python
## 11. Conclusions and Summary

print("\n=== SUMMARY AND CONCLUSIONS ===")
print(f"1. Dataset contains {df.shape[0]} samples with {df.shape[1]} feature
print(f"2. Target variable has {len(label_encoder.classes_)} classes: {', '.
print(f"3. Used stratified sampling: {len(X_train)} training samples, {len(X
print(f"4. Applied one-hot encoding to categorical features")
print(f"5. Best performing model: {best_model_name}")
print(f"6. Final test accuracy: {results[best_model_name]['test_accuracy']:.

# Performance metrics summary
print(f"\n=== PERFORMANCE METRICS ===")
for metric in ['precision', 'recall', 'f1-score']:
    macro_avg = report['macro avg'][metric]
    weighted_avg = report['weighted avg'][metric]
    print(f"{metric.title()}: Macro avg = {macro_avg:.4f}, Weighted avg = {w

print(f"\n=== IMPLEMENTATION NOTES ===")
print("1. Stratified sampling ensures balanced class distribution in train/t
print("2. One-hot encoding transforms categorical features to numerical form
print("3. Gaussian Naïve Bayes works well with continuous features")
print("4. Model assumes feature independence (Naïve Bayes assumption)")
print("5. Performance can be improved with feature selection and hyperparame

print("\n=== TASK 1 COMPLETED SUCCESSFULLY ===")
```

```
=== SUMMARY AND CONCLUSIONS ===
1. Dataset contains 13200 samples with 11 features
2. Target variable has 4 classes: Cloudy, Rainy, Snowy, Sunny
3. Used stratified sampling: 10560 training samples, 2640 test samples
4. Applied one-hot encoding to categorical features
5. Best performing model: Gaussian Naïve Bayes
6. Final test accuracy: 0.8235

=== PERFORMANCE METRICS ===
Precision: Macro avg = 0.8256, Weighted avg = 0.8256
Recall: Macro avg = 0.8235, Weighted avg = 0.8235
F1-Score: Macro avg = 0.8226, Weighted avg = 0.8226

=== IMPLEMENTATION NOTES ===
1. Stratified sampling ensures balanced class distribution in train/test spl
its
2. One-hot encoding transforms categorical features to numerical format
3. Gaussian Naïve Bayes works well with continuous features
4. Model assumes feature independence (Naïve Bayes assumption)
5. Performance can be improved with feature selection and hyperparameter tun
ing

=== TASK 1 COMPLETED SUCCESSFULLY ===
```