WEATHER PREDICTION

Problem Statement : Weather Condition analysis , The challenge is aimed at making use of Classical Machine Learning Algorithms to predict the weather conditions.The weather dataset consists of different parameters like Temperature, Humidity , Wind Speed , etc...which plays a key role in predicting the Weather Conditions.

Scope : Our assumptions are limited to categorical and numeric data and Image dataset is not dealed here.

Major real-life applications : Weather forecasting is one of the key aspects of many small-scale as well as Large-scale industries.It has a greater impact on Agricultural field. Farmers have to make several day-to-day weather-related decisions to get a better yield.Weather information plays an increasingly instrumental role in the evolving field of precision agriculture, an agricultural practice that emphasizes accuracy and control when it comes to growing crops. This has an impact on the construction field as well because the wages of the workers and belongings will be wasted in rainy conditions. Fishing is one of the primary jobs for many people near marine areas, this will notify them about the climatic changes.

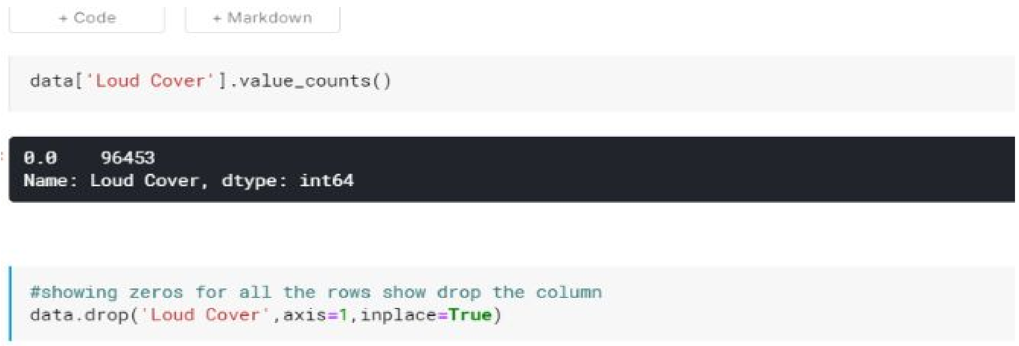
Major challenges : Preprocessing the data is the major task in any machine learning model. So, finding the outliers and replacing them with median values, normalizing the data , converting the categorical data into numerical data, reducing the features are the major tasks.

Source of Data sets: Downloaded the dataset from kaggle and the data has 12 columns (https://www.kaggle.com/muthuj7/weather-dataset) namely Summary,Precip Type, Temperature ,Apparent Temperature,Humidity,Wind Speed (km/h),Wind Bearing (degrees),Visibility,Loud Cover.

Pre-processing :

- **Dropping Unwanted Columns:**

  a. **' Loud Cover' :** This column has zeros in every row , so this will not impact our model.

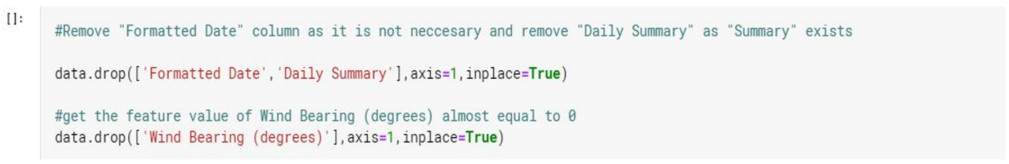


```
data['Loud Cover'].value_counts()
```

```
0.0    96453
Name: Loud Cover, dtype: int64
```

```
#showing zeros for all the rows show drop the column
data.drop('Loud Cover',axis=1,inplace=True)
```

b. **‘Formatted Date’ and ‘Daily summary’ :** Summary column existed twice so remove one and no need for Formatted Date column too.

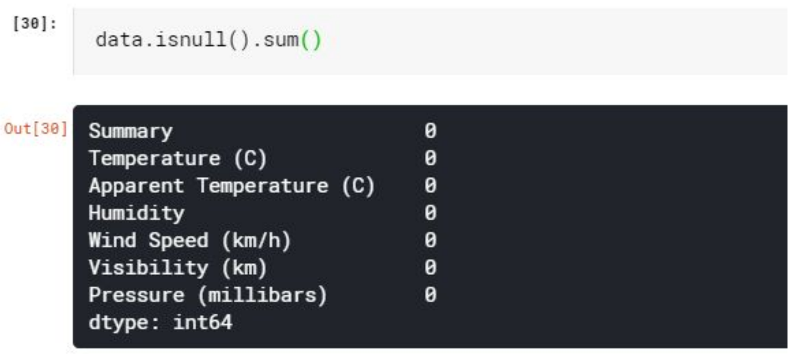


```
[]:  #Remove "Formatted Date" column as it is not neccesary and remove "Daily Summary" as "Summary" exists

     data.drop(['Formatted Date','Daily Summary'],axis=1,inplace=True)

     #get the feature value of Wind Bearing (degrees) almost equal to 0
     data.drop(['Wind Bearing (degrees)'],axis=1,inplace=True)
```

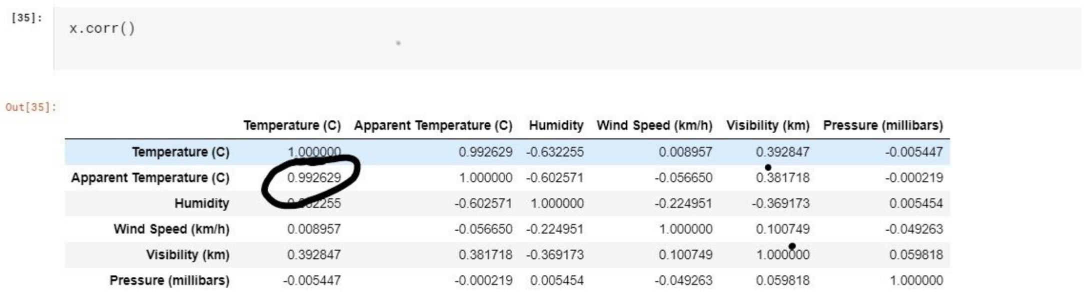
- **Checking for NULL values :** No null values found.



```
[30]:  data.isnull().sum()

Out[30]  Summary                     0
         Temperature (C)             0
         Apparent Temperature (C)    0
         Humidity                    0
         Wind Speed (km/h)           0
         Visibility (km)             0
         Pressure (millibars)        0
         dtype: int64
```

- **Checking Correlation values :** It is observed that correlation value between Temperature and Apparent Temperature is almost equal to one.So, dropped one column

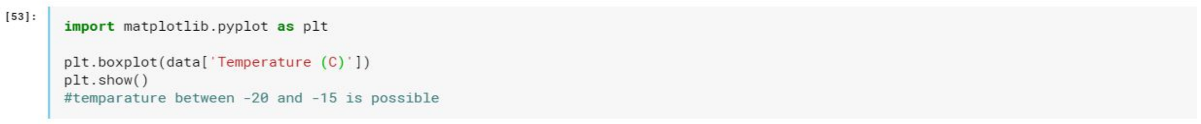


```
[35]:  x.corr()
```

Out[35]:

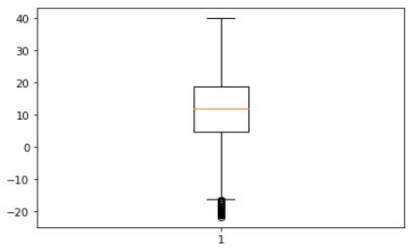
	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Visibility (km)	Pressure (millibars)
Temperature (C)	1.000000	0.992629	-0.632255	0.008957	0.392847	-0.005447
Apparent Temperature (C)	0.992629	1.000000	-0.602571	-0.056650	0.381718	-0.000219
Humidity	-0.632255	-0.602571	1.000000	-0.224951	-0.369173	0.005454
Wind Speed (km/h)	0.008957	-0.056650	-0.224951	1.000000	0.100749	-0.049263
Visibility (km)	0.392847	0.381718	-0.369173	0.100749	1.000000	0.059818
Pressure (millibars)	-0.005447	-0.000219	0.005454	-0.049263	0.059818	1.000000

- **Checking for Outliers:** Outliers existed for almost all columns but those values are possible . For example let’s have the box plot for the Temperature column.

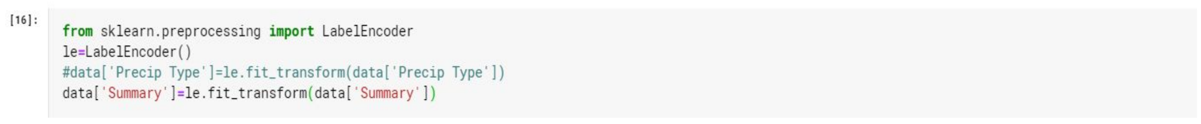


```
[53]:  import matplotlib.pyplot as plt

       plt.boxplot(data['Temperature (C)'])
       plt.show()
       #temparature between -20 and -15 is possible
```



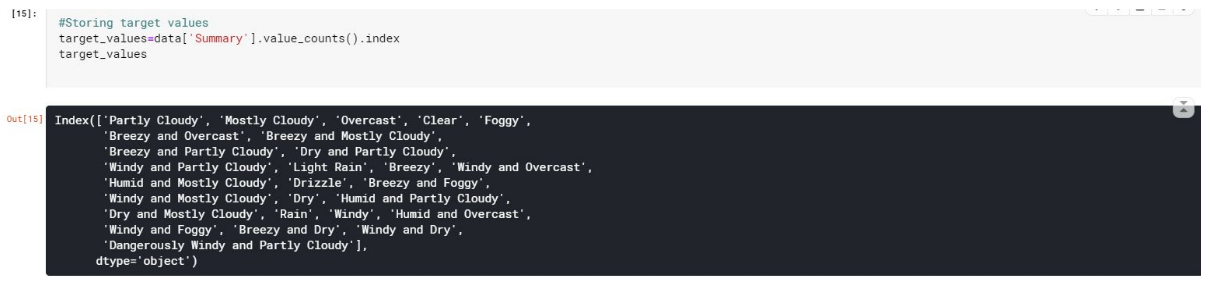
- **Converting Categorical into numerical :** As the categorical values are of string type, in order to classify the Output variable , should convert these categorical values into numerical using pre-processing techniques. Label encoder is one of the encoder used.



```
[16]:
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
#data['Precip Type']=le.fit_transform(data['Precip Type'])
data['Summary']=le.fit_transform(data['Summary'])
```

- **Normalizing the Dataset :** To give equal importance to each and every variable we need to normalize every column

Methodologies : The Summary column which we need to predict has categorical values and the column has 27 different values.



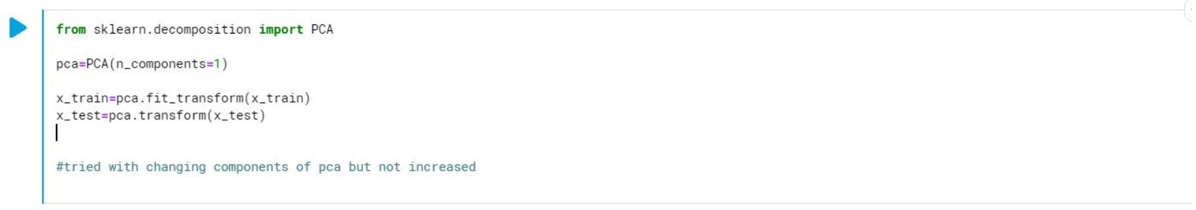
```
[15]:
#Storing target values
target_values=data['Summary'].value_counts().index
target_values
```

```
Out[15]
Index(['Partly Cloudy', 'Mostly Cloudy', 'Overcast', 'Clear', 'Foggy',
       'Breezy and Overcast', 'Breezy and Mostly Cloudy',
       'Breezy and Partly Cloudy', 'Dry and Partly Cloudy',
       'Windy and Partly Cloudy', 'Light Rain', 'Breezy', 'Windy and Overcast',
       'Humid and Mostly Cloudy', 'Drizzle', 'Breezy and Foggy',
       'Windy and Mostly Cloudy', 'Dry', 'Humid and Partly Cloudy',
       'Dry and Mostly Cloudy', 'Rain', 'Windy', 'Humid and Overcast',
       'Windy and Foggy', 'Breezy and Dry', 'Windy and Dry',
       'Dangerously Windy and Partly Cloudy'],
      dtype='object')
```

To classify these categorical values , classifiers are must and should. So, Naive Bayes and Random Forest Classifier are the two classifiers used.

- **Feature Reduction :** The dataset contains around 12 columns and from these 12 columns , one is Dependent variable and remaining are independent variables. In pre-processing of dataset some features are reduced as they are unwanted. At the end of preprocessing we trained both Naive Bayes and Random Forest classifier with the same features.Applied Principle Component Analysis (PCA) by changing the no of components , but no increment in the accuracy.

Using PCA



```
from sklearn.decomposition import PCA

pca=PCA(n_components=1)

x_train=pca.fit_transform(x_train)
x_test=pca.transform(x_test)

#tried with changing components of pca but not increased
```

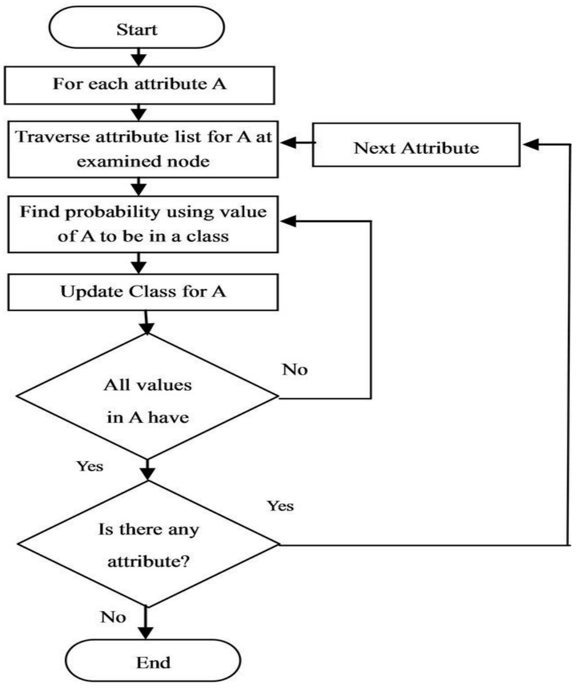
- **Objective function :** Naive Bayes: It is used to find the probability for each class of Y and this is used only for Supervised Classification Problem. This uses Bayes theorem to find the probabilities.

$$P(Y{=}k \mid X1..Xn) = \frac{P(X1 \mid Y{=}k) * P(X2 \mid Y{=}k) \ldots * P(Xn \mid Y{=}k) * P(Y{=}k)}{P(X1) * P(X2) \ldots * P(Xn)}$$

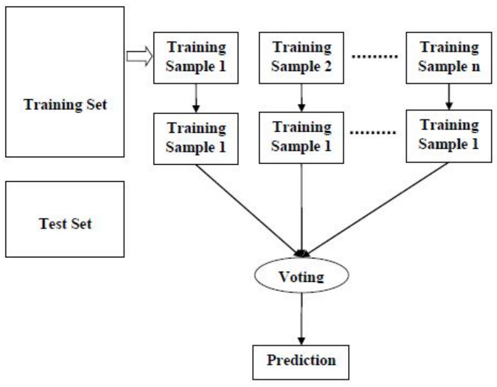
Here k is the one of the classes of Y and x1,x2,x3….Xn are the independent variables or the features of the given classifier problem.

Coming to Random Forest Classifier , it is the voting base technique, where we take votes from multiple decision trees to predict the output. Suppose , there are n decision trees, if the dependent variable is of regressor type then we take the average of the all n decision tree outputs. If the independent variable is of categorical type ,we take the most frequency output among the n decision trees outputs.

Naive Bayes Model :A later likelihood of data instance in class cj of the data model is the concept behind Naïve Bayes' algorithm. The latter likelihood P(ti) is that ti can be called cj. The calculation of P(ti) can be done by multiplying all probabilities of all data instance attributes in the data model: p is indicated as the number of attributes in each data instance. Of all groups, the probability is determined, and the highest probability class will be the label



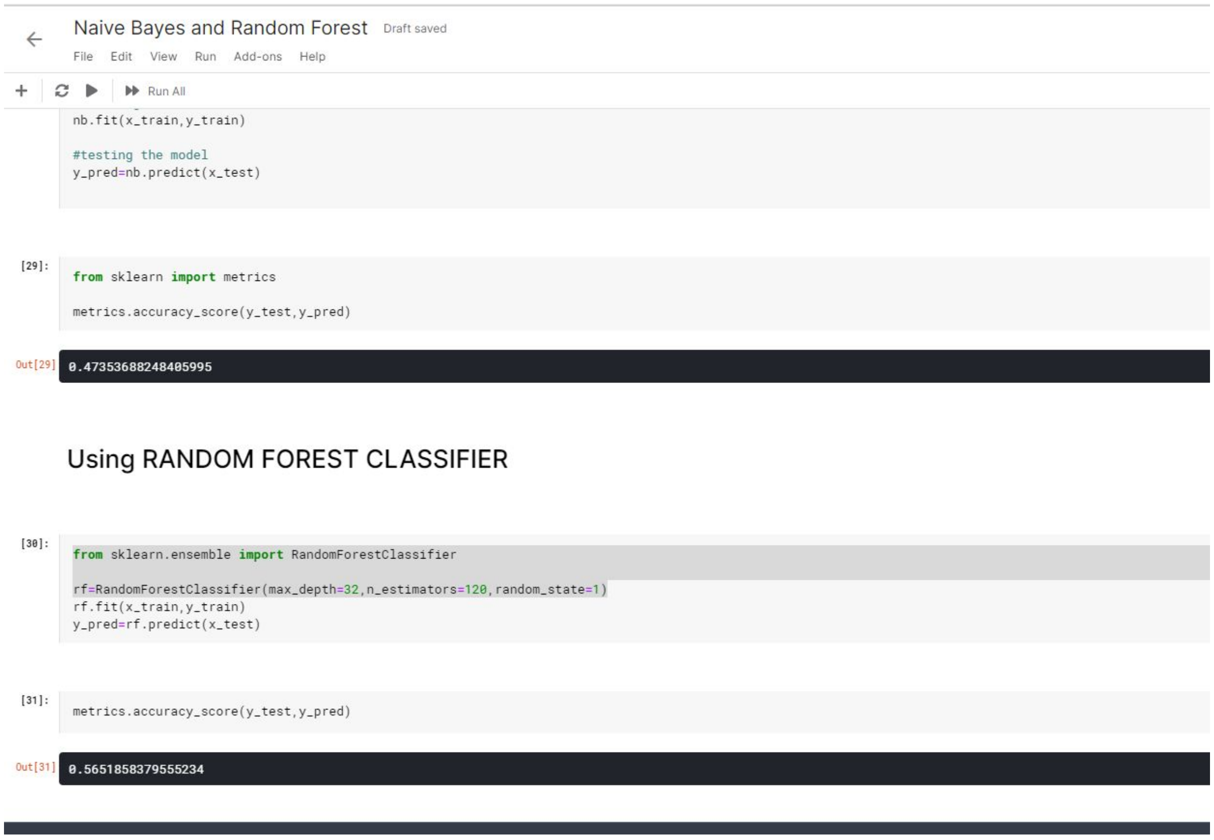
of the case. Figure displays the flowchart of this algorithm.



Random Forest Classifier:Random forest is a supervised learning algorithm used to classify as well as to regress. However, it is used primarily for problems of classification. As we know, a forest consists of trees and more trees, which means stronger forest. Similarly, the algorithm of random forest samples creates decision trees on data samples and then predicts each sample and selects the best solution by voting.It is better than a single decision tree as it reduces the over-a fitting by averaging the outcome.

- **Sensitive Parameters :** For the random forest classifier max_depth , n_estimators are the key parameters.max_depth decides the maximum depth of the trees and n_estimators means the total of no of decision trees to be taken for voting.
  from sklearn.ensemble import RandomForestClassifier
  rf=RandomForestClassifier(max_depth=32,n_estimators=120,random_state=1)

- **Performance Analysis :** There is no doubt in saying that Random Forest is the best Algorithm as it gives highest accuracy , it takes care of sampling, takes care of k-fold algorithm , takes care of null values and mainly it takes care of Over Fitting Results.But when it comes to speed , it depends on the max depth and n_estimators. Generally , Random Forest takes more time to compile.But Naive bayes are compiled in seconds.


Naive Bayes and Random Forest
Draft saved
File Edit View Run Add-ons Help
Run All

```
nb.fit(x_train,y_train)

#testing the model
y_pred=nb.predict(x_test)
```

```
[29]: from sklearn import metrics

      metrics.accuracy_score(y_test,y_pred)

Out[29] 0.47353688248405995
```

Using RANDOM FOREST CLASSIFIER

```
[30]: from sklearn.ensemble import RandomForestClassifier

      rf=RandomForestClassifier(max_depth=32,n_estimators=120,random_state=1)
      rf.fit(x_train,y_train)
      y_pred=rf.predict(x_test)

[31]: metrics.accuracy_score(y_test,y_pred)

Out[31] 0.5651858379555234
```

- **Conclusion** : After preprocessing the data and testing the data with Naive Bayes as well as Random Forest Classifier, we got 47% accuracy with Naive Bayes and 60% accuracy with Random Forest . We also observed that Principle Component analysis has not helped in increasing the accuracy of our model.