project

August 9, 2024

1 Prediction of Future Weather Type

1.1 Summary:

1.1.1 Research Questions:

1. What is the interaction between multiple meteorological factors?
   Answer: The interaction between multiple meteorological factors, such as temperature, humidity, wind speed, atmospheric pressure, and cloud cover, determines the formation of different weather conditions. For instance, high humidity combined with high precipitation, while high temperature will lead high UV index. These factors are interdependent, meaning changes in one can influence the behavior of others, shaping the overall weather patterns we experience.
2. What are the key meteorological factors that most significantly affect the prediction of weather type? My hypothesis is that location and season are important, is that right?
   Answer: There are so many factors that influence the weather type like temperature, UV index, humidity, air pressure, clouds, etc. The data set contains some of these factors and in some way, I find the most important features that influence the temperature. Also, the UV index, visibility, and precipitation are important. Unfortunately, my assumption is not true. These two factors are not important.
3. What is the main type of weather if the temperature is over 25 and the UV index is over 5? I assume that the dominant weather type should be Sunny.
   Answer: Sunny is the main type of weather based on high temperature and UV index. The assumption is correct. If the future weather has a relatively high temperature(over 25 or over 30) and a high UV index(over 5), we can see that almost is sunny day.

1.2 Challenge Goals

1. **Advanced Machine Learning**: I will use various models to gain insights about the data or make predictions about the future. However, different models will generate different perspectives. In order to select the fittest model for our analysis, we need to compare those models with each other until we find the best one.
2. **New Library**: I perform some new libraries like Scipy or PyTorch to do this project.
3. **Statistical Hypothesis Testing**: I will use the Mann White U test to test my hypothesis

1.3 Collaboration and Conduct

Students are expected to follow Washington state law on the Student Conduct Code for the University of Washington. In this course, students must:

- Indicate on your submission any assistance received, including materials distributed in this course.
- Not receive, generate, or otherwise acquire any substantial portion or walkthrough to an assessment.
- Not aid, assist, attempt, or tolerate prohibited academic conduct in others.

Update the following code cell to include your name and list your sources. If you used any kind of computer technology to help prepare your assessment submission, include the queries and/or prompts. Submitted work that is not consistent with sources may be subject to the student conduct process.

[78]:
```
your_name = "Jiatong Gao"
sources = [
    "The lecture notes in the quarter",
    "Some stat knowledge I learnt before",
    "https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.
 ↪LogisticRegression.html",
    "https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.
 ↪GradientBoostingRegressor.html",
    "https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.
 ↪RandomForestClassifier.html",
    "https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.
 ↪KNeighborsRegressor.html",
    "https://scikit-learn.org/stable/auto_examples/ensemble/
 ↪plot_forest_importances.html",
    "https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html",
    "https://pytorch.org/docs/stable/tensors.html",
    "https://pytorch.org/docs/stable/generated/torch.no_grad.html#torch.
 ↪no_grad",
    "https://pytorch.org/docs/stable/generated/torch.nn.Linear.html",
    "https://seaborn.pydata.org/generated/seaborn.heatmap.html",
    "https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.
 ↪mannwhitneyu.html",
    "https://en.wikipedia.org/wiki/Mann%E2%80%93Whitney_U_test"


]

assert your_name != "", "your_name cannot be empty"
assert ... not in sources, "sources should not include the placeholder ellipsis"
assert len(sources) >= 6, "must include at least 6 sources, inclusive of␣
 ↪lectures and sections"
```

1.4 Data Setting

To access the dataset, please visit the website https://www.kaggle.com/datasets/nikhil7280/weather-type-classification and click the download button for the CSV file The dataset contains 11 columns, 4 of them (cloud cover, season, location, weather type) are categorical and the rest of 7 are con-

tinuous, which include temperature, humidity, UV index, wind speed, precipitation, Atmospheric Pressure, and visibility. This dataset is synthetically generated to mimic weather data for classification tasks. It includes various weather-related features and categorizes the weather into four types: Rainy, Sunny, Cloudy, and Snowy. This dataset is designed for practicing classification algorithms, data preprocessing, and outlier detection methods.

1.5 Method

1.5.1 Research Question 1

Firstly, I deal with the data, encoding all the categorical data into the numerical data. Then I use the *corr* function to calculate the correlation between the factors. The correlation ranges from -1 to 1. The value more closer to 1, these two features have a stronger positive correlation. It means if one increases, then the other one also increases. If the correlation is close to -1, it means these two features are negatively correlated, if one increases, then the other will decrease. Moreover, if the correlation is close to 0, it means that these two features don't have any relationship. I made a correlation matrix plot to visualize it, it is easier to analyze from the plot.

1.5.2 Research Question 2

Firstly, I select four models that I am familiar with, these are (1) Logistic Regression (2) Decision Tree (3) Random Forest, and (4) K-nearest neighborhood(KNN). In the beginning, I need to deal with the categorical variables in the dataset, I use the LabelEncoder() in scikit-learn to encode the categorical variables to integers from 0 to 3. And then divide the dataset into features(X) and target(y). y is the column "Weather Type", and x is the other column. I split the data into the training set and testing set, the proportion is 8:2 (training: testing). After that, I fit the model using the training data. Finally, I use the functions from scikit-learn to make the report and confusion matrix about each mode. Then analysis of the precision, recall, accuracy, and rate of the overall performance of each model.
Then, I select the best two performance models and use the *feature_importance* function to visualize the importance of each feature, and then take the mean value of the feature importance of these two different models. And then report the most important feature.
I also assume the location and the season are important features. So I do the Mann–Whitney U test to test my assumption. I set my hypothesis as below:
Null hypothesis H_0: *"Location and Season" have significantly higher feature importance compared to other features.*
Alternative hypothesis H_1 : *"Location and Season" do not have significantly higher feature importance*
If we can reject the Null Hypothesis, it means that we do not have sufficient evidence to say that "Location and Season" have a significant influence. If we cannot reject the Null Hypothesis, it means that there is sufficient evidence to say that "Location and Season" have a significant influence on predicting the weather type.

1.5.3 Research Question 3

In this question, I want to see the future weather type, so I build the neural network. The first step is to deal with data, I split the training set and the testing set with the proportion 8:2. And also, I want to use **PyTorch** to build the neural network, so I need to transfer all the data to the torch. I do this with the *torch.tensor()* function. . And then, I use StandardScaler() to rescale

the training set and make the prediction more accurate. The first layer of this simple network is a fully connected layer, it receives the input neurons the number of features of the training set, and outputs 128 features. For the second layer, the layer normalizes the output of the first fully connected layer, helping to stabilize and speed up the training process by keeping the mean of the output close to 0 and the standard deviation close to 1. Then, follow the same logic, build the second and the third fully connected layer and the second batch normalization layer. After the third fully connected layer, use the ReLU function as the activation function. This active function introduces nonlinearity to the model to help it learn more complex patterns. Finally, build up a Dropout layer, this layer randomly sets 30% of the input units to 0 during training to help prevent overfitting and ensure that the model is not overly dependent on any individual neuron. After defining the neural network, I use CrossEntropyLoss() as the loss function. It is good for the classification problem, And the most used optimizer, Adam optimizer is the optimizer of this problem. In the end, I trained the model 1500 epochs.
And then, I use the test set to test the model. While the model behaves not well, I will continue to adjust the neural network until it satisfies my requirement. Moreover, I select from the dataset that the UV index is over 5 and the temperature above 25 celsius degrees to verify my assumption, and I use some stat tools to test the accuracy of the model on a relatively small dataset.
Finally, I create a random data that UV index over 5 and temperature over 30, to make a prediction.

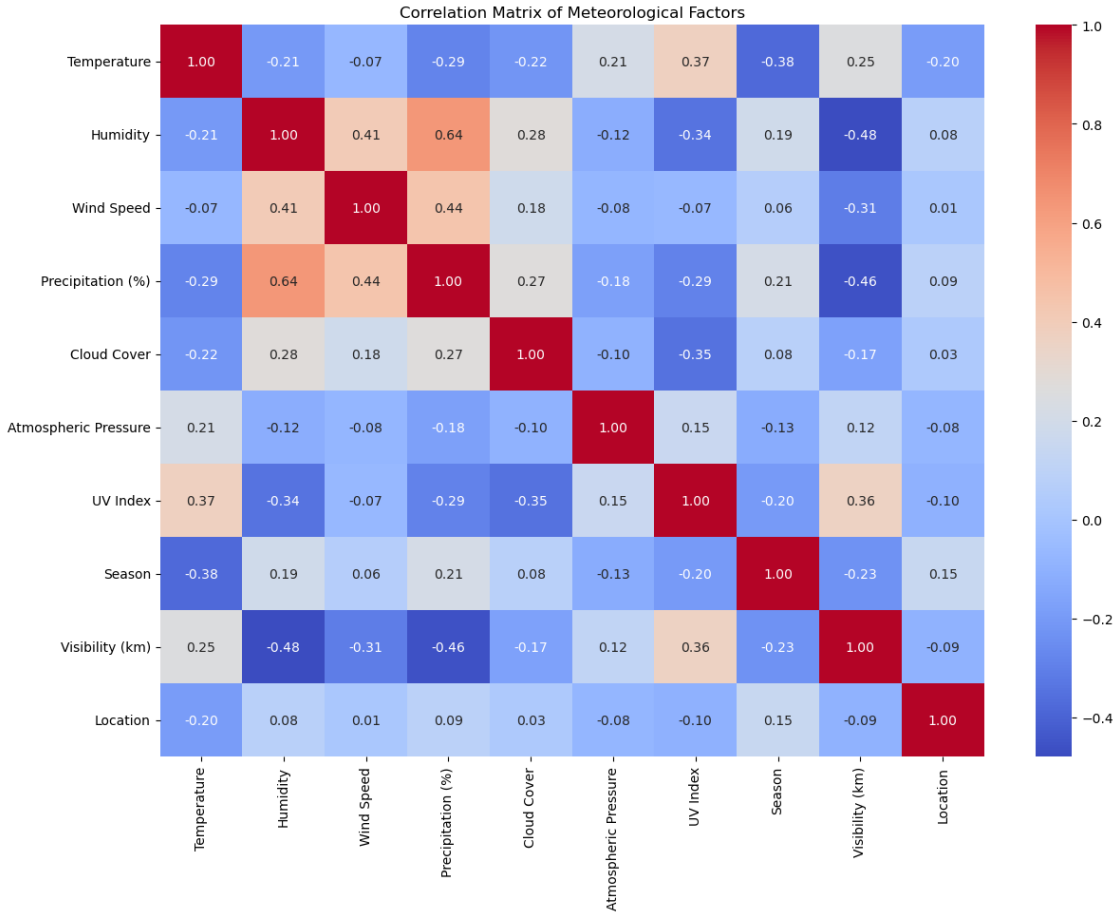
1.6 Results

1.6.1 Research Question 1

[57]:
```
#import the package
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
```

[58]:
```
# deal with the data
data = pd.read_csv('weather_classification_data.csv')
labels = {}
for column in ['Cloud Cover', 'Season', 'Location', 'Weather Type']:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    labels[column] = le
data = data.drop(columns=['Weather Type'])
```

[59]:
```
# calculate the correlation and plot
corr_matrix = data.corr()
# plot
plt.figure(figsize=(14, 10))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.xticks(rotation=90)
plt.title('Correlation Matrix of Meteorological Factors')
plt.show()
```



There are some key relationships in this matrix.

High Positive Correlations: Look for pairs of factors with correlation coefficients close to +1. For example, temperature and UV index are highly positively correlated, the correlation is 0.37 which suggests that as temperature increases, the UV index tends to increase as well. Also, the humidity and precipitation are highly positively correlated, the correlation between these two are 0.64

High Negative Correlations: Look for pairs with coefficients close to -1. For example, Wind speed and atmospheric pressure are highly negatively correlated, which suggests that higher wind speeds are associated with lower atmospheric pressures.

Weak or No Correlations: Factors with coefficients near 0 indicate little to no linear relationship. For example, there is a weak correlation between humidity and UV index, which suggests that changes in humidity do not significantly impact the UV index.

In conclusion, from this plot, we can see that if the color is closer to red, the higher positive correlations between these two features. If the color is closer to blue, the higher negative correlations between these two features. If the color is white or gray, it means there are weak or no correlations between these features.

1.6.2 Research Question 2

[60]:
```
# import the packages
import pandas as pd
import random
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from scipy.stats import mannwhitneyu
# set seed, import data
np.random.seed(42)
random.seed(42)
data = pd.read_csv('weather_classification_data.csv')
```

[61]:
```
# preprocess dataset, encoding the categorical data.
# All encoding to integer follow the alphabet
# Weather Type: 0: cloudy, 1 : rainy,  2 : snowy, 3: sunny
# Cloud Cover: 0: clear 1:cloudy 2: overcast 3: partly cloudy
# Season: 0: Autumn 1: Spring 2: Summer 3: Winter
# Location: 0: Coastal 1 : inland 2: mountain
labels = {}
for column in ['Cloud Cover', 'Season', 'Location', 'Weather Type']:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    labels[column] = le
# Split the data
# Feature X, target : weather (y)
X = data.drop(columns=['Weather Type'])
y = data['Weather Type']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

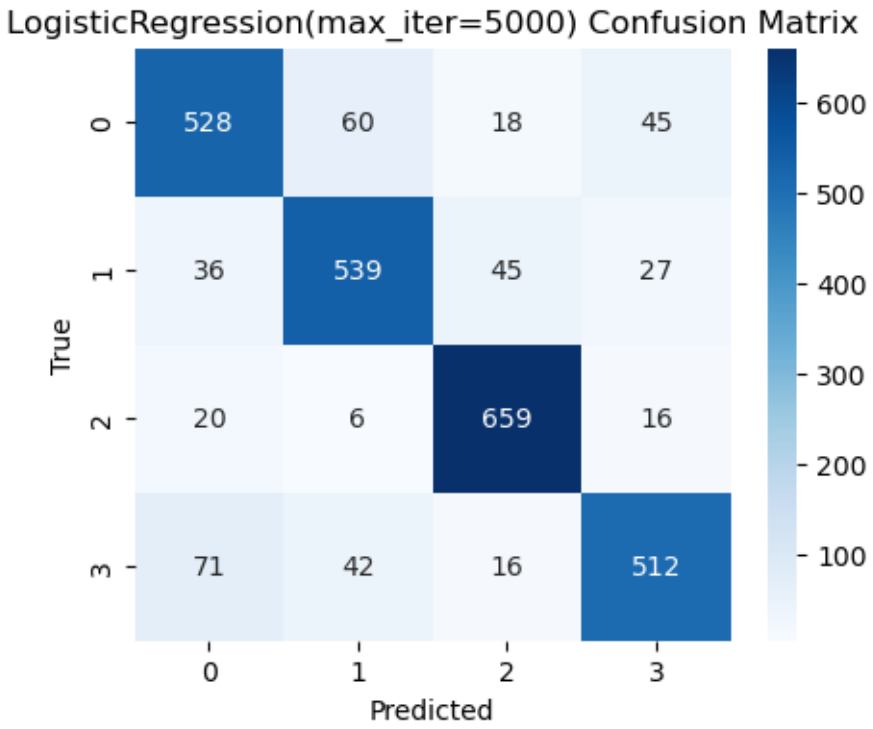
[62]:
```
# fit the model
log = LogisticRegression(max_iter=5000)
log.fit(X_train, y_train)
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
random_forest = RandomForestClassifier()
random_forest.fit(X_train, y_train)
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

[62]:
```
KNeighborsClassifier()
```

[63]:
```
# Define the plot function
def plot_confusion_matrix(model, X_test, y_test, class_names):
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)

    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.title(f'{model} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()
```

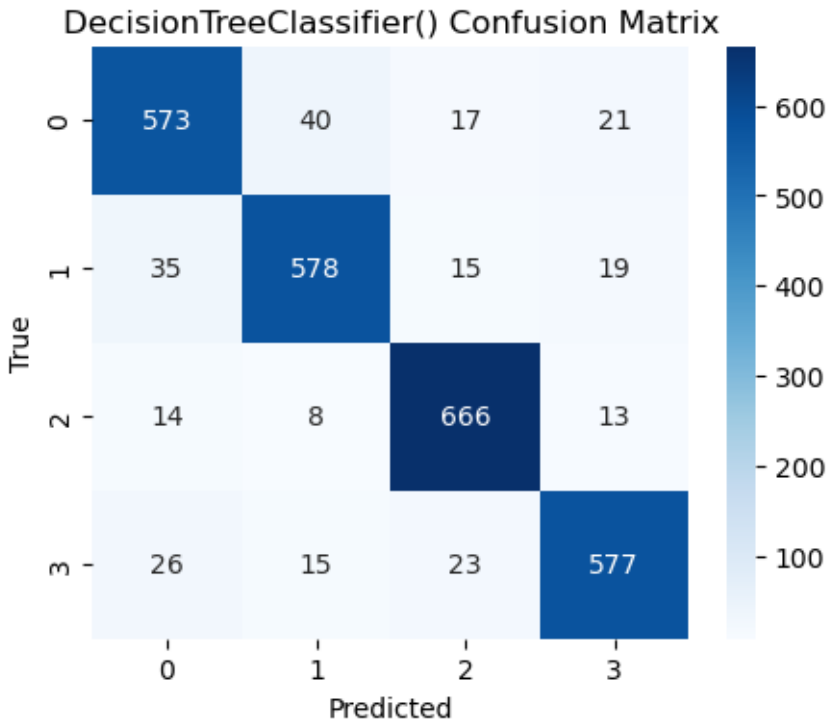
[64]:
```
# Show the result
for model in [log, decision_tree, random_forest, knn]:
    report = classification_report(y_test, model.predict(X_test))
    print(f"{model}:\n{report}")
    plot_confusion_matrix(model, X_test, y_test, labels)
```

```
LogisticRegression(max_iter=5000):
              precision    recall  f1-score   support

           0       0.81      0.81      0.81       651
           1       0.83      0.83      0.83       647
           2       0.89      0.94      0.92       701
           3       0.85      0.80      0.83       641

    accuracy                           0.85      2640
   macro avg       0.85      0.85      0.85      2640
weighted avg       0.85      0.85      0.85      2640
```



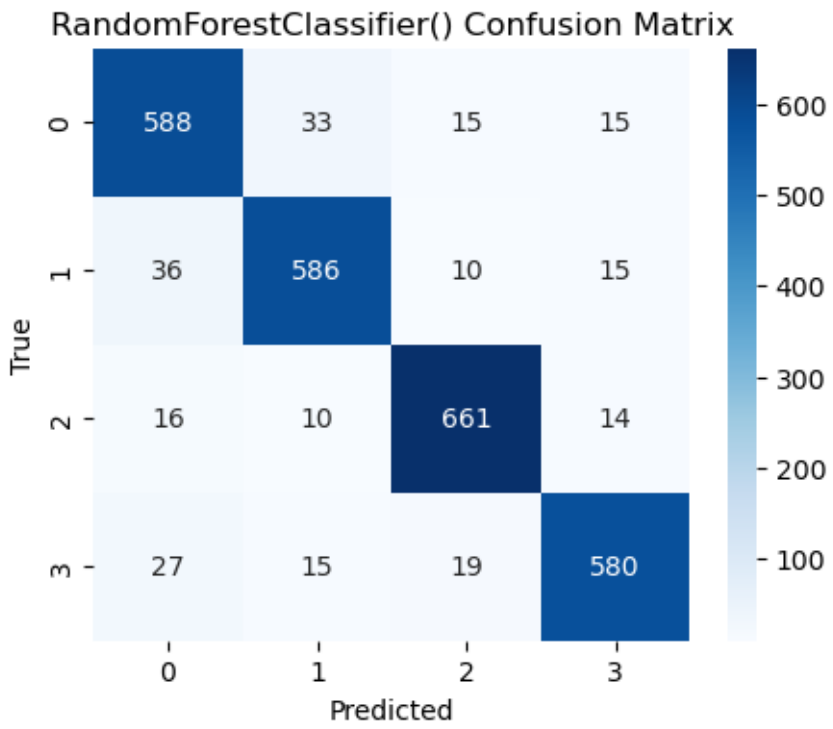
DecisionTreeClassifier():

	precision	recall	f1-score	support
0	0.88	0.88	0.88	651
1	0.90	0.89	0.90	647
2	0.92	0.95	0.94	701
3	0.92	0.90	0.91	641
accuracy			0.91	2640
macro avg	0.91	0.91	0.91	2640
weighted avg	0.91	0.91	0.91	2640



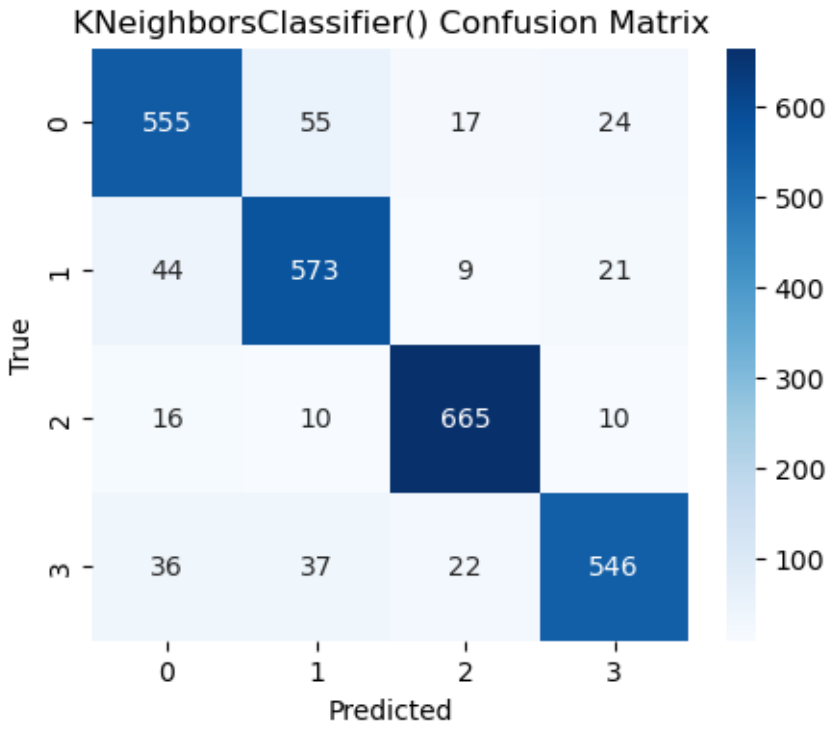
RandomForestClassifier():

	precision	recall	f1-score	support
0	0.88	0.90	0.89	651
1	0.91	0.91	0.91	647
2	0.94	0.94	0.94	701
3	0.93	0.90	0.92	641
accuracy			0.91	2640
macro avg	0.91	0.91	0.91	2640
weighted avg	0.92	0.91	0.91	2640



KNeighborsClassifier():

	precision	recall	f1-score	support
0	0.85	0.85	0.85	651
1	0.85	0.89	0.87	647
2	0.93	0.95	0.94	701
3	0.91	0.85	0.88	641
accuracy			0.89	2640
macro avg	0.89	0.88	0.88	2640
weighted avg	0.89	0.89	0.89	2640



From the report table and confusion matrix we can analyze four models:
(1) Logistic Regression: The model performs overall well because the accuracy is 0.85. The recall is around 80% for this model except for the Snowy Weather Type(Class 2). It means that the misclassification rate is around 20%, and for the weather prediction, this is a little bit high, but not so bad. Also, the precision is around 80%, which means that the model's judgment of "positive prediction" is not very credible. And F1 score is also around 80%, which means the combined performance of the model in terms of completeness and accuracy of detection is good. The noticeable data is that for the Snowy Weather Type prediction, the logistic regression model is good because all the score it is more than 90%. From the confusion matrix, we can see that the darkest part is the Snowy Weather type. So we can say that the prediction of this weather type is the best of this model. Also, all the diagonal entrances are much darker than around, so the prediction overall is not so bad.
(2) Decision Tree: The accuracy for this model is 0.91, it is a relatively high rate, so this model is more accurate than the first one. The recall is around 90% for this model. It means that the misclassification rate is around 10%, and for the weather prediction, this is pretty good. Also, the precision is around 90% and can reach 92%, which means that the model's judgment of "positive prediction" is not very credible. And F1 score is also around 90%, the highest is 94% which means the combined performance of the model in terms of completeness and accuracy of detection is good.
(3) Random Forest: The accuracy for this model is 0.91, it is a relatively high rate, so this model is more accurate than the first one. The recall is around 90% for this model. It means that the misclassification rate is around 10%, and for the weather prediction, this is pretty good. Also, the precision is around 90% and can reach 92%, which means that the model's judgment of "positive prediction" is not very credible. And F1 score is also around 90%, the highest is 94% which means the combined performance of the model in terms of completeness and accuracy of detection is good.

And if we look at the details of the report, every term of this model is slightly better than the decision tree model, like the precision can reach 94% and the decision tree model can only to 92%. The recall is also a little bitter higher than the decision tree model. For the decision tree, we can see that it successfully predicts (True Positive) 2396 data out of 2640. The random forest successfully predicts 2426 data out of 2640, so the random forest behaves better.

(4) K-nearest-neighborhood: The accuracy for this model is 0.89, it is a relatively high rate, and it is better than the logistic model but worse than the other two. The recall is like 85% - 90% for this model. It means that the misclassification rate is around 10% - 15%, and for the weather prediction, this is pretty good. Also, the precision is around 85% - 90% and can reach 93%, which means that the model's judgment of "positive prediction" is not very credible. And F1 score is also around 90%, the highest is 94% which means the combined performance of the model in terms of completeness and accuracy of detection is good.

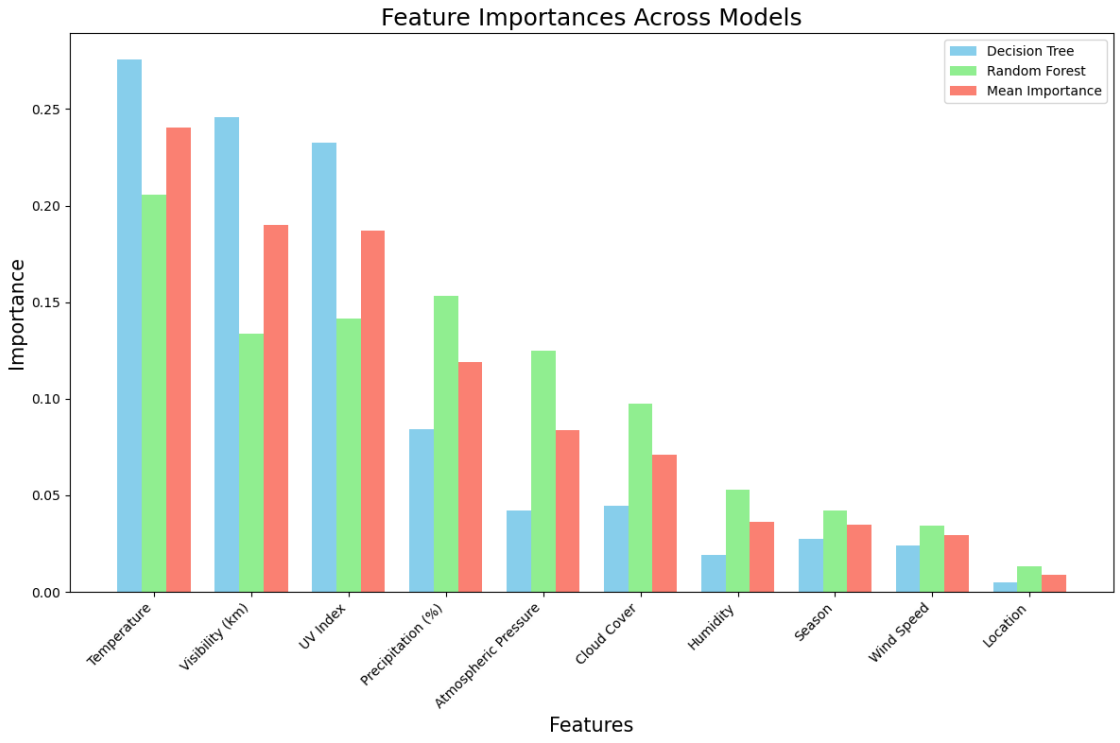
Conclusion: In conclusion, all these four models are good on this dataset. The accuracy rate is all over 85%. Moreover, from the confusion matrix, we can see that all these four models predict the Snowy Weather Type best. The most successful prediction for this type of weather can reach to 94%. And if we compare these four models in detail, we can see that the Random Forest behaves the best. That's because it is the most accurate model, also the recall and the f1-score behave well. The second best one is the Decision Tree, which also has a very high accuracy.

[65]:
```
# feature importance of trees model
decision_tree_importances = decision_tree.feature_importances_
random_forest_importances = random_forest.feature_importances_
# make into dataframe
feature_importances = pd.DataFrame({
    'Feature': X_train.columns,
    'Decision_Tree': decision_tree_importances,
    'Random_Forest': random_forest_importances
})

# Calculate the mean importance across models (only if it makes sense)
feature_importances['Mean_Importance'] = feature_importances[['Decision_Tree',␣
 ↪'Random_Forest']].mean(axis=1)
# Sort the features by their mean importance
sorted_feature_importances = feature_importances.
 ↪sort_values(by='Mean_Importance', ascending=False)
# Display the result
print("Sorted Feature Importances Across Models:")
print(sorted_feature_importances)

# make a plot easy to see
index = np.arange(len(sorted_feature_importances['Feature']))
plt.figure(figsize=(12, 8))
# Bar plots for Decision Tree and Random Forest
plt.bar(index, sorted_feature_importances['Decision_Tree'], 0.25,␣
 ↪label='Decision Tree', color='skyblue')
plt.bar(index + bar_width, sorted_feature_importances['Random_Forest'], 0.25,␣
 ↪label='Random Forest', color='lightgreen')
```

```
# Bar plot for Mean Importance
plt.bar(index + 2 * bar_width, sorted_feature_importances['Mean_Importance'], 0.
 ↪25, label='Mean Importance', color='salmon')
# Adding labels and title
plt.xlabel('Features', fontsize=15)
plt.ylabel('Importance', fontsize=15)
plt.title('Feature Importances Across Models', fontsize=18)
plt.xticks(index + 0.25, sorted_feature_importances['Feature'], rotation=45,␣
 ↪ha='right')
plt.legend()
# Show the plot
plt.tight_layout()
plt.show()
```

Sorted Feature Importances Across Models:

	Feature	Decision_Tree	Random_Forest	Mean_Importance
0	Temperature	0.275402	0.205545	0.240473
8	Visibility (km)	0.245563	0.133900	0.189731
6	UV Index	0.232688	0.141632	0.187160
3	Precipitation (%)	0.084123	0.153431	0.118777
5	Atmospheric Pressure	0.041998	0.125045	0.083522
4	Cloud Cover	0.044617	0.097452	0.071034
1	Humidity	0.019285	0.053049	0.036167
7	Season	0.027347	0.042400	0.034874
2	Wind Speed	0.024230	0.034303	0.029266
9	Location	0.004748	0.013243	0.008996



From the table above, we can find that the most important feature is temperature. It contributes 24% of the total importance. Visibility and UV index also have significant influence on the weather type. These two factors all have around 18% importance. Also, Precipitation is another key feature we need to consider. But the other features are below 10%, so they are not as significant as temperature, visibility, UV Index, and precipitation.

[66]:
```
# isolate the importances for Location and Season
location_importance = feature_importances.loc[feature_importances['Feature'] ==␣
 ↪'Location', 'Mean_Importance'].values[0]
season_importance = feature_importances.loc[feature_importances['Feature'] ==␣
 ↪'Season', 'Mean_Importance'].values[0]
# combine the two importances
location_season_importances = [location_importance, season_importance]
# get the importances for all other features
other_importances = feature_importances.loc[~feature_importances['Feature'].
 ↪isin(['Location', 'Season']), 'Mean_Importance'].values
# perform the Mann White U test
u_statistic, p_value = mannwhitneyu(location_season_importances,␣
 ↪other_importances, alternative='greater')
# print the results
print(f"Mann-Whitney U Statistic: {u_statistic}")
print(f"P-Value: {p_value}")
# report
# significance level
alpha = 0.05
if p_value < alpha:
    print("Fail to reject the null hypothesis")
else:
    print("Reject the null hypothesis")
```

```
Mann-Whitney U Statistic: 1.0
P-Value: 0.9777777777777777
Reject the null hypothesis
```

The p-value is very high (close to 1), which means there is strong evidence against the null hypothesis. This indicates that there is no significant evidence to suggest that "Location and Season" have a unique or significant impact on predicting the weather type compared to other features. In other words, their importance in the model is likely not higher than the importance of the other features. This outcome aligns with our previous observations that "Location" and "Season" had relatively low importance compared to other features like "Temperature" or "Visibility."

1.6.3 Research Question 3

[67]:
```
# import packages
import numpy as np
import pandas as pd
import random
import torch
import matplotlib.pyplot as plt
import seaborn as sns
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
```

[68]:
```
# set seed for recurrence
seed = 42
np.random.seed(seed)
random.seed(seed)
torch.manual_seed(seed)
```

[68]:
```
<torch._C.Generator at 0x7acbd9bd8e70>
```

[69]:
```
# Deal with the data
data = pd.read_csv('weather_classification_data.csv')
labels = {}
for column in ['Cloud Cover', 'Season', 'Location', 'Weather Type']:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    labels[column] = le
# Split the data
# Feature X, target : weather (y)
X = data.drop(columns=['Weather Type'])
y = data['Weather Type']
# scale
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train.values, dtype=torch.long)
y_test = torch.tensor(y_test.values, dtype=torch.long)
```

[70]:
```
# Define the neural net
class WeatherTypePredictionModel(nn.Module):
    def __init__(self, input_size, num_classes, num_epochs):
        super(WeatherTypePredictionModel, self).__init__()
```

```
        self.fc1 = nn.Linear(input_size, 256)  # Increased number of neurons
        self.bn1 = nn.BatchNorm1d(256)  # Batch normalization
        self.fc2 = nn.Linear(256, 128)
        self.bn2 = nn.BatchNorm1d(128)  # Batch normalization
        self.fc3 = nn.Linear(128, num_classes)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)  # Dropout for regularization

        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(self.parameters(), lr=0.001)
        self.num_epochs = num_epochs

    def forward(self, x):
        x = self.relu(self.bn1(self.fc1(x)))
        x = self.dropout(x)  # Apply dropout
        x = self.relu(self.bn2(self.fc2(x)))
        x = self.fc3(x)
        return x

    def train_model(self, X_train, y_train):
        losses = []
        for epoch in range(self.num_epochs):
            self.train()
            outputs = self(X_train)
            loss = self.criterion(outputs, y_train)
            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()
            losses.append(loss.item())

            if (epoch + 1) % 10 == 0:
                print(f'Epoch [{epoch + 1}/{self.num_epochs}], Loss: {loss.
↪item():.4f}')
        plt.plot(range(self.num_epochs), losses, label='Training Loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.title('Training Loss Curve')
        plt.show()

    def predict(self, X_test):
        self.eval()
        with torch.no_grad():
            outputs = self(X_test)
            _, predicted = torch.max(outputs.data, 1)
        return predicted
```

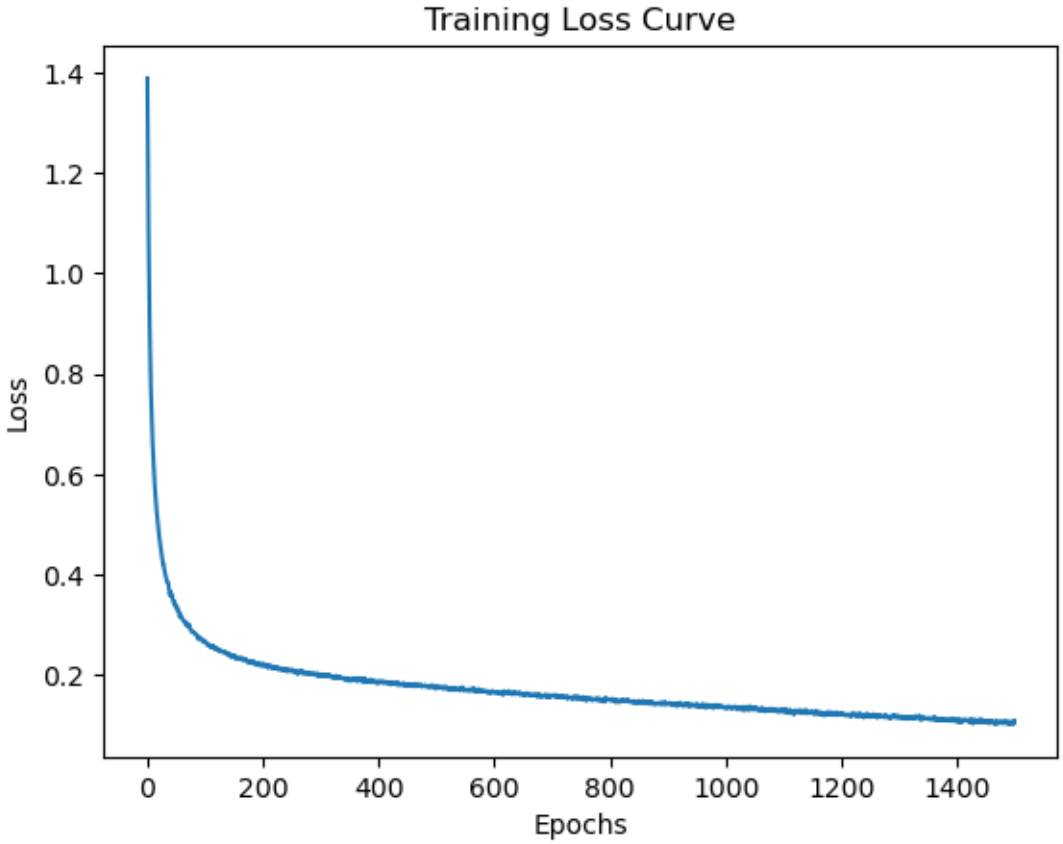
[71]:
```
# use the NN to predict
input_size = X_train.shape[1]
num_classes = len(labels['Weather Type'].classes_)
num_epochs = 1500
model = WeatherTypePredictionModel(input_size, num_classes, num_epochs)
model.train_model(X_train, y_train)
predictions = model.predict(X_test)
```

```
Epoch [10/1500], Loss: 0.6627
Epoch [20/1500], Loss: 0.4898
Epoch [30/1500], Loss: 0.4116
Epoch [40/1500], Loss: 0.3595
Epoch [50/1500], Loss: 0.3355
Epoch [60/1500], Loss: 0.3155
Epoch [70/1500], Loss: 0.2969
Epoch [80/1500], Loss: 0.2845
Epoch [90/1500], Loss: 0.2738
Epoch [100/1500], Loss: 0.2678
Epoch [110/1500], Loss: 0.2577
Epoch [120/1500], Loss: 0.2512
Epoch [130/1500], Loss: 0.2474
Epoch [140/1500], Loss: 0.2436
Epoch [150/1500], Loss: 0.2365
Epoch [160/1500], Loss: 0.2313
Epoch [170/1500], Loss: 0.2319
Epoch [180/1500], Loss: 0.2255
Epoch [190/1500], Loss: 0.2235
Epoch [200/1500], Loss: 0.2207
Epoch [210/1500], Loss: 0.2133
Epoch [220/1500], Loss: 0.2140
Epoch [230/1500], Loss: 0.2132
Epoch [240/1500], Loss: 0.2101
Epoch [250/1500], Loss: 0.2065
Epoch [260/1500], Loss: 0.2058
Epoch [270/1500], Loss: 0.2064
Epoch [280/1500], Loss: 0.2044
Epoch [290/1500], Loss: 0.2000
Epoch [300/1500], Loss: 0.2009
Epoch [310/1500], Loss: 0.1965
Epoch [320/1500], Loss: 0.1953
Epoch [330/1500], Loss: 0.1943
Epoch [340/1500], Loss: 0.1922
Epoch [350/1500], Loss: 0.1916
Epoch [360/1500], Loss: 0.1889
Epoch [370/1500], Loss: 0.1928
Epoch [380/1500], Loss: 0.1899
Epoch [390/1500], Loss: 0.1884
```

```
Epoch [400/1500], Loss: 0.1859
Epoch [410/1500], Loss: 0.1830
Epoch [420/1500], Loss: 0.1839
Epoch [430/1500], Loss: 0.1820
Epoch [440/1500], Loss: 0.1817
Epoch [450/1500], Loss: 0.1819
Epoch [460/1500], Loss: 0.1775
Epoch [470/1500], Loss: 0.1782
Epoch [480/1500], Loss: 0.1782
Epoch [490/1500], Loss: 0.1765
Epoch [500/1500], Loss: 0.1719
Epoch [510/1500], Loss: 0.1748
Epoch [520/1500], Loss: 0.1738
Epoch [530/1500], Loss: 0.1723
Epoch [540/1500], Loss: 0.1704
Epoch [550/1500], Loss: 0.1708
Epoch [560/1500], Loss: 0.1721
Epoch [570/1500], Loss: 0.1673
Epoch [580/1500], Loss: 0.1665
Epoch [590/1500], Loss: 0.1672
Epoch [600/1500], Loss: 0.1639
Epoch [610/1500], Loss: 0.1650
Epoch [620/1500], Loss: 0.1649
Epoch [630/1500], Loss: 0.1628
Epoch [640/1500], Loss: 0.1632
Epoch [650/1500], Loss: 0.1613
Epoch [660/1500], Loss: 0.1612
Epoch [670/1500], Loss: 0.1585
Epoch [680/1500], Loss: 0.1567
Epoch [690/1500], Loss: 0.1568
Epoch [700/1500], Loss: 0.1593
Epoch [710/1500], Loss: 0.1559
Epoch [720/1500], Loss: 0.1559
Epoch [730/1500], Loss: 0.1551
Epoch [740/1500], Loss: 0.1542
Epoch [750/1500], Loss: 0.1524
Epoch [760/1500], Loss: 0.1480
Epoch [770/1500], Loss: 0.1520
Epoch [780/1500], Loss: 0.1524
Epoch [790/1500], Loss: 0.1492
Epoch [800/1500], Loss: 0.1510
Epoch [810/1500], Loss: 0.1490
Epoch [820/1500], Loss: 0.1466
Epoch [830/1500], Loss: 0.1499
Epoch [840/1500], Loss: 0.1447
Epoch [850/1500], Loss: 0.1457
Epoch [860/1500], Loss: 0.1456
Epoch [870/1500], Loss: 0.1428
```

```
Epoch [880/1500], Loss: 0.1433
Epoch [890/1500], Loss: 0.1430
Epoch [900/1500], Loss: 0.1424
Epoch [910/1500], Loss: 0.1403
Epoch [920/1500], Loss: 0.1421
Epoch [930/1500], Loss: 0.1394
Epoch [940/1500], Loss: 0.1366
Epoch [950/1500], Loss: 0.1415
Epoch [960/1500], Loss: 0.1377
Epoch [970/1500], Loss: 0.1382
Epoch [980/1500], Loss: 0.1339
Epoch [990/1500], Loss: 0.1363
Epoch [1000/1500], Loss: 0.1358
Epoch [1010/1500], Loss: 0.1328
Epoch [1020/1500], Loss: 0.1353
Epoch [1030/1500], Loss: 0.1342
Epoch [1040/1500], Loss: 0.1335
Epoch [1050/1500], Loss: 0.1343
Epoch [1060/1500], Loss: 0.1323
Epoch [1070/1500], Loss: 0.1272
Epoch [1080/1500], Loss: 0.1322
Epoch [1090/1500], Loss: 0.1307
Epoch [1100/1500], Loss: 0.1261
Epoch [1110/1500], Loss: 0.1240
Epoch [1120/1500], Loss: 0.1291
Epoch [1130/1500], Loss: 0.1300
Epoch [1140/1500], Loss: 0.1236
Epoch [1150/1500], Loss: 0.1262
Epoch [1160/1500], Loss: 0.1240
Epoch [1170/1500], Loss: 0.1195
Epoch [1180/1500], Loss: 0.1219
Epoch [1190/1500], Loss: 0.1221
Epoch [1200/1500], Loss: 0.1231
Epoch [1210/1500], Loss: 0.1194
Epoch [1220/1500], Loss: 0.1236
Epoch [1230/1500], Loss: 0.1172
Epoch [1240/1500], Loss: 0.1148
Epoch [1250/1500], Loss: 0.1182
Epoch [1260/1500], Loss: 0.1193
Epoch [1270/1500], Loss: 0.1177
Epoch [1280/1500], Loss: 0.1172
Epoch [1290/1500], Loss: 0.1164
Epoch [1300/1500], Loss: 0.1173
Epoch [1310/1500], Loss: 0.1168
Epoch [1320/1500], Loss: 0.1127
Epoch [1330/1500], Loss: 0.1151
Epoch [1340/1500], Loss: 0.1118
Epoch [1350/1500], Loss: 0.1139
```

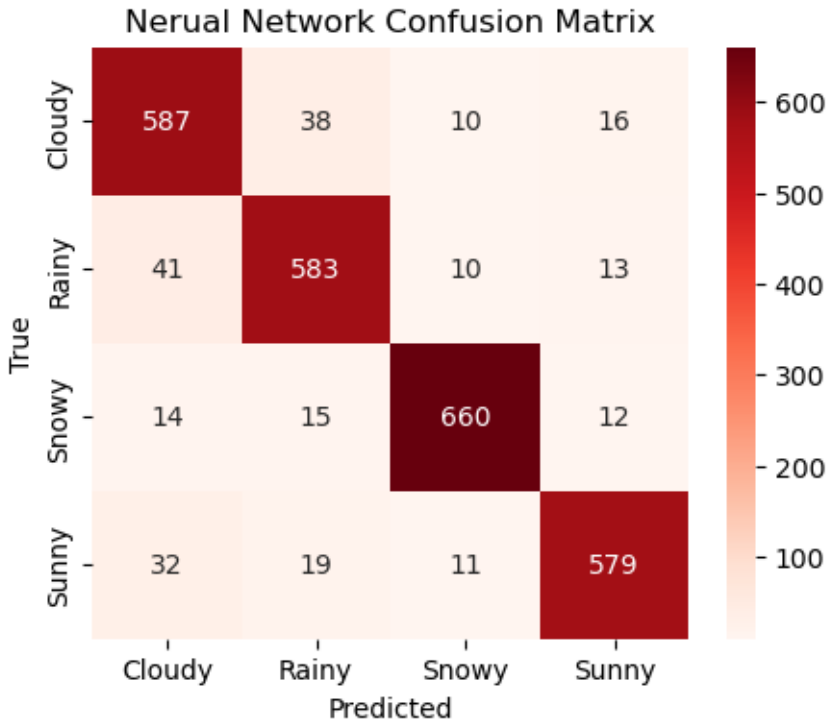
```
Epoch [1360/1500], Loss: 0.1107
Epoch [1370/1500], Loss: 0.1097
Epoch [1380/1500], Loss: 0.1105
Epoch [1390/1500], Loss: 0.1101
Epoch [1400/1500], Loss: 0.1104
Epoch [1410/1500], Loss: 0.1098
Epoch [1420/1500], Loss: 0.1092
Epoch [1430/1500], Loss: 0.1082
Epoch [1440/1500], Loss: 0.1105
Epoch [1450/1500], Loss: 0.1077
Epoch [1460/1500], Loss: 0.1092
Epoch [1470/1500], Loss: 0.1066
Epoch [1480/1500], Loss: 0.1071
Epoch [1490/1500], Loss: 0.1070
Epoch [1500/1500], Loss: 0.1086
```



From the epochs I printed and the plot of the loss function. We can see that the loss of neural net after 1500 epochs is relatively low and close to zero. It is a good sign that our neural network behaves well after 1500 epochs.

[72]:
```
# test the accuracy of the neural net
report = classification_report(y_test, predictions,␣
 ↪target_names=labels['Weather Type'].classes_)
print(report)
cm = confusion_matrix(y_test, predictions)
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Reds", xticklabels=labels['Weather␣
 ↪Type'].classes_, yticklabels=labels['Weather Type'].classes_)
plt.title('Nerual Network Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
              precision    recall  f1-score   support

      Cloudy       0.87      0.90      0.89       651
       Rainy       0.89      0.90      0.90       647
       Snowy       0.96      0.94      0.95       701
       Sunny       0.93      0.90      0.92       641

    accuracy                           0.91      2640
   macro avg       0.91      0.91      0.91      2640
weighted avg       0.91      0.91      0.91      2640
```



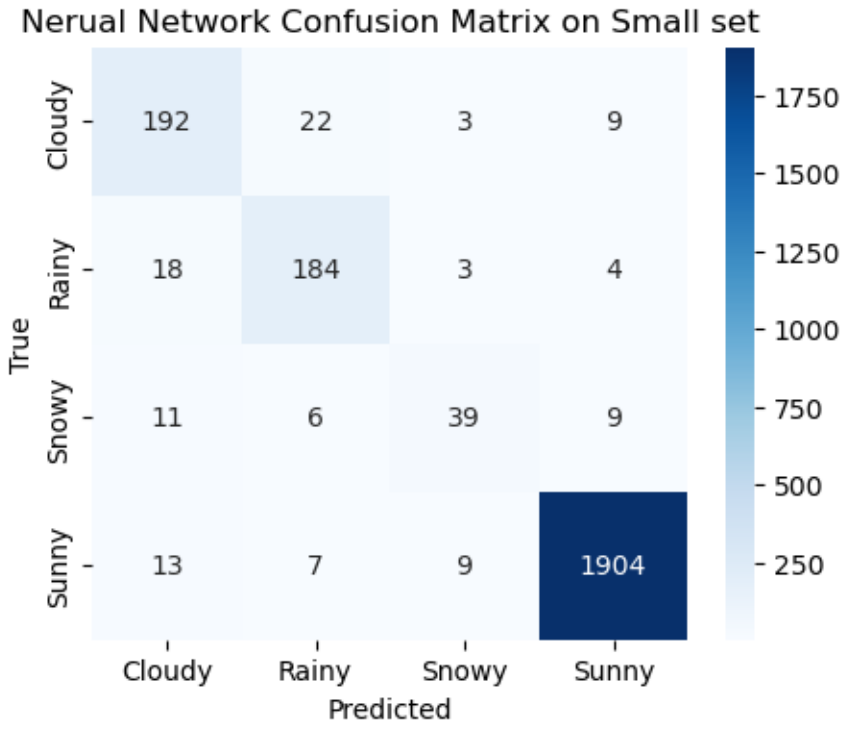
From the report, we can see that the accuracy is high for this neural network. The accuracy rate is 91% for the network. It means that the model behaves overall pretty well. More precisely, from the confusion matrix, this model successfully predicts 2400 out of 2640. And the precision, recall, and the f1-score are around 90%. It means that the misclassification rate is low, and the balance of this model is well. Also, if we look at the confusion matrix more precisely, we can discover that most misclassifications are between classes that might have similar features. For example, some instances of Cloudy are misclassified as Rainy or Snowy, and vice versa. In addition, the snowy class has the highest true positive rate, indicating that the model performs exceptionally well in identifying snowy weather.
Although the model behaves very well, there are also some recommendations for improvement if we want the model more accurate. (1): Further Feature Engineering: Explore additional features that could help differentiate between the classes more effectively. (2)Ensemble Methods: Consider combining the neural network with other models, like Random Forest, to potentially improve accuracy. For instance, we can use the linear method to ensemble the neural network and random forest(which behaves the best in research question 1). (3) Data Augmentation: Increase the amount of training data, especially for classes with slightly lower performance, to help the model generalize better.

[73]:
```
# select the data
filtered_data = data[(data['Temperature'] > 25) & (data['UV Index'] > 5)]
# drop the target column from the filtered dataset to get the features for␣
 ↪prediction
X_filtered = filtered_data.drop(columns=['Weather Type'])

# standardize the data
X_filtered = scaler.transform(X_filtered)
# convert to the tensor
X_filtered = torch.tensor(X_filtered, dtype=torch.float32)
# prediction
predicted_weather = model.predict(X_filtered)
```

[74]:
```
# Test on small set
report = classification_report(y_test, predictions,␣
 ↪target_names=labels['Weather Type'].classes_)
print(report)
cm_small = confusion_matrix(y_filtered, predicted_weather)
plt.figure(figsize=(5, 4))
sns.heatmap(cm_small, annot=True, fmt="d", cmap="Blues",␣
 ↪xticklabels=labels['Weather Type'].classes_, yticklabels=labels['Weather␣
 ↪Type'].classes_)
plt.title('Nerual Network Confusion Matrix on Small set')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
              precision    recall  f1-score   support
```

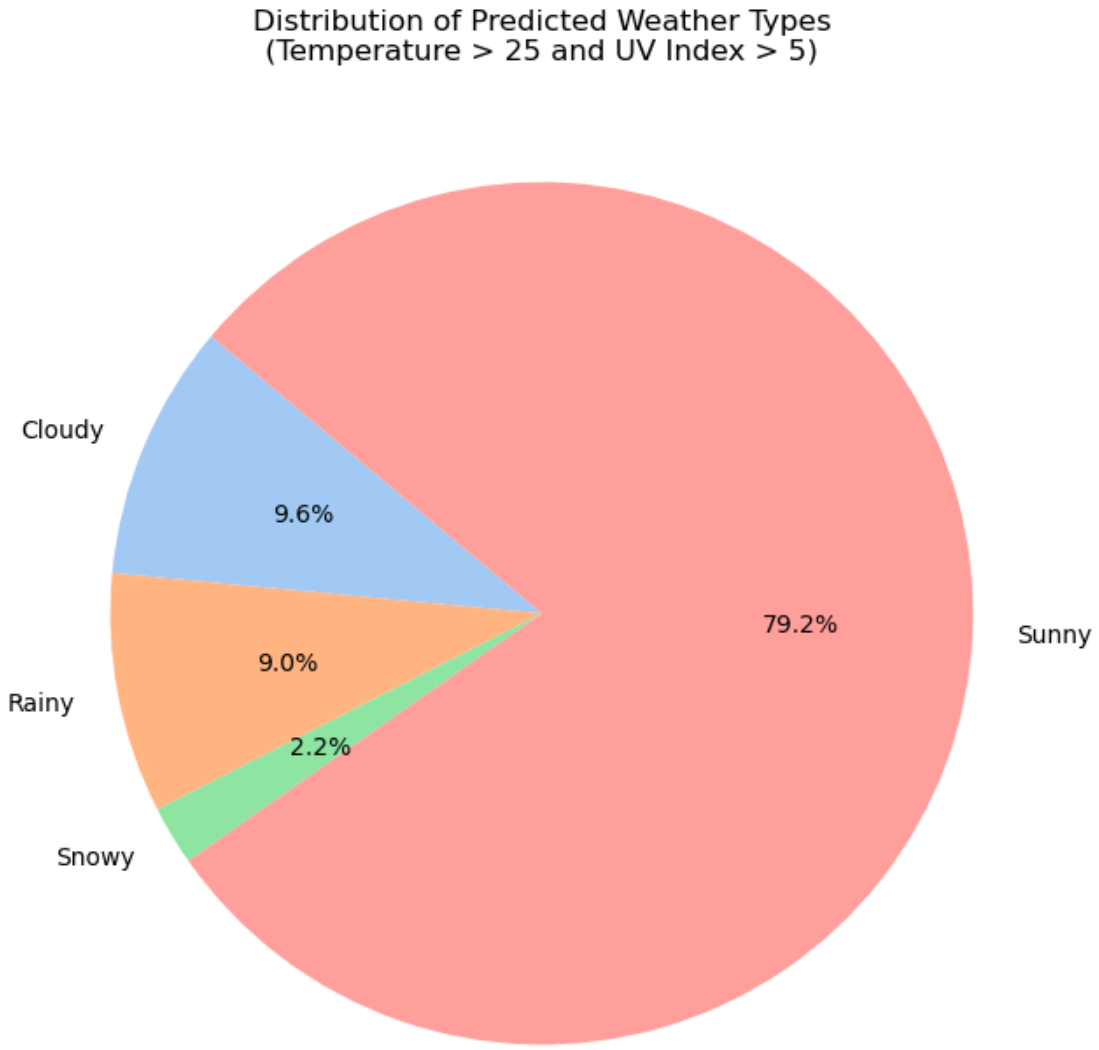
```
              Cloudy       0.87      0.90      0.89       651
               Rainy       0.89      0.90      0.90       647
               Snowy       0.96      0.94      0.95       701
               Sunny       0.93      0.90      0.92       641

            accuracy                           0.91      2640
           macro avg       0.91      0.91      0.91      2640
        weighted avg       0.91      0.91      0.91      2640
```



When I test it on the smaller dataset, it still behaves well. The accuracy rate is very well, it is 91%. And the recall is all above 90%. It means that the misclassify rate is very low, it is good for weather type predictions. So

[75]:
```
#Verify my assumption
# convert to numpy format
predicted_weather_np = predicted_weather.numpy()
# count the occurrences of each weather type
unique, counts = np.unique(predicted_weather_np, return_counts=True)
weather_counts = dict(zip(unique, counts))
# create a pie chart
# use the inverse tranform to get the true label
```

```
weather_labels = labels['Weather Type'].inverse_transform(list(weather_counts.
 ↪keys()))
weather_values = list(weather_counts.values())
plt.figure(figsize=(8, 8))
plt.pie(weather_values, labels=weather_labels, autopct='%1.1f%%',␣
 ↪startangle=140, colors=sns.color_palette('pastel'))
plt.title('Distribution of Predicted Weather Types\n(Temperature > 25 and UV␣
 ↪Index > 5)')
plt.show()
```



The pie chart will show the proportion of each weather type as a percentage of the total predictions.

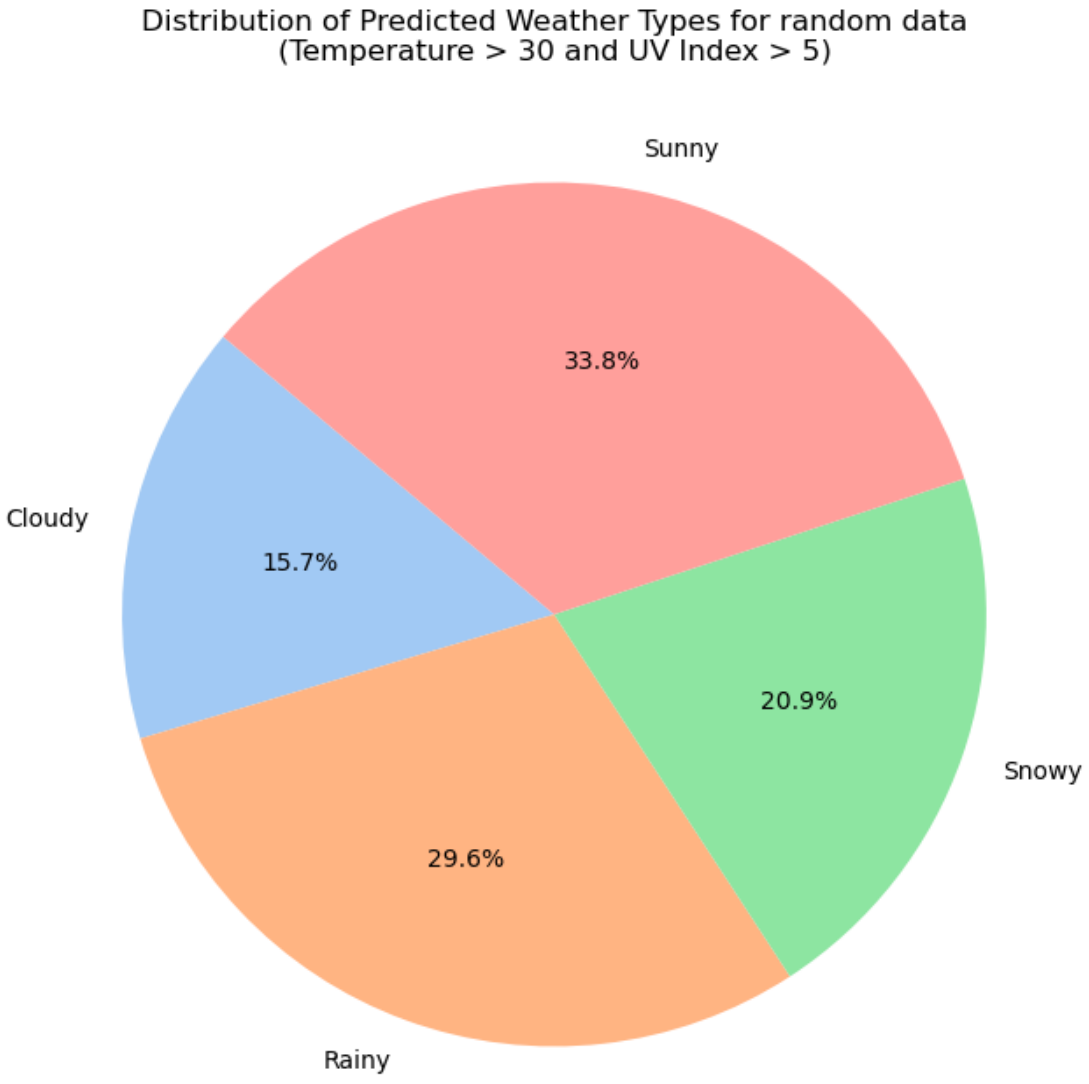
We can see that 79.2% of this dataset is Sunny. It means that my assumption is correct. Also, by intuition, the predictions are almost correct if we have a high UV index and high temperature, which means almost the sun is not been hidden by clouds. So there are almost sunny days. The prediction result matches our intuition.

[76]:
```
# do something on the random set
num_samples = 2000
temperature = np.random.uniform(30, 42, num_samples)
uv_index = np.random.uniform(5, 12, num_samples)
# generate random data
humidity = np.random.uniform(data['Humidity'].min(), data['Humidity'].max(),␣
 ↪num_samples)
wind_speed = np.random.uniform(data['Wind Speed'].min(), data['Wind Speed'].
 ↪max(), num_samples)
precipitation = np.random.uniform(data['Precipitation (%)'].min(),␣
 ↪data['Precipitation (%)'].max(), num_samples)
atmospheric_pressure = np.random.uniform(data['Atmospheric Pressure'].min(),␣
 ↪data['Atmospheric Pressure'].max(), num_samples)
visibility = np.random.uniform(data['Visibility (km)'].min(), data['Visibility␣
 ↪(km)'].max(), num_samples)
cloud_cover = random.choices(data['Cloud Cover'].unique(), k=num_samples)
season = random.choices(data['Season'].unique(), k=num_samples)
location = random.choices(data['Location'].unique(), k=num_samples)
random_data = pd.DataFrame({
    'Temperature': temperature,
    'Humidity': humidity,
    'Wind Speed': wind_speed,
    'Precipitation (%)': precipitation,
    'Cloud Cover': cloud_cover,
    'Atmospheric Pressure': atmospheric_pressure,
    'UV Index': uv_index,
    'Season': season,
    'Visibility (km)': visibility,
    'Location': location
})
```

[77]:
```
# do the prediction
X_random = scaler.transform(random_data)
X_random = torch.tensor(X_random, dtype=torch.float32)
predicted_weather_random = model.predict(X_random)
predicted_weather_random_np = predicted_weather_random.numpy()
```

[48]:
```
#plot
unique_random, counts_random = np.unique(predicted_weather_random_np,␣
 ↪return_counts=True)
weather_counts_random = dict(zip(unique_random, counts_random))
```

```
weather_labels_random = labels['Weather Type'].
  ↪inverse_transform(list(weather_counts_random.keys()))
weather_values_random = list(weather_counts_random.values())
plt.figure(figsize=(8, 8))
plt.pie(weather_values_random, labels=weather_labels_random, autopct='%1.1f%%',␣
  ↪startangle=140, colors=sns.color_palette('pastel'))
plt.title('Distribution of Predicted Weather Types for random␣
  ↪data\n(Temperature > 30 and UV Index > 5)')
plt.show()
```



From the randomly generated dataset(only constrain temperature from 26 to 42 and uv index from

6 to 12), we can see that it basically follows the intuition and the prediction we made on the known dataset. The sunny days are nearly 34%. It can also verify that the model behaves well on the weather prediction.

1.7 Implications and Limitations

1. Lack of Real-World Complexity: The dataset is not representative, it is a data set from Kaggle. It means the dataset is used for the machine learning or test some computer behavior. So if we use the model that trained on this dataset to predict the weather, there may be some misleading information.
2. Limitation of Correlation Matrix: one key limitation of a correlation matrix is that it only captures linear relationships between variables. This means that the correlation matrix can miss important interactions in the data. Also, correlation does not imply causation, so even if two variables are strongly correlated, it doesn't mean one causes the other. Other factors or hidden variables might be influencing both, leading to a fake correlation.
3. Error of Data Distribution: The random generation of data assumes a uniform distribution of features like temperature, UV index, etc., which may not reflect actual weather data distributions. Also, random generation might miss edge cases or rare events that are important for weather prediction

[ ]:

[ ]: