DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING
BACHELORS IN COMPUTER SYSTEMS ENGINEERING
Course Code: CS-324
Course Title: Machine Learning

Complex Engineering Problem

TE Batch 2020, Spring Semester 2023
Grading Rubric
TERM PROJECT

Group Members:

Student No.	Name	Roll No.
S1	**Gorve Kumar**	**CS-125**
S2	**Malaika Ali**	**CS-019**

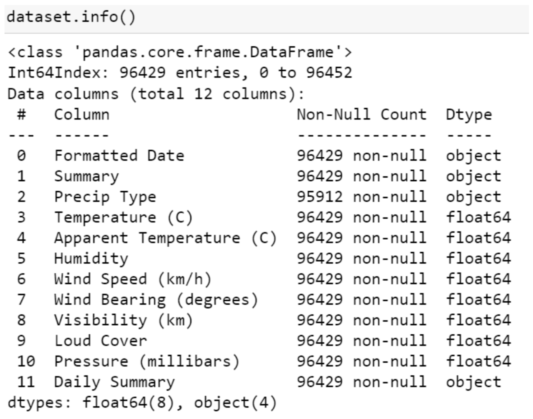
CRITERIA AND SCALES				**Marks Obtained S1**	**Marks Obtained S2**
Criterion 1: Does the application meet the desired specifications and produce the desired outputs? (CPA-1, CPA-2, CPA-3) **[8 marks]**					
1	2	3	4		
The application does not meet the desired specifications and is producing incorrect outputs.	The application partially meets the desired specifications and is producing incorrect or partially correct outputs.	The application meets the desired specifications but is producing incorrect or partially correct outputs.	The application meets all the desired specifications and is producing correct outputs.		
Criterion 2: How well is the code organization? **[2 marks]**					
1	2	3	4		
The code is poorly organized and very difficult to read.	The code is readable only to someone who knows what it is supposed to be doing.	Some part of the code is well organized, while some part is difficult to follow.	The code is well organized and very easy to follow.		
Criterion 3: Does the report adhere to the given format and requirements? **[6 marks]**					
1	2	3	4		
The report does not contain the required information and is formatted poorly.	The report contains the required information only partially but is formatted well.	The report contains all the required information but is formatted poorly.	The report contains all the required information and completely adheres to the given format.		
Criterion 4: How does the student performed individually and as a team member? (CPA-1, CPA-2, CPA-3) **[4 marks]**					
1	2	3	4		
The student did not work on the assigned task.	The student worked on the assigned task, and accomplished goals partially.	The student worked on the assigned task, and accomplished goals satisfactorily.	The student worked on the assigned task, and accomplished goals beyond expectations.		

Final Score = (Criterial_1_score x 2) + (Criteria_2_score / 2) + (Criteria_3_score x (3/2)) + (Criteria_4_score)

= ________________________

Table of Contents

Weather Dataset:



```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 96429 entries, 0 to 96452
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Formatted Date            96429 non-null  object 
 1   Summary                   96429 non-null  object 
 2   Precip Type               95912 non-null  object 
 3   Temperature (C)           96429 non-null  float64
 4   Apparent Temperature (C)  96429 non-null  float64
 5   Humidity                  96429 non-null  float64
 6   Wind Speed (km/h)         96429 non-null  float64
 7   Wind Bearing (degrees)    96429 non-null  float64
 8   Visibility (km)           96429 non-null  float64
 9   Loud Cover                96429 non-null  float64
 10  Pressure (millibars)      96429 non-null  float64
 11  Daily Summary             96429 non-null  object 
dtypes: float64(8), object(4)
```

DATA PREPROCESSING:

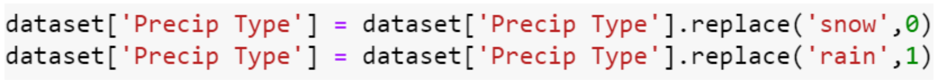
It is a most crucial step before implementing any machine learning algorithm. For data preprocessing, following libraries are used like **NumPy, Pandas, Matplotlib, Seaborn,** and **scikit-learn**.

Scikit-learn is a powerful and widely used open-source machine learning library in Python. It offers a comprehensive set of tools for data preprocessing, feature selection, model training, evaluation, and other essential functionalities.

1. Handling Categorical Data:

Preprocessing techniques, such as label encoding or one-hot encoding, convert categorical variables into numerical representations suitable for machine learning algorithms.

In weather dataset, **'Precip Type'** is a categorical feature with two value 'Snow' and 'Rain', so we can do dummy variable encoding here.



```
dataset['Precip Type'] = dataset['Precip Type'].replace('snow',0)
dataset['Precip Type'] = dataset['Precip Type'].replace('rain',1)
```

2. Feature Engineering:

Creating new features or existing features are transformed to better represent the underlying patterns in the data.

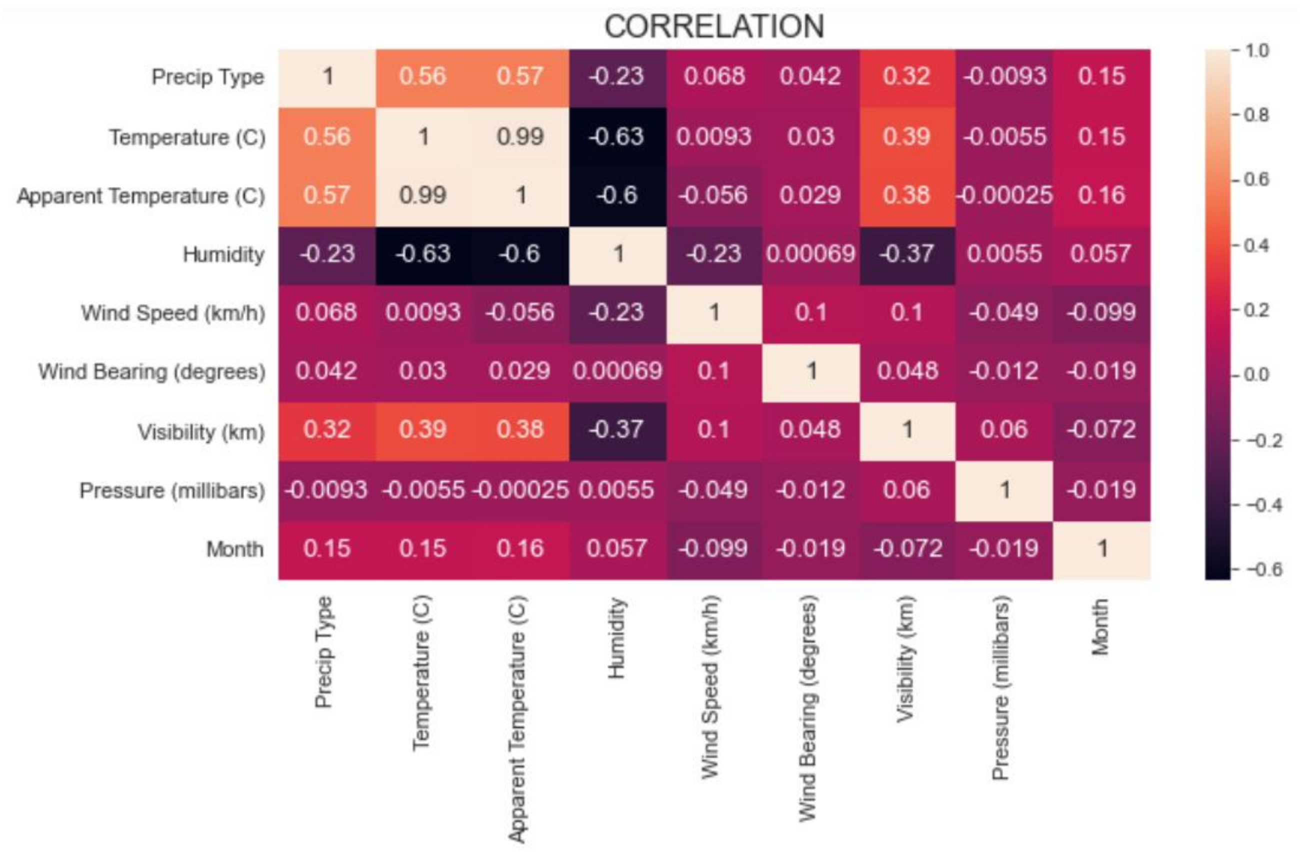
In the weather dataset, the **"Formatted Date"** feature captures the date, which has a significant impact on weather patterns due to different seasons. However, the impact of the year itself is considered to be minimal, and including hourly information would lead to increased computational time for machine learning techniques. Hence, for simplicity and efficiency, we have decided to extract the **"Month"** from the date and consider it as a representative feature.

Furthermore, the **"Daily Summary"** feature is merely a description of our target output column, **"Summary"**. So, we can safely remove it from the dataset since it does not contribute directly to the analysis.

Another feature, **"Loud Cover"** always has a value of 0.0 based on the results of the describe method applied to the dataset. Consequently, it does not provide any meaningful information or contribute to the analysis. Hence, it is better to remove this feature as well.

```
dataset['Formatted Date'] = pd.to_datetime(dataset['Formatted Date'], utc=True)
dataset['Month'] = dataset['Formatted Date'].dt.month
dataset = dataset.drop(['Formatted Date', 'Daily Summary', 'Loud Cover'], axis=1)
```

We constructed a correlation matrix plot, which provides valuable insights into the relationships among the features in the dataset. Understanding the correlations between features helps in identifying redundant or highly influential variables, facilitating the selection of appropriate features for further analysis or modeling tasks.



	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Pressure (millibars)	Month
Precip Type	1	0.56	0.57	-0.23	0.068	0.042	0.32	-0.0093	0.15
Temperature (C)	0.56	1	0.99	-0.63	0.0093	0.03	0.39	-0.0055	0.15
Apparent Temperature (C)	0.57	0.99	1	-0.6	-0.056	0.029	0.38	-0.00025	0.16
Humidity	-0.23	-0.63	-0.6	1	-0.23	0.00069	-0.37	0.0055	0.057
Wind Speed (km/h)	0.068	0.0093	-0.056	-0.23	1	0.1	0.1	-0.049	-0.099
Wind Bearing (degrees)	0.042	0.03	0.029	0.00069	0.1	1	0.048	-0.012	-0.019
Visibility (km)	0.32	0.39	0.38	-0.37	0.1	0.048	1	0.06	-0.072
Pressure (millibars)	-0.0093	-0.0055	-0.00025	0.0055	-0.049	-0.012	0.06	1	-0.019
Month	0.15	0.15	0.16	0.057	-0.099	-0.019	-0.072	-0.019	1

Notably, we observed a high correlation of 99% between features **"Temperature"** & **"Apparent Temperature"**. To prevent bias in the dataset, it is better to remove one of these highly correlated variables, as they convey essentially the same information.

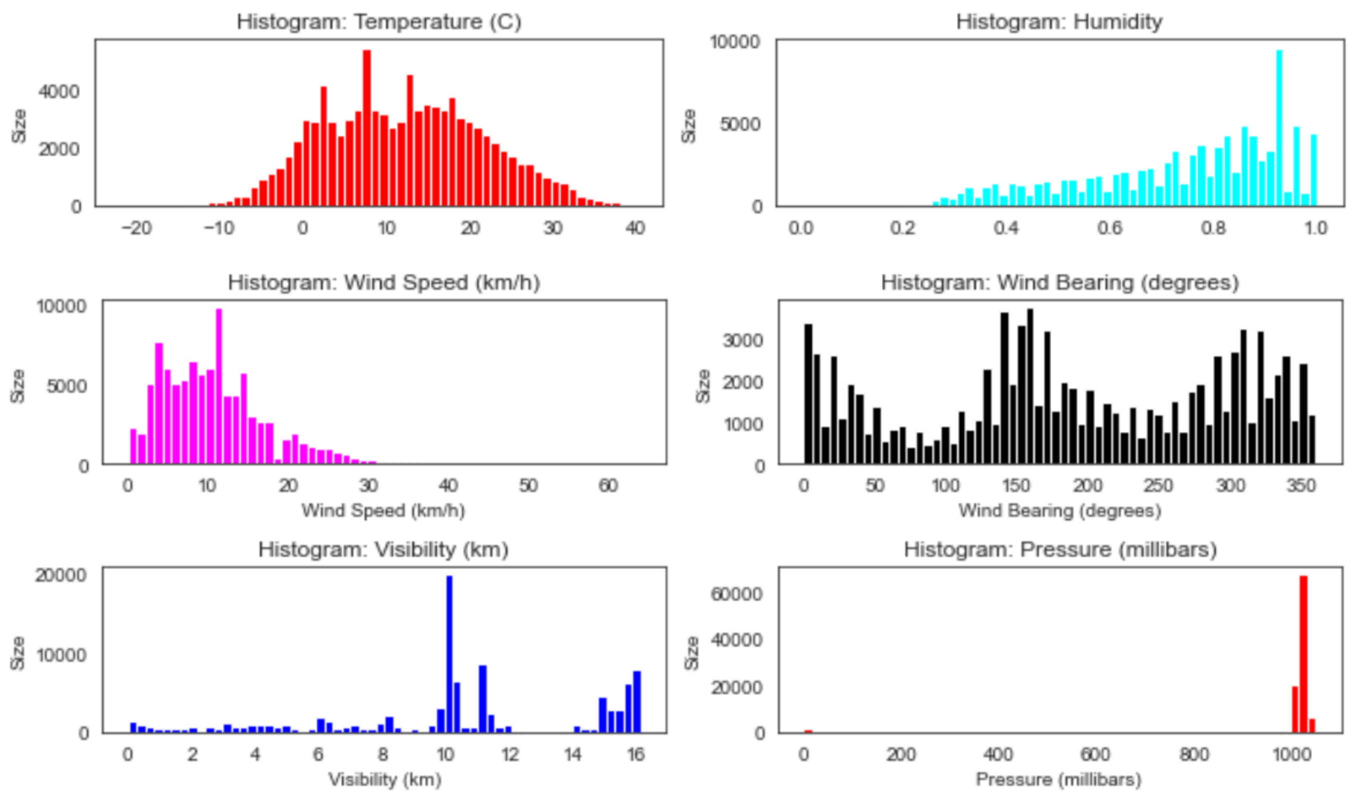
On the other hand, we did not identify any other features that exhibit significant correlation with each other. This suggests that the remaining variables in the dataset are relatively independent and do not display strong linear relationships.

3. Dealing with Outliers:

Outliers are data points that deviate significantly from the majority of the data.

Data Distribution:

Each histogram provides a visual representation of the distribution of the features values for a specific feature in the dataset, allowing for an assessment of outliers or unusual patterns. The range of values is determined based on the minimum and the maximum values of each respective column in the dataset.



Outliers were identified in the features **“Temperature (C)”, “Humidity”, “Wind Speed (km/h)”** and **“Pressure (millibars)”**.

- ✓ Temperature: Values below -12.0°C and above 36.5°C are considered as outliers.
- ✓ Humidity: Entries with a humidity value below 0.22 can be considered as outliers.
- ✓ Wind Speed: Data points with wind speed values exceeding 30.0 km/h are identified as outliers.
- ✓ Pressure: Entries with pressure values below 985.0 millibars or above 1040.0 millibars are identified as outliers.

By applying these outlier removal steps, we aimed to improve the quality and reliability of the dataset for subsequent analysis or modeling tasks. Removing outliers helps mitigate the potential impact of extreme values on statistical analyses and model performance, enabling more accurate and robust results.

4. <u>Feature Scaling:</u>

Different features in a dataset may have different scales or units. Scaling techniques, such as normalization or standardization, are applied to bring all features to a similar scale. This ensures that no single feature dominates the learning process and helps algorithms converge faster.

We will utilize the MinMaxScaler() function from scikit-learn to scale specific columns in the dataset. The features, are transformed to a normalized range between 0 and 1.

```
scaler = MinMaxScaler()
columns = ['Temperature (C)', 'Humidity', 'Wind Speed (km/h)', 'Wind Bearing (degrees)',
           'Visibility (km)', 'Pressure (millibars)', 'Month']
dataset[columns] = scaler.fit_transform(dataset[columns])
```

5. Analyzing the Output Column:

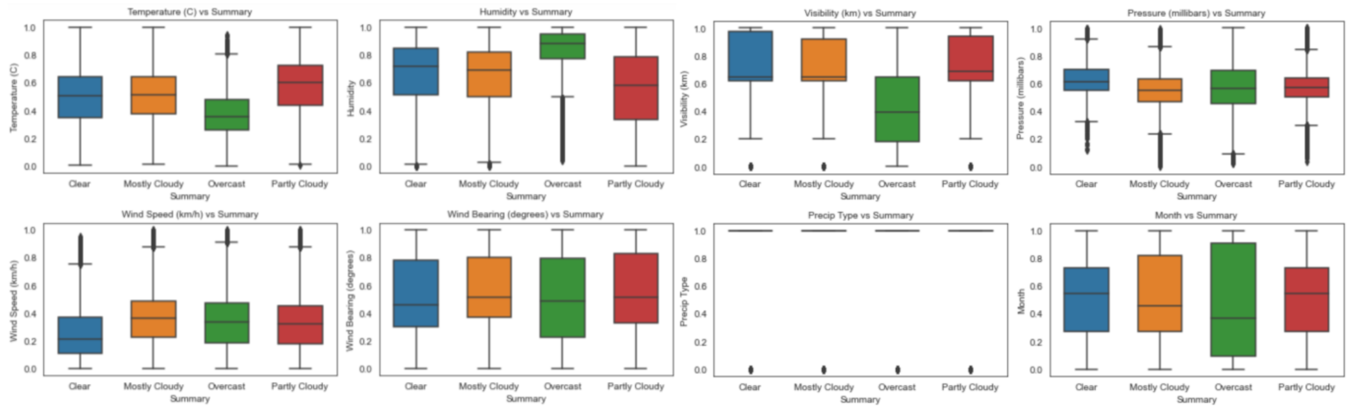
The classes that have fewer instances are removed from the dataset. By applying the filtering, the dataset is refined to focus on the more prevalent classes and eliminate the classes with limited representation.

This filtering step helps address potential imbalances in class distribution, ensuring that the dataset contains a sufficient number of examples for each class. It prevents bias towards the majority classes and enables more accurate and robust modeling.

After applying the filtering and combining the classes Overcast and Foggy, we are left with a refined dataset containing the following classes: **Partly Cloudy**, **Mostly Cloudy**, **Overcast**, & **Clear.** These remaining classes represent the prevailing weather conditions in the dataset after removing the classes with fewer examples.

Analyzing the Relationship between Target and Features:

Encoding the 'Summary' column of the dataset into numerical labels. Plotting the boxplots to visualize the relationship between the encoded 'Summary' variable and all other features. Each boxplot shows the distribution of the numerical feature for different categories of the encoded 'Summary' variable.



The box plots provide insights into the relationship between each feature and the different classes of the 'Summary' column. By visualizing the distributions, it becomes apparent that the **'Precip Type'** feature does not significantly contribute to predicting the output class.

From the distribution, we can observe that **"Partly Cloudy"** and **"Mostly Cloudy"** have almost similar feature distribution, so we merge them into a broader category like **"Cloudy"** aims to simplify and consolidate the classes based on their feature distributions, potentially improving the model's interpretability and performance.

```
dataset = dataset.drop(['Precip Type'], axis=1)
dataset['Summary'] = dataset['Summary'].replace({'Partly Cloudy' : 'Cloudy'})
dataset['Summary'] = dataset['Summary'].replace({'Mostly Cloudy' : 'Cloudy'})
dataset.reset_index(drop=True, inplace=True)
```

The dataset now consists of the following weather classes: Cloudy, Overcast, and Clear.

6. Handling the Missing Data:

Preprocessing allows us to handle missing data effectively. Missing data can adversely affect the performance of machine learning model.

The **'Precip Type'** column in the dataset contains only 0.0536% missing data. As we have already removed this feature from the dataset, therefore we have no missing values left in our dataset.

By performing these preprocessing steps, we can improve the quality and reliability of the data, reduce noise, and make it suitable for various machine learning algorithms, leading to more accurate and robust models.

Saving the Cleaned Dataset:

Saving the cleaned dataset to a CSV file, this creates a clear separation between the data cleaning step and modeling steps. This makes the code modular and easier to understand.

Saving the cleaned dataset to a CSV file helps in data management, reproducibility, and ensuring the availability of clean data for future use.

MACHINE LEARNING ALGORITHMS:

Machine learning models are widely used across various domains and industries due to their ability to learn from data and make predictions or decisions without being explicitly programmed.

1. K-Nearest Neighbors Model (Non-Parametric):

KNN can be used for both classification and regression predictive problems. KNN is a non-parametric algorithm, which means it does not assume any specific data distribution or class imbalance. It can handle imbalanced datasets without requiring additional adjustments or techniques.

KNN is suitable in this case because:

- ✓ It handles multiclass classification problems very effectively. The class label is assigned based on the majority class among the K nearest neighbors.
- ✓ It makes predictions based on the majority class among the K nearest neighbors. This local decision-making process can be beneficial in unbalanced datasets because it gives more weight to the neighboring instances. If the majority of the neighbors belong to a particular class, KNN tends to favor that class, making it effective in capturing the patterns of the minority classes. 3.
- ✓ It can handle complex decision boundaries and nonlinear relationships in the data. This flexibility allows it to better capture the nuances and variations present in unbalanced datasets, where the minority classes may have different distributions and characteristics compared to the majority class.

I. KNN Model 1 (K=3):

Selecting a smaller value for K in the KNN algorithm can result in decision boundaries that are more adaptable to complex patterns and noisy data in the dataset. The model shows relatively good performance on the training set (higher accuracy), but the test accuracy is slightly lower, indicating some overfitting.

The test accuracy is noticeably lower than the training accuracy, which suggests that the model might not generalize well to unseen data.

```
Train Accuracy : 86.9%
Test  Accuracy : 75.1%
```

II. KNN Model 2 (K=5):

A moderate value is chosen for K hyperparameter. This model shows strong performance, achieving an accuracy of 83.7% on the training data and a test accuracy of 76.6%. The model is ability to generalize reasonably well to new data. The resultant model here also does not result in either underfitting or overfitting.

```
Train Accuracy : 83.7%
Test  Accuracy : 76.6%
```

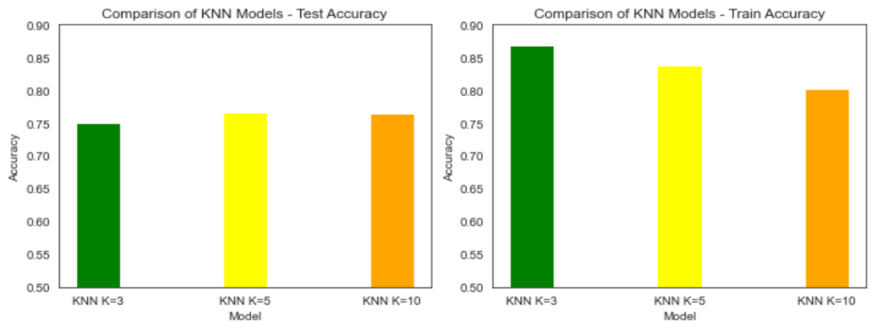
III. KNN Model 3 (K=10):

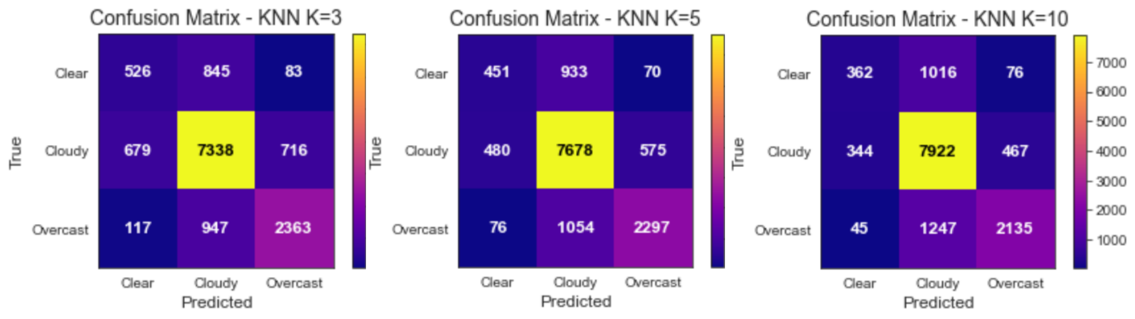
Selecting a larger K value results in smoother decision boundaries, reducing the impact of outliers but potentially oversimplifying the model.

```
Train Accuracy : 80.3%
Test  Accuracy : 76.5%
```

KNN Model Selection:

Since the dataset is unbalanced, using the F1-score is a more appropriate metric for model selection. The F1-score considers both the precision and the recall score, making it more suitable for evaluating models on imbalanced datasets, and it helps us choose a model that achieves a better balance between false positives and false negatives.





```
Recall Score    : 63.05%
Precision Score : 64.96%
F1 Score        : 63.99%
```

```
Recall Score    : 61.99%
Precision Score : 67.43%
F1 Score        : 64.60%
```

```
Recall Score    : 59.30%
Precision Score : 68.57%
F1 Score        : 63.60%
```

Among the three KNN models, the model with K=5 yields highest F1-score, making it the best model to select for this task.

2. Logistic Regression (Parametric):

Logistic regression is a popular statistical technique used for predicting categorical outcomes. It's like a mathematical tool that helps us understand the relationship between input variables and the probability of an event happening or not. By analyzing patterns in the data, logistic regression allows us to make predictions and uncover the factors that influence the outcome. With logistic regression, we can make sense of the data and make informed decisions based on probabilities.

Logistic Regression is suitable in this case because:

- ✓ Firstly, it is computationally efficient and relatively simple to implement.
- ✓ Secondly, it provides interpretable results, allowing us to understand the influence of input variables on each class separately.
- ✓ Lastly, logistic regression can handle imbalanced datasets and noisy data reasonably well.

However, it's worth noting that logistic regression may not be the best choice for highly complex multi-class problems with a large number of classes and intricate relationships between variables.

IV. Logistic Regression Model 1 (C=0.1):

Smaller values of C increase the regularization, helping prevent overfitting.

multi_class='ovr' - The logistic regression is fitted using the one-vs-rest (ovr) strategy for multiclass classification. It treats each class as a binary classification problem.

class_weight='balanced' - This automatically adjusts the weights of classes in inverse proportion to their frequencies. It helps to handle imbalanced class distributions by giving more importance to minority classes.

```
logreg1 =LogisticRegression(C=0.1, multi_class='ovr', class_weight='balanced')

                                Train Accuracy : 60.7%
                                Test  Accuracy : 61.0%
```

This model is underfitting the training set.

V. Logistic Regression Model 2 (C=0.01):

A smaller value of C increases the regularization strength, reducing the impact of large coefficients and potentially preventing overfitting.

multi_class='ovr' - It uses the one-vs-rest (ovr) strategy for multiclass classification.

solver='sag' - This solver stands for Stochastic Average Gradient descent, which is an optimization algorithm suitable for large datasets.

l2 regularization (Ridge regularization) is used, which adds the squared magnitude of coefficients to the loss function to prevent overfitting.

```
logreg2 =LogisticRegression(C=0.01,multi_class='ovr',solver='sag',penalty='l2')
                                Train Accuracy : 72.7%
                                Test  Accuracy : 73.2%
```

This model performs reasonably well on seen and unseen data.

VI. Logistic Regression Model 3 (C=5):

A higher value of C reduces the regularization strength, allowing the model to fit the training data more closely.

multi_class='ovr' - It uses the one-vs-rest (ovr) strategy for multiclass classification.

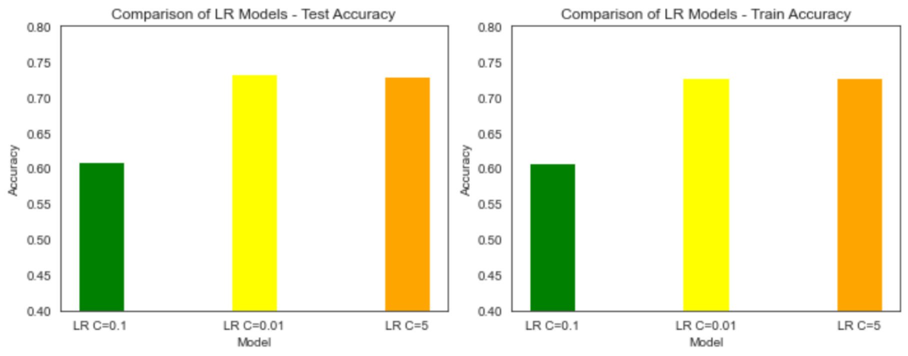
L1 regularization (Lasso regularization) is used, which adds the absolute magnitude of coefficients to the loss function. L1 regularization can lead to sparse solutions, as it tends to set some coefficients to zero.

```
logreg3 =LogisticRegression(C=5,multi_class='ovr',penalty='l2')
```

```
Train Accuracy : 72.7%
Test  Accuracy : 72.9%
```

The tabular and graphical representations above show that increasing or decreasing the values of C does not affect the model' performance much. We have seen only a slight variation in the accuracies of the models on increasing the value of C. The regularization can impact model performance. C is the inverse of regularization strength.

Logistic Regression Model Selection:



Overall, based on the results, Model 2 and Model 3 seem to perform better than Model 1. The choice of hyperparameters can significantly impact the model's performance, and it's essential to select the best hyperparameters to achieve optimal results. Additionally, these results highlight the importance of evaluating models on both training and test datasets to assess their generalization abilities.

So, the best model of logistic regression is model 3 i.e.; with C=5.

3. <u>Artificial Neural Network:</u>

Inspired by the structure and functioning of the human brain, ANNs are computational models that can learn and make decisions on their own. Their ability to process complex data, learn patterns, and make accurate predictions has made them a popular choice for solving a wide range of problems, including image recognition, natural language processing, and financial forecasting.

ANNs consist of interconnected artificial neurons organized in layers, with each neuron connected to neurons in adjacent layers. These connections have weights associated with them, which determine the strength or importance of the information being passed. During the training process, the network adjusts these weights based on the input data and desired output, striving to minimize the difference between predicted and actual results.

ANN is suitable in this case because:

- ✓ One of the key reasons of using an ANN model is its ability to handle multiclass classification problems. In multiclass classification, the goal is to classify instances into more than two classes. ANNs excel at capturing complex non-linear relationships between input features and class labels, making them well-suited for tasks that involve intricate patterns and decision boundaries.

- ✓ ANNs are capable of learning complex non-linear relationships between input features and class labels. This makes them effective for capturing intricate patterns and decision boundaries that exist in multiclass classification problems. Other linear classifiers may struggle with such complexities.

- ✓ ANNs are highly scalable and can handle large datasets with numerous input features and multiple classes. They can effectively process and learn from vast amounts of data, which is often required in real-world multiclass classification problems.

- ✓ ANNs offer flexibility in terms of network architecture and layer configurations. They can be designed with various hidden layers and different activation functions to suit the complexity of the problem. This flexibility allows ANNs to adapt to different types of multiclass classification tasks.

- ✓ ANNs have the ability to adapt and update their internal weights based on new data. This means that as new examples become available, the network can continue learning and improving its classification performance over time.

Before implementing the ANN model:

i. The **LabelEncoder** is used to convert categorical target labels into numeric values, enabling the ANN to process them during training.

ii. The **train_test_split** function is applied to split the data into training and validation sets (80% training, 20% validation) to evaluate the ANN's performance and prevent overfitting.

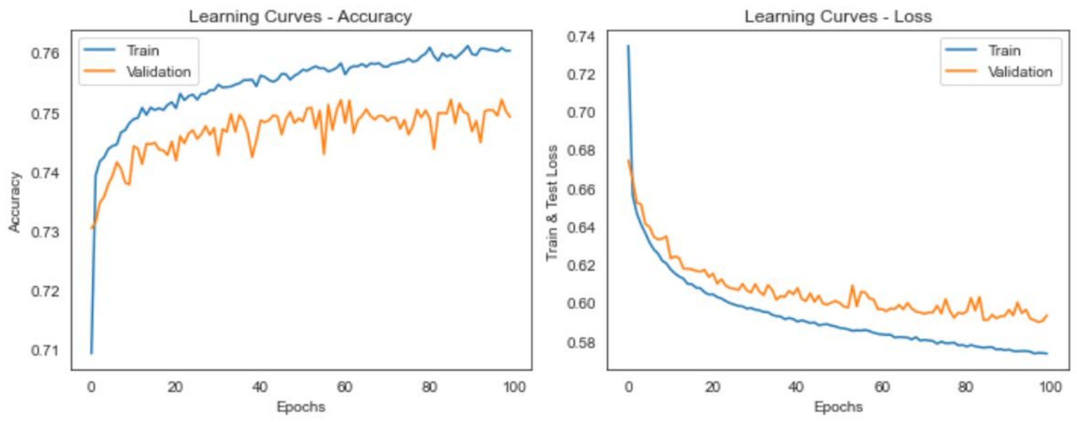
iii. The **class_label_mapping** is being used in the ANN to decode the numeric predictions into their corresponding class labels. In the ANN model, after making predictions, the output is usually a vector of probabilities representing the likelihood of the input belonging to each class. The predicted output may look like [0.1, 0.7, 0.2], where each value corresponds to the probability of the input belonging to a specific class.

    The class_label_mapping is crucial here because it serves as a reference to map the numeric predictions back to their original class labels. By using the ***np.argmax()*** function, the index of the highest probability in the predicted output is obtained, and then this index is used to access the corresponding class label. It is essential for providing meaningful and human-readable weather summaries as the output of the ANN model instead of dealing with numeric values directly.

VII. ANN Model 1 (Batch_size = 100, Epochs = 100):

This model defined with four layers: three hidden layers with **64, 32** and **16 units**, respectively, using the **Relu** activation function, and an output layer with 3 units and the **softmax** activation function.

The model is compiled with the ***Adam optimizer***, using the sparse categorical cross-entropy loss function for multi-class classification. The model is trained using the training data for ***100 epochs*** with a ***batch size 100***, and the training history is stored in ***history_1***.



The model is ***overfitting*** the training data. This means that the model is learning the specific details of the training data, but it is not learning the ***general patterns*** that would allow it to make accurate predictions on ***new data***.

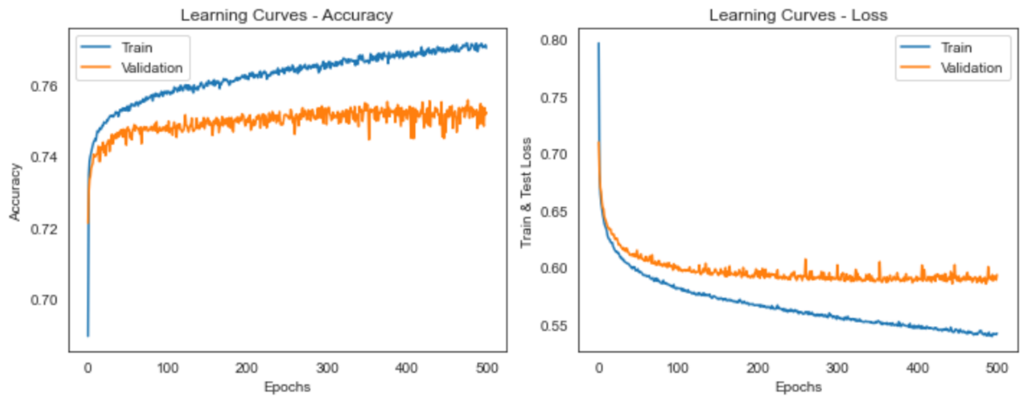
As a result, the model is not able to make accurate predictions on new data with greater epochs on this model.

VIII. ANN Model 2 (Batch_size = 1000, Epochs = 500):

This model is defined with five layers: four hidden layers with **64, 64, 32** and **16 units**, respectively, all using the ***Relu*** activation function, and an output layer with 3 units and the ***softmax*** activation function.

The model is compiled with the Adam optimizer and uses the sparse categorical cross-entropy loss function for multi-class classification. No custom loss weights are specified.

The model is trained using the training data for ***500 epochs*** with a ***batch size 1000***. The training history is stored in ***history_2.***



The learning curves indicate that the model is **learning and improving** over time, and is not overfitting. So, the model is likely to generalize well to new data. Accuracy measures how often the model makes the correct prediction, while loss measures the average error of the model's predictions.

The **learning curves are not perfectly smooth**, there will always be some variation in the performance of a model, even when it is learning well.

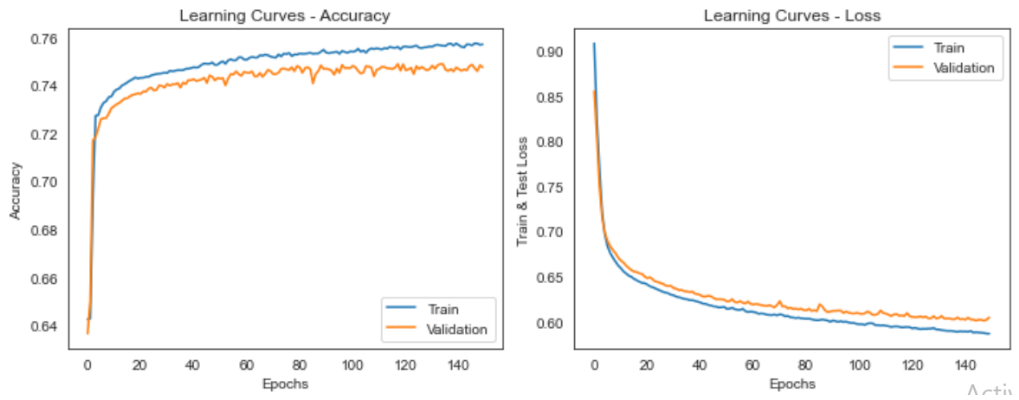
The learning curves plateau after a certain number of epochs. This is because the model is eventually able to learn all of the information in the training data. **The model can only improve by learning from new data.**

IX. ANN Model 3 (Batch_size = 5000, Epochs = 150):

This model is defined with four layers: three hidden layers with **128, 64,** and **32 units,** respectively, all using the ***Relu*** activation function, and an output layer with 3 units and the ***softmax*** activation function.

The model is compiled with the ***Adam*** optimizer and uses the sparse categorical cross-entropy loss function for multi-class classification. No custom loss weights are specified.

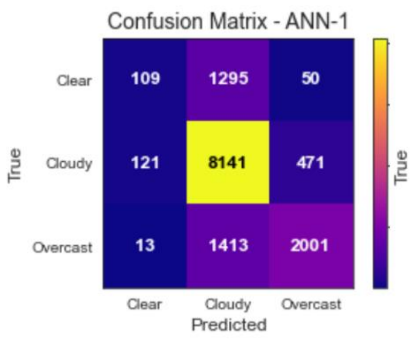
The model is trained using the training data for **150 epochs** with a **batch size 5000.** The training history is stored in ***history_3.***



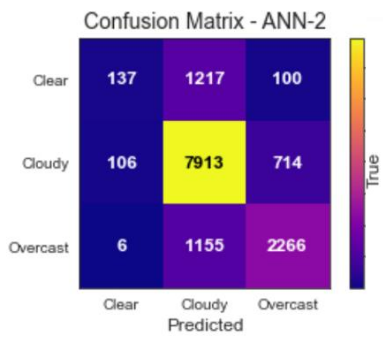
The accuracy of the model increases as the number of training epochs increases and loss of the model also decreases as the number of training epochs increases, indicating that the model is ***learning and improving*** its performance over time. Thus, model is well suited for the dataset. The validation accuracy is slightly lower than the training accuracy and thus indicating that the model is not overfitting the training data. The validation loss is also slightly higher than the training loss.

The learning curves are smooth, which indicates that the model is not making large jumps in performance as the number of training epochs increases.

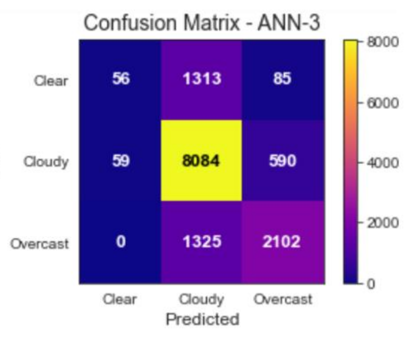
ANN Model Selection:



```
Recall Score    : 53.04%
Precision Score : 66.41%
F1 Score        : 58.97%
```



```
Recall Score    : 55.38%
Precision Score : 68.51%
F1 Score        : 61.25%
```



```
Recall Score    : 52.59%
Precision Score : 66.60%
F1 Score        : 58.77%
```

Based on the accuracy, it appears that model 3 performs the best among the ANN models.

However, considering the presence of ***unbalanced data***, it is more appropriate to use the ***F1 Score*** as a metric to select the best algorithm. Hence, we choose ***model 2*** as the preferred ANN model due to its better performance in handling the imbalanced class distribution.

Some Other Models:

1. Naive Bayes (MultinomialNB, GaussianNB):

It is a probabilistic algorithm based on Bayes' theorem. It assumes that features are conditionally independent given the class label. Naive Bayes is simple and efficient.

Naive Bayes is a simple yet powerful machine learning algorithm that is commonly used for classification tasks.

The ***Multinomial Naive Bayes model*** has alpha value of 0.5. The alpha value represents the Laplace smoothing parameter, which is used to handle unseen features in the test data. A smaller alpha value (like 0.5) means less smoothing, which can lead to better performance when dealing with discrete data.

The ***Gaussian Naive Bayes model*** has a var_smoothing value of 0.1. The var_smoothing parameter is used to add a small value to the variances of all features, preventing the possibility of division by zero when calculating probabilities. A smaller var_smoothing value (like 0.1) adds less smoothing to the variances, potentially improving the model's performance on continuous data.

```
Naive Bayes (alpha=0.5) Train Accuracy: 64.3%
Naive Bayes (alpha=0.5) Test Accuracy: 64.1%
Gaussian Naive Bayes (var_smoothing=0.1) Train Accuracy: 70.5%
Gaussian Naive Bayes (var_smoothing=0.1) Test Accuracy: 70.7%
```

2. Support Vector Machine:

SVM is a non-parametric machine learning algorithm. Non-parametric algorithms do not make strong assumptions about the underlying data distribution or the number of parameters needed to represent it.

In SVM, the goal is to find the hyperplane that maximizes the margin between different classes while minimizing the classification error. The number of support vectors, which are the data points closest to the decision boundary, determines the number of parameters needed to represent the model.

```
SVM (C=0.1, linear kernel) Train Accuracy: 73.0%
SVM (C=0.1, linear kernel) Test Accuracy: 73.3%
```

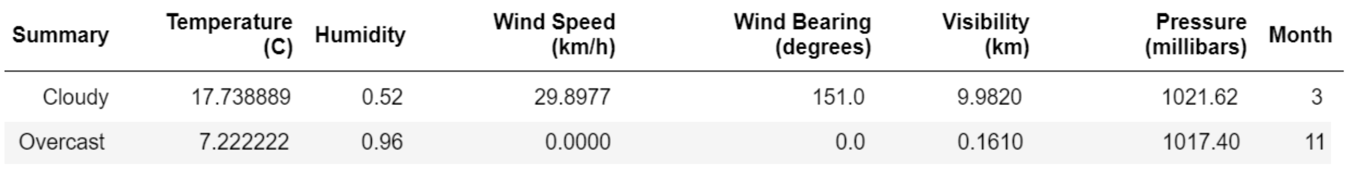
User Interface:

The program allows the user to choose between three machine learning models (K-Nearest Neighbors, Logistic Regression, and Artificial Neural Network) to predict the weather summary based on input weather features.

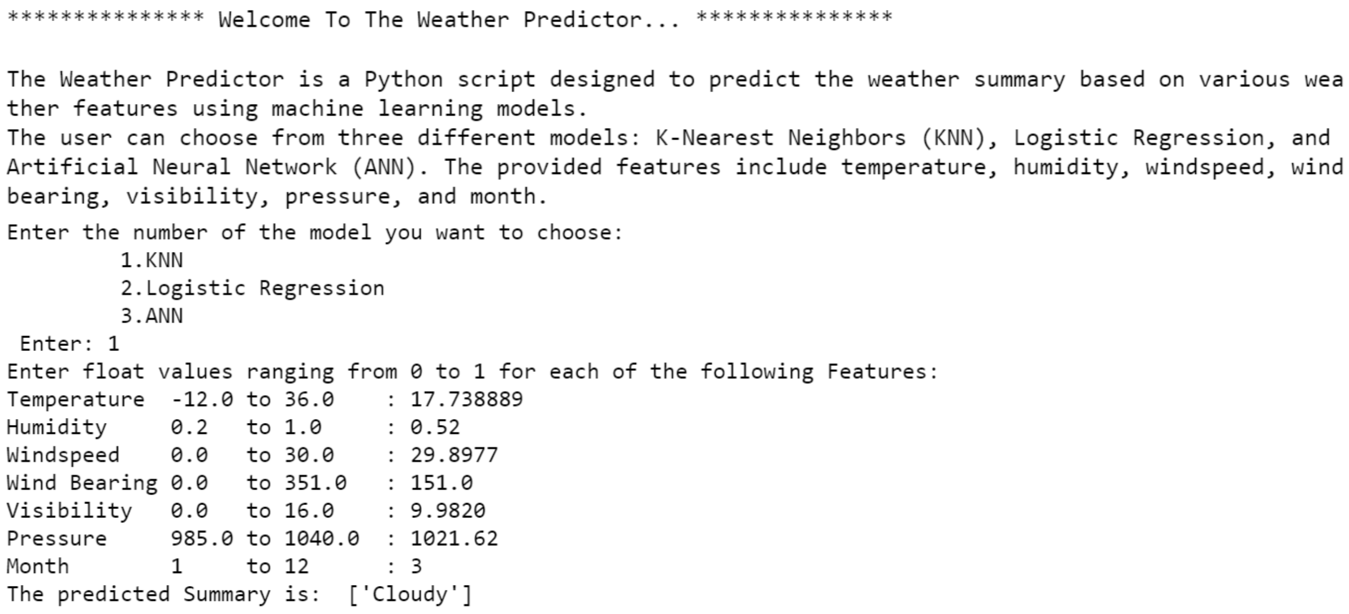
The user is prompted to enter float values for temperature, humidity, windspeed, wind bearing, visibility, pressure, and month, and these values are then normalized between 0 and 1.

Depending on the chosen model, the script makes predictions using the corresponding machine learning model and displays the predicted weather summary. If an invalid model number is entered, an error message is shown.

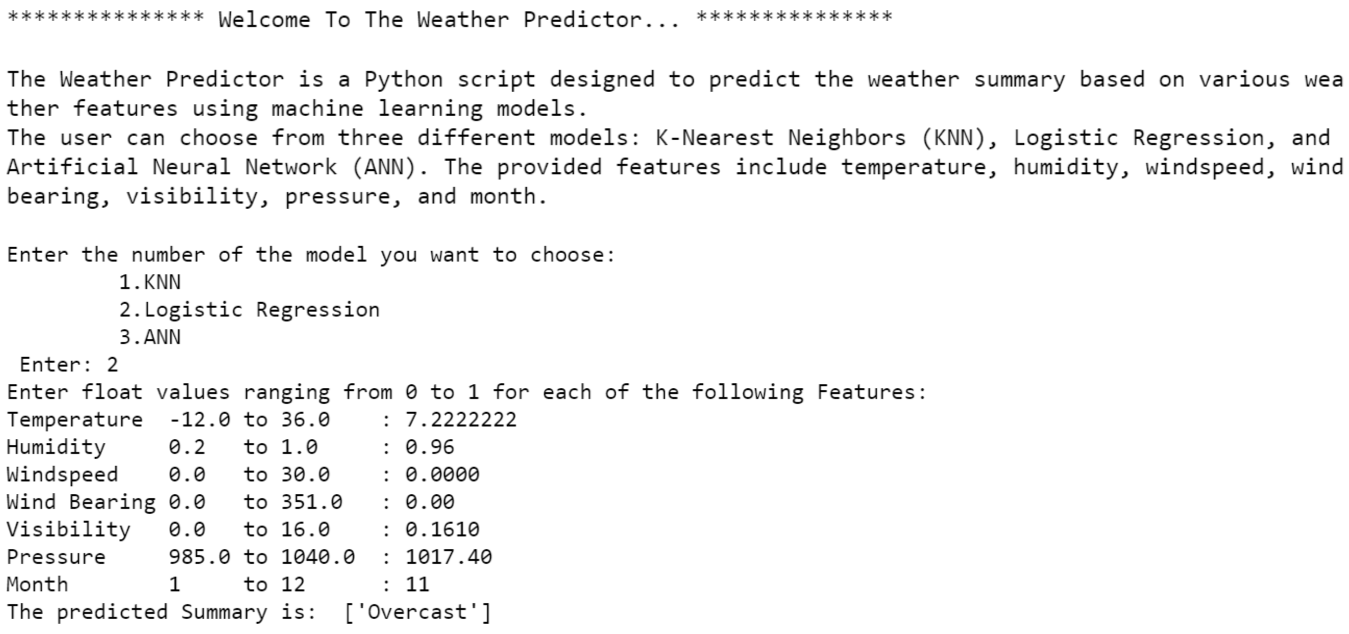
Test Cases:



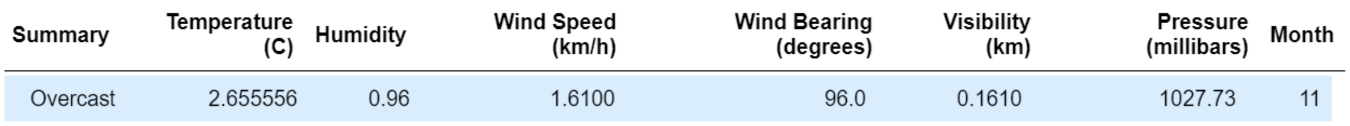
Summary	Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Pressure (millibars)	Month
Cloudy	17.738889	0.52	29.8977	151.0	9.9820	1021.62	3
Overcast	7.222222	0.96	0.0000	0.0	0.1610	1017.40	11



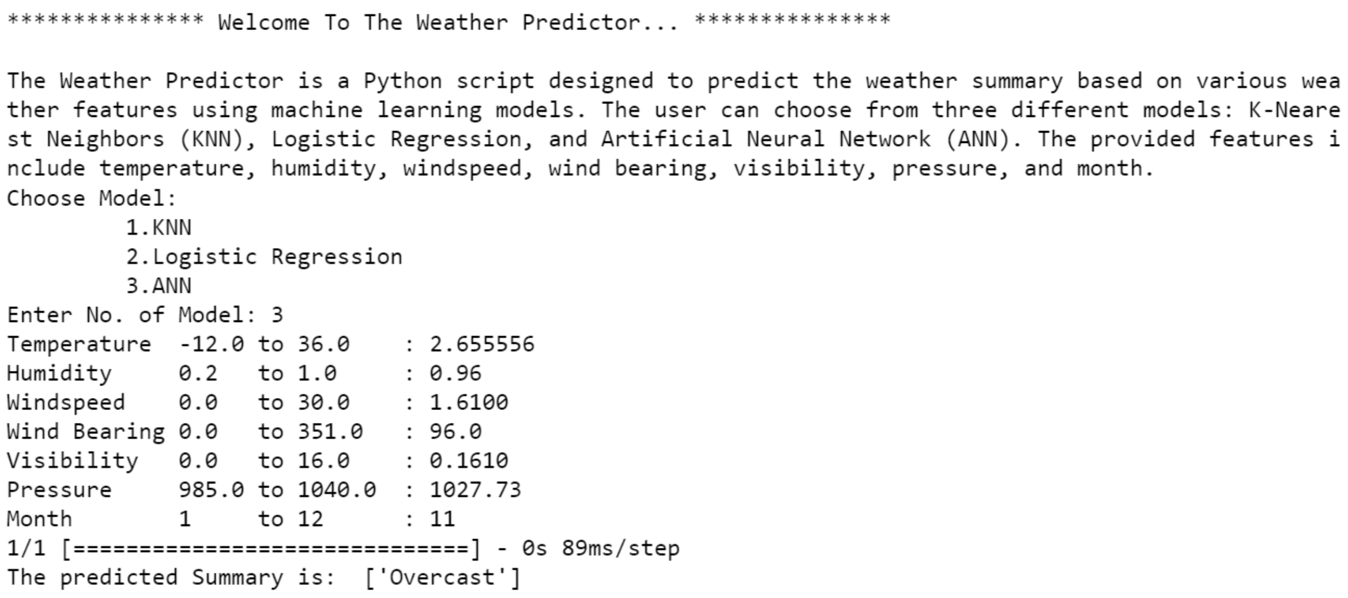
```
*************** Welcome To The Weather Predictor... ***************

The Weather Predictor is a Python script designed to predict the weather summary based on various wea
ther features using machine learning models.
The user can choose from three different models: K-Nearest Neighbors (KNN), Logistic Regression, and
Artificial Neural Network (ANN). The provided features include temperature, humidity, windspeed, wind
bearing, visibility, pressure, and month.
Enter the number of the model you want to choose:
         1.KNN
         2.Logistic Regression
         3.ANN
 Enter: 1
Enter float values ranging from 0 to 1 for each of the following Features:
Temperature  -12.0 to 36.0    : 17.738889
Humidity     0.2   to 1.0     : 0.52
Windspeed    0.0   to 30.0    : 29.8977
Wind Bearing 0.0   to 351.0   : 151.0
Visibility   0.0   to 16.0    : 9.9820
Pressure     985.0 to 1040.0  : 1021.62
Month        1     to 12      : 3
The predicted Summary is:  ['Cloudy']
```



```
*************** Welcome To The Weather Predictor... ***************

The Weather Predictor is a Python script designed to predict the weather summary based on various wea
ther features using machine learning models.
The user can choose from three different models: K-Nearest Neighbors (KNN), Logistic Regression, and
Artificial Neural Network (ANN). The provided features include temperature, humidity, windspeed, wind
bearing, visibility, pressure, and month.

Enter the number of the model you want to choose:
         1.KNN
         2.Logistic Regression
         3.ANN
 Enter: 2
Enter float values ranging from 0 to 1 for each of the following Features:
Temperature  -12.0 to 36.0    : 7.2222222
Humidity     0.2   to 1.0     : 0.96
Windspeed    0.0   to 30.0    : 0.0000
Wind Bearing 0.0   to 351.0   : 0.00
Visibility   0.0   to 16.0    : 0.1610
Pressure     985.0 to 1040.0  : 1017.40
Month        1     to 12      : 11
The predicted Summary is:  ['Overcast']
```



Summary	Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Pressure (millibars)	Month
Overcast	2.655556	0.96	1.6100	96.0	0.1610	1027.73	11



```
*************** Welcome To The Weather Predictor... ***************

The Weather Predictor is a Python script designed to predict the weather summary based on various wea
ther features using machine learning models. The user can choose from three different models: K-Neare
st Neighbors (KNN), Logistic Regression, and Artificial Neural Network (ANN). The provided features i
nclude temperature, humidity, windspeed, wind bearing, visibility, pressure, and month.
Choose Model:
         1.KNN
         2.Logistic Regression
         3.ANN
Enter No. of Model: 3
Temperature  -12.0 to 36.0    : 2.655556
Humidity     0.2   to 1.0     : 0.96
Windspeed    0.0   to 30.0    : 1.6100
Wind Bearing 0.0   to 351.0   : 96.0
Visibility   0.0   to 16.0    : 0.1610
Pressure     985.0 to 1040.0  : 1027.73
Month        1     to 12      : 11
1/1 [==============================] - 0s 89ms/step
The predicted Summary is:  ['Overcast']
```

References:

- https://scikit-learn.org/stable/
- https://matplotlib.org/https://seaborn.pydata.org/
- https://pandas.pydata.org/pandas-docs/stable/user_guide/missing_data.html
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://towardsdatascience.com/a-complete-guide-to-naive-bayes-algorithm-20c49beae7ac