Importing the Necessary Libraries

First we import the necessary libraries of the python for demostration of the Decision Tree Classifier

In [1]:

```
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

Read the data of the weather from the csv file using read_csv function of pandas dataframe

In [3]:

```
data = pd.read_csv('daily_weather.csv')
```

Daily Weather Data Description

The file **daily_weather.csv** is a comma-separated file that contains weather data. This data comes from a weather station located in San Diego, California. The weather station is equipped with sensors that capture weather-related measurements such as air temperature, air pressure, and relative humidity. Data was collected for a period of three years, from September 2011 to September 2014, to ensure that sufficient data for different seasons and weather conditions is captured.

Let's now check all the columns in the data.

Know about various columns in the dataset.

In [4]:

```
data.columns
```

Out[4]:

```
Index(['number', 'air_pressure_9am', 'air_temp_9am', 'avg_wind_direction_9
am',
       'avg_wind_speed_9am', 'max_wind_direction_9am', 'max_wind_speed_9a
m',
       'rain_accumulation_9am', 'rain_duration_9am', 'relative_humidity_9a
m',
       'relative_humidity_3pm'],
      dtype='object')
```

In [5]:

```
data.head()
```

Out[5]:

	number	air_pressure_9am	air_temp_9am	avg_wind_direction_9am	avg_wind_speed_9am
0	0	918.060000	74.822000	271.100000	2.080354
1	1	917.347688	71.403843	101.935179	2.443009
2	2	923.040000	60.638000	51.000000	17.067852
3	3	920.502751	70.138895	198.832133	4.337363
4	4	921.160000	44.294000	277.800000	1.856660

Checking is there exists null values in the dataset or not

In [6]:

```
data[data.isnull().any(axis=1)].head()
```

Out[6]:

	number	air_pressure_9am	air_temp_9am	avg_wind_direction_9am	avg_wind_speed_9an
16	16	917.890000	NaN	169.200000	2.19220
111	111	915.290000	58.820000	182.600000	15.61384
177	177	915.900000	NaN	183.300000	4.71994
262	262	923.596607	58.380598	47.737753	10.63627
277	277	920.480000	62.600000	194.400000	2.75143

Exploratory Data Analysis

We will not need to number for each row so we can clean it.

Data Cleaning process --> As number column contains unique values which can not help us making any decision

In [7]:

```
del data['number']
```

Calculatoing the amount of data or say number of rows in the dataset before removing the rows containg null values

In [8]:

```
before_rows = data.shape[0]
print(before_rows)
```

```
1095
```

Removing the rows which contains the null values

In [9]:

```
data = data.dropna()
```

Calculatoing the amount of data or say number of rows in the dataset after removing the rows containg null values

In [10]:

```
after_rows = data.shape[0]
print(after_rows)
```

```
1064
```

Calculate how many rows are deleted which contains the Null Values

In [ ]:

```
```

In [11]:

```
before_rows - after_rows
```

Out[11]:

```
31
```

Filter the values which contains more than 24.99 relative humidity at 3pm.

In [12]:

```
clean_data = data.copy()
clean_data['high_humidity_label'] = (clean_data['relative_humidity_3pm'] >24.99) *1
clean_data['high_humidity_label'].head()
```

Out[12]:

```
0    1
1    0
2    0
3    0
4    1
Name: high_humidity_label, dtype: int64
```

In [13]:

```
y = clean_data[['high_humidity_label']].copy()
y.head()
```

Out[13]:

	high_humidity_label
0	1
1	0
2	0
3	0
4	1

In [14]:

```
clean_data['relative_humidity_3pm'].head()
```

Out[14]:

```
0    36.160000
1    19.426597
2    14.460000
3    12.742547
4    76.740000
Name: relative_humidity_3pm, dtype: float64
```

In [15]:

```
y.head()
```

Out[15]:

	high_humidity_label
0	1
1	0
2	0
3	0
4	1

Use 9am Sensor Signals as Features to Predict Humidity at 3pm

Storing all the Morning features other than Humidity at 3 pm in the morning feature

In [16]:

```
morning_features = ['air_pressure_9am', 'air_temp_9am', 'avg_wind_direction_9am',
       'avg_wind_speed_9am', 'max_wind_direction_9am', 'max_wind_speed_9am',
       'rain_accumulation_9am', 'rain_duration_9am', 'relative_humidity_9am']
```

Copying the values from the clean_data dataset to new dataset x which only consist of the Morning Feature Data

In [17]:

```
x=clean_data[morning_features].copy()
x.columns
```

Out[17]:

```
Index(['air_pressure_9am', 'air_temp_9am', 'avg_wind_direction_9am',
       'avg_wind_speed_9am', 'max_wind_direction_9am', 'max_wind_speed_9a
m',
       'rain_accumulation_9am', 'rain_duration_9am', 'relative_humidity_9a
m'],
      dtype='object')
```

In [18]:

```
y.columns
```

Out[18]:

```
Index(['high_humidity_label'], dtype='object')
```

Spliting the Dataset

By using train_test_split we have split the data into traing dataset and testing datasets.

In [20]:

```
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.33,random_state=324)
```

3. Support Vector Machine (SVM)

- “Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression
- In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.
- Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.
- The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem.
- It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

Fit on Train Set

In [21]:

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:760: Da
taConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using rav
el().
  y = column_or_1d(y, warn=True)
```

Out[21]:

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linea
r',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Making Predictions

In [22]:

```
y_pred = svclassifier.predict(X_test)
```

Evaluating the algorithm - Confusion matrix, precision, recall, and F1 measures are the most commonly used metrics for classification tasks. Scikit-Learn's metrics library contains the classification_report and confusion_matrix methods, which can be readily used to find out the values for these important metrics.

In [23]:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[162  13]
 [ 22 155]]
              precision    recall  f1-score   support

           0       0.88      0.93      0.90       175
           1       0.92      0.88      0.90       177

    accuracy                           0.90       352
   macro avg       0.90      0.90      0.90       352
weighted avg       0.90      0.90      0.90       352
```

Measure Accuracy of the Classifier

In [24]:

```
print(accuracy_score(y_test,y_pred)*100)
```

```
90.05681818181817
```

4. Naive Bayes Classifier

- It is a classification technique based on Bayes’ Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.
- Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as ‘Naive’.
- Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.
- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.

In [25]:

```
from sklearn.naive_bayes import GaussianNB

clf = GaussianNB()
clf.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/naive_bayes.py:206: DataCon
versionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[25]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [26]:

```
y_pred1 = clf.predict(X_test)
```

Evalutaing the algorithm-

In [27]:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred1))
print(classification_report(y_test,y_pred1))
```

```
[[171   4]
 [ 73 104]]
              precision    recall  f1-score   support

           0       0.70      0.98      0.82       175
           1       0.96      0.59      0.73       177

    accuracy                           0.78       352
   macro avg       0.83      0.78      0.77       352
weighted avg       0.83      0.78      0.77       352
```

Measure Accuracy of the Classifier

In [28]:

```
print(accuracy_score(y_test,y_pred1)*100)
```

```
78.125
```

5. Decision Tree

- Decision tree algorithm falls under the category of supervised learning.
- They can be used to solve both regression and classification problems.
- Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.
- We can represent any boolean function on discrete attributes using the decision tree.

We have made a classifier for making the Decision Tree and to train the data using this classifier

In [29]:

```
humidity_classifier = DecisionTreeClassifier(max_leaf_nodes=10,random_state=0)
humidity_classifier.fit(X_train,y_train)
```

Out[29]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=1
0,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

In [30]:

```
type(humidity_classifier)
```

Out[30]:

```
sklearn.tree._classes.DecisionTreeClassifier
```

Predict on Test Set

Using humidity_classifier we have predicted the value for the X_test and stored it to y_predicted

In [31]:

```
y_predicted = humidity_classifier.predict(X_test)
```

In [32]:

```
y_predicted[:10]
```

Out[32]:

```
array([0, 0, 1, 1, 1, 1, 1, 0, 1, 1])
```

In [33]:

```
y_test['high_humidity_label'][:10]
```

Out[33]:

```
456     0
845     0
693     1
259     1
723     1
224     1
300     1
442     0
585     1
1057    1
Name: high_humidity_label, dtype: int64
```

Measure Accuracy of the Classifier

Checking our accuracy of the model using accuracy_score function from sklearn metrics which in this case is with around 90% accuracy

In [34]:

```
accuracy_score(y_test,y_predicted)*100
```

Out[34]:

```
90.05681818181817
```