AMITY UNIVERSITY
UTTAR PRADESH

CSE 313: Fundamentals of Machine Learning

Group Project

On

Weather Dataset of Manaus

By

Ankita Singh A7605217051
Saurabh Yadav A7605217073
Rabab Jafri A7605217060
Arshita Srivastava A7605217072
Mohit Singh Walia A7605217025

Submitted on
10/4/2020
To
Dr. Pooja Khanna

Amity School of Engineering and Technology

INDEX

Introduction to Machine Learning

Machine Learning is undeniably one of the most influential and powerful technologies in today's world. More importantly, we are far from seeing its full potential. There's no doubt, it will continue to be making headlines for the foreseeable future.

Machine learning is a tool for turning information into knowledge. In the past 50 years, there has been an explosion of data. This mass of data is useless unless we analyse it and find the patterns hidden within. Machine learning techniques are used to automatically find the valuable underlying patterns within complex data that we would otherwise struggle to discover. The hidden patterns and knowledge about a problem can be used to predict future events and perform all kinds of complex decision making.

Types of learning:

1. **Supervised Learning**

Supervised Learning is the one, where you can consider the learning is guided by a teacher. We have a dataset which acts as a teacher and its role is to train the model or the machine. Once the model gets trained it can start making a prediction or decision when new data is given to it.

2. **Unsupervised Learning**

The model learns through observation and finds structures in the data. Once the model is given a dataset, it automatically finds patterns and relationships in the dataset by creating clusters in it. What it cannot do is add labels to the cluster, like it cannot say this a

group of apples or mangoes, but it will separate all the apples from mangoes. Suppose we presented images of apples, bananas and mangoes to the model, so what it does, based on some patterns and relationships it creates clusters and divides the dataset into those clusters. Now if a new data is fed to the model, it adds it to one of the created clusters.

3. Reinforcement learning

It is the ability of an agent to interact with the environment and find out what is the best outcome. It follows the concept of hit and trial method. The agent is rewarded or penalized with a point for a correct or a wrong answer, and on the basis of the positive reward points gained the model trains itself. And again once trained it gets ready to predict the new data presented to it.

Applications of Machine Learning

Mentioned below are few known applications of Machine Learning:

- Online Fraud Detection
- Product Recommendation
- Predictions while computing
- Virtual Personal Assistants
- Email Spam and Malware Filtering
- Search Engine result refining

About Dataset

The weather dataset of Manaus, the capital and largest city of the Brazilian state of Amazona, contains 313 rows and 7 columns. The first column contains the dates of every month. The second column contains the values for cloudiness, the third about Rainfall, fourth about Max temperature on those dates, fifth about the Min temperature on those dates. Sixth column is about humidity and the Seventh column is the Mean temperature.

A	B	C	D	E	F	G
)ate	Cloudiness	Rainfall	Max Temperature	Min Temperature	Humidity	Mean Temperature
31-01-1990	6.591398	234.3	30.525806	23.122581	83.846774	26.208387
28-02-1990	7.571429	190	30.307143	23.114286	83.157407	26.09
31-03-1990	7.043011	299.2	30.548387	23.070968	84.758065	26.055484
30-04-1990	6.640449	236.4	30.343333	22.91	84.836207	26.046897
31-05-1990	6.516129	244.6	31.2	23.33871	82.225806	26.776774
30-06-1990	5.539326	89.3	31.296667	23.236667	79.405172	26.85931
31-07-1990	5.688172	173.9	30.854839	23.032258	75.927419	26.794194
31-08-1990	3.971429	55.4	31.72	22.912	71.521739	27.033913
30-09-1990	4.411111	39	32.536667	23.27	70.358333	27.883333
31-10-1990	5.139785	79.4	32.812903	23.63871	70.064516	27.830968
30-11-1990	4.844444	37.9	33.723333	24.183333	66.733333	28.709333
31-12-1990	7.172043	137.9	32.06129	24.158065	75.491935	27.674839
31-01-1991	7.793478	278.5	29.951613	23.025806	84.5	26.176129
28-02-1991	7.107143	223.8	30.603571	23.335714	83.080357	26.494286
31-03-1991	8.204301	309.2	29.964516	23.119355	87.016129	25.998065
30-04-1991	7.444444	342.9	30.506667	22.79	83.825	26.137333
31-05-1991	6.688172	322.2	30.241935	23.135484	84.104839	26.316774
30-06-1991	6.388889	137.9	30.84	23.546667	80.916667	26.894
31-07-1991	5.688172	173.9	30.854839	23.032258	75.927419	26.794194
31-08-1991	3.971429	55.4	31.72	22.912	71.521739	27.033913
30-09-1991	4.411111	39	32.536667	23.27	70.358333	27.883333
31-10-1991	5.139785	79.4	32.812903	23.63871	70.064516	27.830968
30-11-1991	4.844444	37.9	33.723333	24.183333	66.733333	28.709333
31-12-1991	7.172043	137.9	32.06129	24.158065	75.491935	27.674839
31-01-1992	6.945652	236.5	32.051613	23.880645	77.933333	27.386667
29-02-1992	7.390805	262.7	31.227586	23.544828	78.991379	27.041379

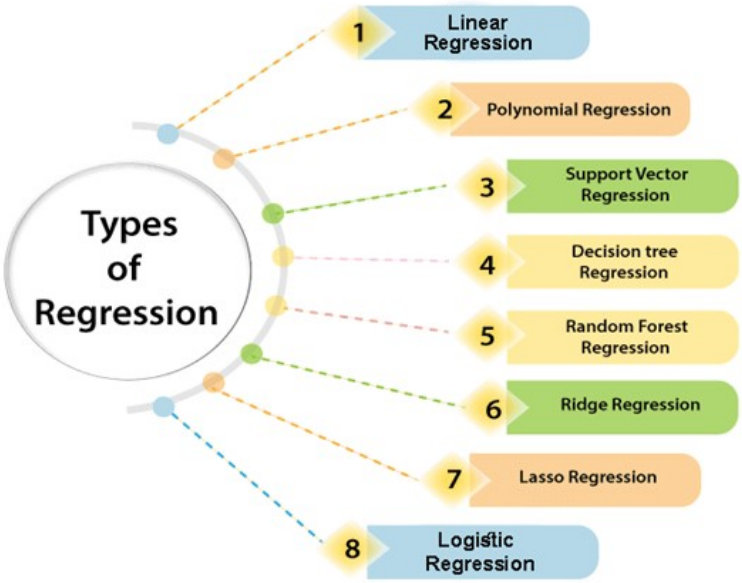
Regression in Machine Learning

In machine learning, regression algorithms attempt to estimate the mapping function (f) from the input variables (x) to numerical or continuous output variables (y).

In this case, y is a real value, which can be an integer or a floating point value. Therefore, regression prediction problems are usually quantities or sizes.

For example, when provided with a dataset about houses, and you are asked to predict their prices, that is a regression task because price will be a continuous output.

Examples of the common regression algorithms include linear regression, Support Vector Regression, and regression trees.



Classification in Machine Learning

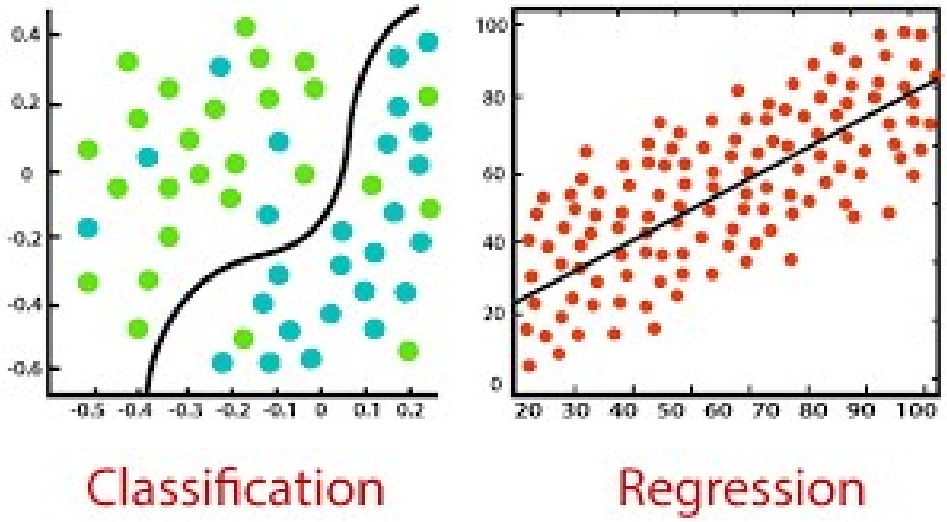
Unlike Regression, classification algorithms attempt to estimate the mapping function (f) from the input variables (x) to discrete or categorical output variables (y).

In this case, y is a category that the mapping function predicts. If provided with a single or several input variables, a classification model will attempt to predict the value of a single or several conclusions.

For example, when provided with a dataset about houses, a classification algorithm can try to predict whether the prices for the houses "sell more or less than the recommended retail price."

Here, the houses will be classified whether their prices fall into two discrete categories: above or below the said price.

Examples of the common classification algorithms include logistic regression, Naïve Bayes, decision trees, and K Nearest Neighbors.



Visualizing our Dataset

Data visualization is a technique that uses an array of static and interactive visuals within a specific context to help people understand and make sense of large amounts of data. The data is often displayed in a story format that visualizes patterns, trends and correlations that may otherwise go unnoticed. Data visualization is regularly used as an avenue to monetize data as a product.

```
In [4]: data.isnull().sum()
```

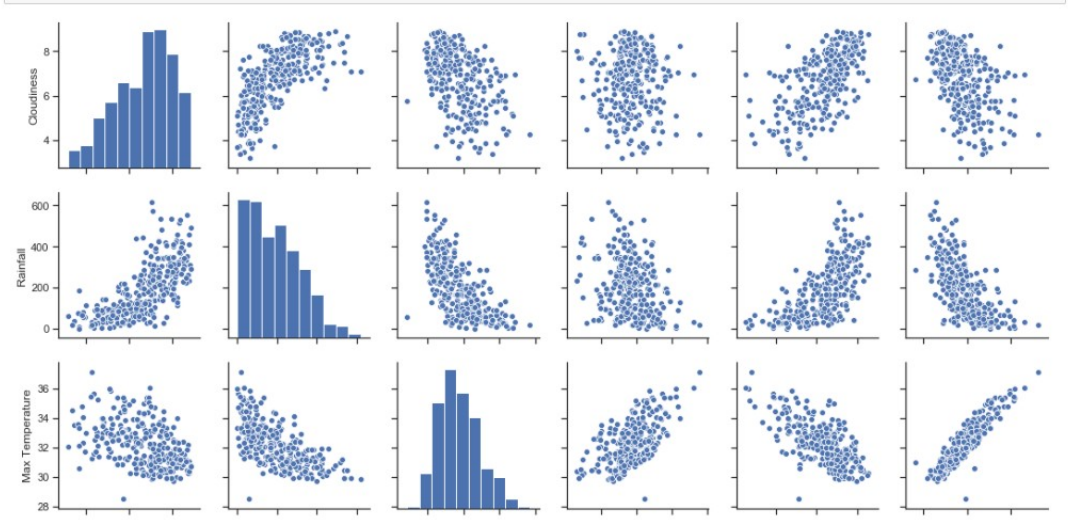
```
Out[4]: Date                0
        Cloudiness          0
        Rainfall            0
        Max Temperature     0
        Min Temperature     0
        Humidity            0
        Mean Temperature    0
        dtype: int64
```

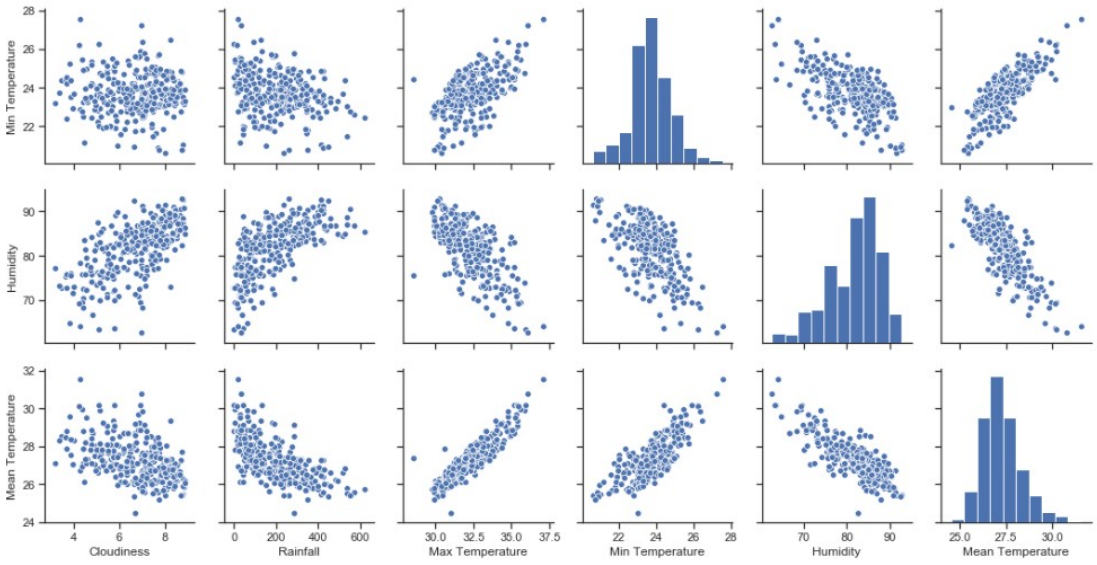
```
In [5]: data.describe()
```

Out[5]:

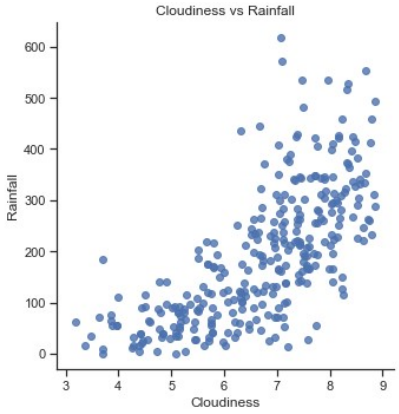
	Cloudiness	Rainfall	Max Temperature	Min Temperature	Humidity	Mean Temperature
count	312.000000	312.000000	312.000000	312.000000	312.000000	312.000000
mean	6.688191	191.636538	32.182255	23.689143	81.645229	27.256696
std	1.323341	130.506865	1.431420	1.060786	6.069194	1.059385
min	3.188889	0.000000	28.546667	20.612903	62.645161	24.516129
25%	5.744624	81.150000	31.143225	23.112419	77.365524	26.551538
50%	6.959677	173.850000	32.011560	23.645860	82.959677	27.104194
75%	7.719433	279.075000	33.068548	24.320000	85.975231	27.841484
max	8.869048	617.400000	37.106667	27.543333	92.767857	31.557931

```
In [6]: sns.set(style="ticks", color_codes=True)
        sns.pairplot(data)
        plt.show()
```

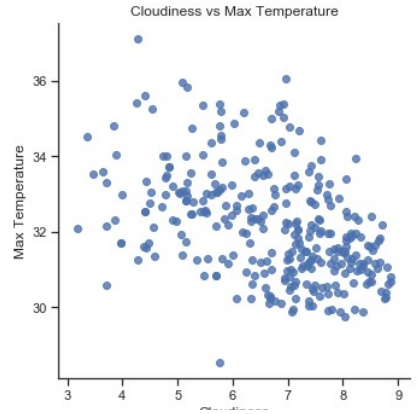




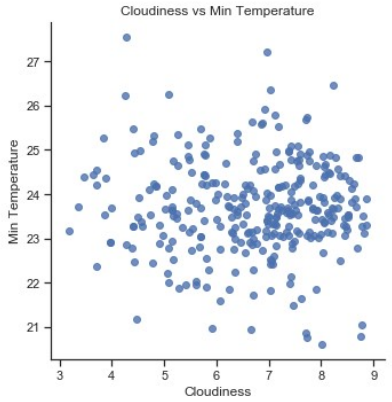
In [7]:
```
sns.lmplot("Cloudiness","Rainfall",data=data,fit_reg=False)
plt.xlabel("Cloudiness")
plt.ylabel("Rainfall")
plt.title("Cloudiness vs Rainfall")
plt.show()
```



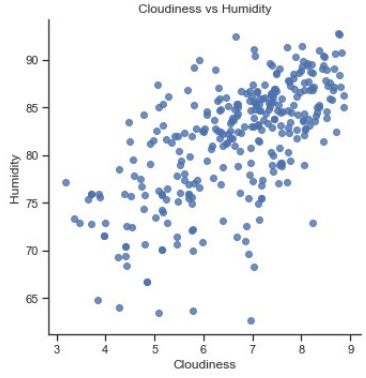
In [8]:
```
sns.lmplot("Cloudiness","Max Temperature",data=data,fit_reg=False)
plt.xlabel("Cloudiness")
plt.ylabel("Max Temperature")
plt.title("Cloudiness vs Max Temperature")
plt.show()
```



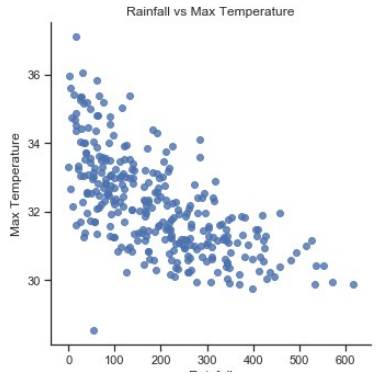
In [9]:
```
sns.lmplot("Cloudiness","Min Temperature",data=data,fit_reg=False)
plt.xlabel("Cloudiness")
plt.ylabel("Min Temperature")
plt.title("Cloudiness vs Min Temperature")
plt.show()
```



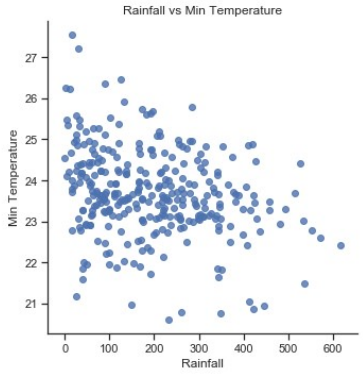
In [10]:
```
sns.lmplot("Cloudiness","Humidity",data=data,fit_reg=False)
plt.xlabel("Cloudiness")
plt.ylabel("Humidity")
plt.title("Cloudiness vs Humidity")
plt.show()
```



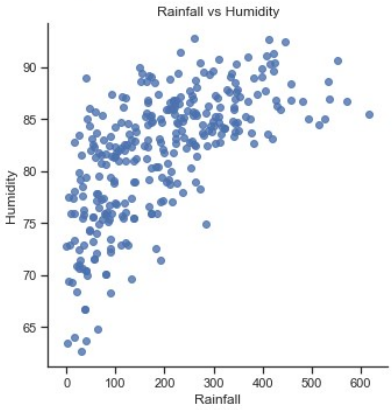
In [11]:
```
sns.lmplot("Rainfall","Max Temperature",data=data,fit_reg=False)
plt.xlabel("Rainfall")
plt.ylabel("Max Temperature")
plt.title("Rainfall vs Max Temperature")
plt.show()
```



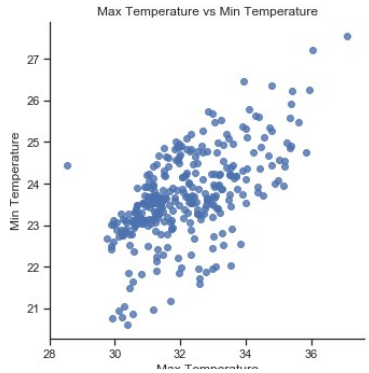
In [12]:
```
sns.lmplot("Rainfall","Min Temperature",data=data,fit_reg=False)
plt.xlabel("Rainfall")
plt.ylabel("Min Temperature")
plt.title("Rainfall vs Min Temperature")
plt.show()
```



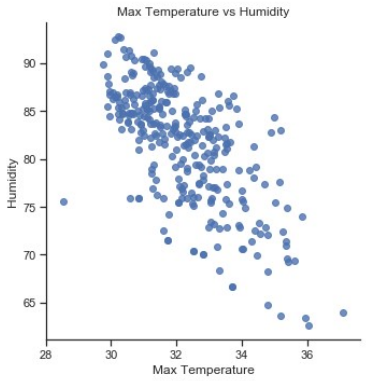
In [13]:
```
sns.lmplot("Rainfall","Humidity",data=data,fit_reg=False)
plt.xlabel("Rainfall")
plt.ylabel("Humidity")
plt.title("Rainfall vs Humidity")
plt.show()
```



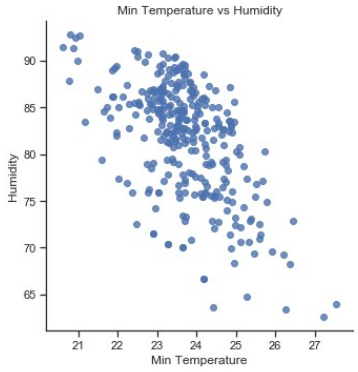
In [14]:
```
sns.lmplot("Max Temperature","Min Temperature",data=data,fit_reg=False)
plt.xlabel("Max Temperature")
plt.ylabel("Min Temperature")
plt.title("Max Temperature vs Min Temperature")
plt.show()
```



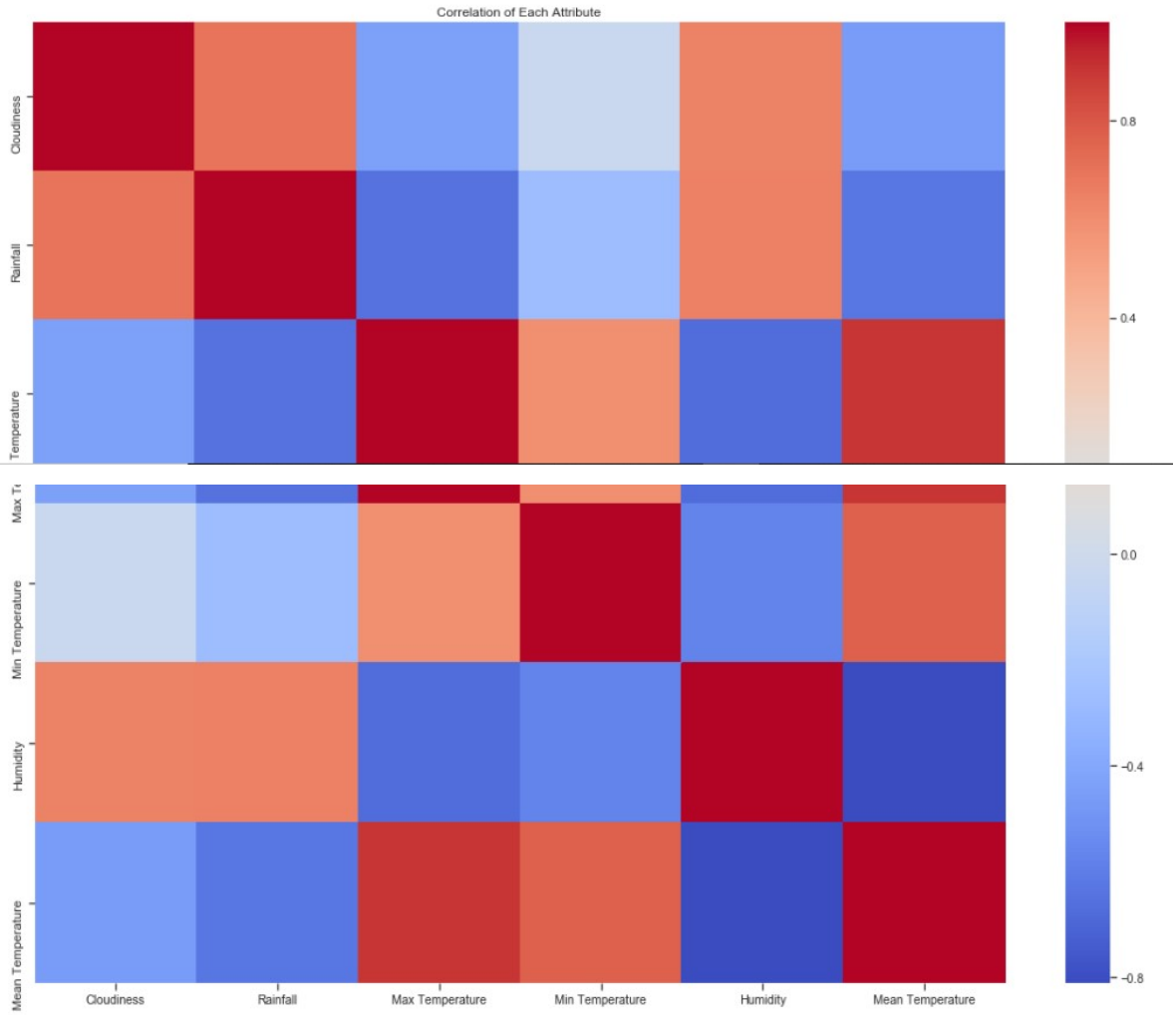
In [15]:
```
sns.lmplot("Max Temperature","Humidity",data=data,fit_reg=False)
plt.xlabel("Max Temperature")
plt.ylabel("Humidity")
plt.title("Max Temperature vs Humidity")
plt.show()
```



In [16]:
```
sns.lmplot("Min Temperature","Humidity",data=data,fit_reg=False)
plt.xlabel("Min Temperature")
plt.ylabel("Humidity")
plt.title("Min Temperature vs Humidity")
plt.show()
```



In [17]:
```
plt.subplots(figsize=(20,15))
ax = plt.axes()
ax.set_title("Correlation of Each Attribute")
corr = data.corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values,
           cmap="coolwarm")
plt.show()
```



Applying Logistic regression

It is also called a logit model .When a liner line does not fit we use the logistic regression. It is the modeling of an event occurring vs the event not occurring. It uses the concept of probability. It models the probability of events like pass or fail , win or lose , healthy or sick etc. It uses a logistic function to model a binary dependent variable. A binary logistic model has a dependent variable with two possible values only where the two variables are labeled as 0 or 1. The dependent variable has two levels (categorical). If more than two outputs they are modeled by multinomial logistic regression. The goal is to find the best fitting model for independent and dependent variable relationship. Independent variables can be continuous or binary. It is used in machine learning widely.
P=1- q , where , p is the probability that event will occur and q is the probability that event does not occur.
Odds= probability of occurring / probability of not occurring.
Logit(P)= ln(p/1-p).

Applying Logistic Regression

In [38]:
```
newdata_LoR=data.iloc[:,1:7]
newdata_LoR.head(1)
```

Out[38]:

	Cloudiness	Rainfall	Max Temperature	Min Temperature	Humidity	Mean Temperature
0	6.591398	234.3	30.525806	23.122581	83.846774	26.208387

In [39]:
```
hmt=data['Mean Temperature'].max()
lmt=data['Mean Temperature'].min()
amt=data['Mean Temperature'].mean()
print('Lowest Mean Temp : ',"{:0.1f}".format(lmt),'Degrees')
print('Average Mean Temp : ',"{:0.1f}".format(amt),'Degrees')
print('Highest Mean Temp : ',"{:0.1f}".format(hmt),'Degrees')
```

```
Lowest Mean Temp :  24.5 Degrees
Average Mean Temp :  27.3 Degrees
Highest Mean Temp :  31.6 Degrees
```

In [40]:
```
newdata_LoR.loc[(newdata_LoR['Mean Temperature']>=24.5)&(newdata_LoR['Mean Temperature']<26.3),'Mean Temperature']=25
newdata_LoR.loc[(newdata_LoR['Mean Temperature']>=26.3)&(newdata_LoR['Mean Temperature']<28.3),'Mean Temperature']=27
newdata_LoR.loc[(newdata_LoR['Mean Temperature']>=28.3)&(newdata_LoR['Mean Temperature']<=32.6),'Mean Temperature']=29
```

In [41]:
```
newdata_LoR['Mean Temperature'].value_counts()
```

Out[41]:
```
27.0    212
25.0     53
29.0     47
Name: Mean Temperature, dtype: int64
```

```
In [42]: X_LoR=np.asanyarray(newdata_LoR.iloc[:,0:5])
         Y_LoR=np.asanyarray(newdata_LoR.iloc[:,5])

In [43]: X_train_LoR,X_test_LoR,Y_train_LoR,Y_test_LoR=train_test_split(X_LoR,Y_LoR,test_size=0.3,random_state = 101)

In [44]: from sklearn.linear_model import LogisticRegression

In [45]: trainer_LoR=LogisticRegression(solver='lbfgs')
         learner_LoR=trainer_LoR.fit(X_train_LoR,Y_train_LoR)
         yac_LoR=Y_test_LoR
         yp_LoR=learner_LoR.predict(X_test_LoR)
         acs_LoR=accuracy_score(yac_LoR,yp_LoR)*100

In [46]: acs_LoR
Out[46]: 85.1063829787234

In [47]: jcs_LoR
Out[47]: 85.1063829787234
```

Applying K-Nearest Neighbor Classification Algorithm

***K-Nearest Neighbor Classifier*:**

K-Nearest Neighbor (or KNN) is the simplest classifier which classifies the data based
on the k-Nearest Neighbor class information.

K-Nearest Neighbor is an efficient supervised machine learning algorithm that can be used for both classification and regression. This algorithm is based on the similar things that exist in the closest proximity. KNN captures the idea of similarity or sometimes called distance or closeness. We have learned to find distance between two points (x1, y1) and (x2, y2).

There are many mathematical formulas or methods to calculate the distance between two points and some of most important formulas are as follows:

- Euclidian distance: $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$
- Manhattan distance: $|x_1 - x_2| + |y_1 - y_2|$
- Cosine similarity: $\sin(A, B) = \cos\theta = \frac{A.B}{\|A\|\|B\|}$

Steps to Perform KNN Algorithm:

1) Load the Crop Production in India Dataset.
2) Preprocess data if required.
3) Initialize k with your chosen nearest neighbor and vary it for **Cross Validation** Purpose.

4) Import **KNN** algorithm from **scikit-learn** Library.
5) Chosen Input Feature (**District, Season, Area**)
6) Split data into train and test dataset.
7) Fit the training data in to model.
8) After completion of training predict the class from test dataset.
9) Find accuracy from actual class over predicted class.
10) Save the model for further calculations.

Applying KNN

In [18]:
```
#data.drop(['Date'],axis=1,inplace=True) #single line statement to remove the date column permanently for the running instace
newdata_K=data.iloc[:,1:7]   #extracting the required part of data from the main data set
```

In [19]:
```
newdata_K.head(1)
```

Out[19]:

	Cloudiness	Rainfall	Max Temperature	Min Temperature	Humidity	Mean Temperature
0	6.591398	234.3	30.525806	23.122581	83.846774	26.208387

In [20]:
```
hmt=data['Mean Temperature'].max()
lmt=data['Mean Temperature'].min()
amt=data['Mean Temperature'].mean()
```

In [21]:
```
print('Lowest Mean Temp : ',"{:0.1f}".format(lmt),'Degrees')
print('Average Mean Temp : ',"{:0.1f}".format(amt),'Degrees')
print('Highest Mean Temp : ',"{:0.1f}".format(hmt),'Degrees')
```

```
Lowest Mean Temp :  24.5 Degrees
Average Mean Temp :  27.3 Degrees
Highest Mean Temp :  31.6 Degrees
```

In [22]:
```
newdata_K.loc[(newdata_K['Mean Temperature']>=24.5)&(newdata_K['Mean Temperature']<26.3),'Mean Temperature']=25
newdata_K.loc[(newdata_K['Mean Temperature']>=26.3)&(newdata_K['Mean Temperature']<28.3),'Mean Temperature']=27
newdata_K.loc[(newdata_K['Mean Temperature']>=28.3)&(newdata_K['Mean Temperature']<=32.6),'Mean Temperature']=29
```

In [23]:
```
newdata_K['Mean Temperature'].value_counts()
```

Out[23]:
```
27.0    212
25.0     53
29.0     47
Name: Mean Temperature, dtype: int64
```

In [24]:
```
X_K=np.asanyarray(newdata_K.iloc[:,0:5])
Y_K=np.asanyarray(newdata_K.iloc[:,5])
```

In [25]:
```
X_train_K,X_test_K,Y_train_K,Y_test_K=train_test_split(X_K,Y_K,test_size=0.3,random_state = 101)
```

In [26]:
```
from sklearn.neighbors import KNeighborsClassifier
k=18
trainer_K=KNeighborsClassifier(n_neighbors=k)
learner_K=trainer_K.fit(X_train_K,Y_train_K)
yac=Y_test_K
yp=learner_K.predict(X_test_K)
acs_K=accuracy_score(yac,yp)*100
jcs_K=jaccard_similarity_score(yac,yp)*100
```

In [27]:
```
acs_K
```

Out[27]:
```
73.40425531914893
```

In [28]:
```
jcs_K
```

Out[28]:
```
73.40425531914893
```

Applying Polynomial Regression

If the data set used for observation does not fit a straight line then the data is more suitable for polynomial regression and polynomial regression fits it more.
Polynomial regression like the linear regression also uses two variable i.e. x and y to find the best possible matching line to pass the data set points.

Polynomial Regression

In [72]:
```
newdata_PR=data.iloc[:,1:7]
newdata_PR.head()
```

Out[72]:

	Cloudiness	Rainfall	Max Temperature	Min Temperature	Humidity	Mean Temperature
0	6.591398	234.3	30.525806	23.122581	83.846774	26.208387
1	7.571429	190.0	30.307143	23.114286	83.157407	26.090000
2	7.043011	299.2	30.548387	23.070968	84.758065	26.055484
3	6.640449	236.4	30.343333	22.910000	84.836207	26.046897
4	6.516129	244.6	31.200000	23.338710	82.225806	26.776774

In [73]:
```
#X_PR=np.asanyarray(newdata_PR.iloc[:,0:5])
#Y_PR=np.asanyarray(newdata_PR.iloc[:,5])
X_PR = np.squeeze(np.asanyarray(newdata_PR.iloc[:,0:5]))

Y_PR = np.squeeze(np.asanyarray(newdata_PR.iloc[:,5]))
```

In [74]:
```
m,n=np.shape(X_PR)
Y_PR=Y_PR.reshape(m,1)
X_PR=np.c_[np.ones((m,1)),X_PR]
theta=np.zeros((n+1,1))
```

In [75]:
```
X_train_PR,X_test_PR,Y_train_PR,Y_test_PR=train_test_split(X_PR,Y_PR,test_size=0.3,random_state = 101)
```

In [76]:
```
from sklearn.preprocessing import PolynomialFeatures
```

In [77]:
```
trainer_PR = PolynomialFeatures(degree = 4)
learner_PR = trainer_PR.fit_transform(X_PR)
#trainer_PR.fit_transform(X_train_PR,Y_train_PR)
learner_PR_LR = LinearRegression()
learner_PR_LR.fit(learner_PR,Y_PR)
yp_PR=learner_PR_LR.predict(learner_PR)
yac_PR=Y_PR
```

In [78]:
```
print('Mean Absolute Error:', mean_absolute_error(yac_PR,yp_PR)*100)
print('Mean Squared Error:', mean_squared_error(yac_PR,yp_PR)*100)
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(yac_PR,yp_PR))*100)
```

```
Mean Absolute Error: 10.280397898036748
Mean Squared Error: 2.6304524441347112
Root Mean Squared Error: 16.218669625264308
```

Applying Lasso Regression

Lasso regression stands for Least Absolute Shrinkage and Selection Operator .The lasso regression is an example of regression i.e. regularized regression.
Regularization of data is done to overcome the problem of overfitting by adding additional information and thereby shrinking the values of the parameter of the data set or model to induce a penalty against complexity.

Lasso Regression

In [89]:
```
newdata_LaR=data.iloc[:,1:7]
newdata_LaR.head()
```

Out[89]:

	Cloudiness	Rainfall	Max Temperature	Min Temperature	Humidity	Mean Temperature
0	6.591398	234.3	30.525806	23.122581	83.846774	26.208387
1	7.571429	190.0	30.307143	23.114286	83.157407	26.090000
2	7.043011	299.2	30.548387	23.070968	84.758065	26.055484
3	6.640449	236.4	30.343333	22.910000	84.836207	26.046897
4	6.516129	244.6	31.200000	23.338710	82.225806	26.776774

In [90]:
```
X_LaR = np.squeeze(np.asanyarray(newdata_LaR.iloc[:,0:5]))

Y_LaR = np.squeeze(np.asanyarray(newdata_LaR.iloc[:,5]))
```

In [91]:
```
X_train_LaR,X_test_LaR,Y_train_LaR,Y_test_LaR=train_test_split(X_LaR,Y_LaR,test_size=0.3,random_state = 101)
```

In [92]:
```
from sklearn.linear_model import Lasso
```

In [93]:
```
lasso = Lasso(alpha = 1)
lasso.fit(X_train_LaR, Y_train_LaR)
yac_LaR = Y_test_LaR
yp_LaR = lasso.predict(X_test_LaR)
print('Mean Absolute Error:', mean_absolute_error(yac_LaR,yp_LaR)*100)
print('Mean Squared Error:', mean_squared_error(yac_LaR,yp_LaR)*100)
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(yac_LaR,yp_LaR))*100)
```

```
Mean Absolute Error: 45.84582460921266
Mean Squared Error: 32.1427678675741
Root Mean Squared Error: 56.694592217930364
```

Applying Decision Tree Algorithm

Decision tree is a part of supervised machine learning i.e. what the output corresponding to a given input in the training set
The data in decision tree is continuously split into further data according to a particular input or parameter. The leaves of the decision tree are called as decisions or final result.

Applying Decision Tree

In [48]:
```
newdata_DT=data.iloc[:,1:7]
newdata_DT.head(1)
```

Out[48]:

	Cloudiness	Rainfall	Max Temperature	Min Temperature	Humidity	Mean Temperature
0	6.591398	234.3	30.525806	23.122581	83.846774	26.208387

In [49]:
```
hmt=data['Mean Temperature'].max()
lmt=data['Mean Temperature'].min()
amt=data['Mean Temperature'].mean()
print('Lowest Mean Temp : ',"{:0.1f}".format(lmt),'Degrees')
print('Average Mean Temp : ',"{:0.1f}".format(amt),'Degrees')
print('Highest Mean Temp : ',"{:0.1f}".format(hmt),'Degrees')
```

```
Lowest Mean Temp :  24.5 Degrees
Average Mean Temp :  27.3 Degrees
Highest Mean Temp :  31.6 Degrees
```

In [50]:
```
newdata_DT.loc[(newdata_DT['Mean Temperature']>=24.5)&(newdata_DT['Mean Temperature']<26.3),'Mean Temperature']=25
newdata_DT.loc[(newdata_DT['Mean Temperature']>=26.3)&(newdata_DT['Mean Temperature']<28.3),'Mean Temperature']=27
newdata_DT.loc[(newdata_DT['Mean Temperature']>=28.3)&(newdata_DT['Mean Temperature']<=32.6),'Mean Temperature']=29
```

In [51]:
```
newdata_DT['Mean Temperature'].value_counts()
```

Out[51]:
```
27.0    212
25.0     53
29.0     47
Name: Mean Temperature, dtype: int64
```

In [52]:
```
X_DT=np.asanyarray(newdata_DT.iloc[:,0:5])
Y_DT=np.asanyarray(newdata_DT.iloc[:,5])
```

In [53]:
```
X_train_DT,X_test_DT,Y_train_DT,Y_test_DT=train_test_split(X_DT,Y_DT,test_size=0.3,random_state = 101)
```

In [54]:
```
from sklearn.tree import DecisionTreeClassifier
trainer_DT = DecisionTreeClassifier(criterion = "gini",
            random_state = 101,max_depth=3, min_samples_leaf=5)
learner_DT=trainer_DT.fit(X_train_DT,Y_train_DT)
yac_DT=Y_test_DT
yp_DT=learner_DT.predict(X_test_DT)
acs_DT=accuracy_score(yac_DT,yp_DT)*100
jcs_DT=jaccard_similarity_score(yac_DT,yp_DT)*100
```

In [55]:
```
acs_DT
```

Out[55]: 85.1063829787234

In [56]:
```
jcs_DT
```

Out[56]: 85.1063829787234

Applying Support vector machine

SVM or Support Vector Machine **is** a linear model for **classification** and regression problems. It **can** solve linear and non-linear problems and work well for many practical problems. The idea of **SVM is** simple: The algorithm creates a line or a hyperplane which separates the data into classes. It lies in the supervised learning algorithm category. It is majorly used for classification problems. Objective of a support vector machine algorithm is to find a hyperplane in N dimensional space where N is the number of features that classifies the data points distinctly. It creates the best line or the decision boundary to segregate two different classes so it becomes easy for us to put the new data in its corresponding class category. There can be various hyperplane that could be chosen to separate two classes of given data points. The goal is to find a plane that has the maximum margin that is the maximum distance between data points of both the classes to provide some kind of reinforcement that allows us to classify data with more confidence in the future. Data points that fall on either side of the hyperplane can be attributed to different classes. The dimension of the hyperplane depends upon the number of features that implies that if the number of input features is 2 then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane.

Linear support vector machine is the newest extremely fast machine learning algorithm for solving multiclass classification problems from large data sets that implements an original proprietary version of a cutting plane algorithm for designing a **linear** support vector machine.

Applying Support Vector Machine

In [57]:
```
newdata_SVM=data.iloc[:,1:7]
newdata_SVM.head(1)
```

Out[57]:

	Cloudiness	Rainfall	Max Temperature	Min Temperature	Humidity	Mean Temperature
0	6.591398	234.3	30.525806	23.122581	83.846774	26.208387

In [58]:
```
hmt=data['Mean Temperature'].max()
lmt=data['Mean Temperature'].min()
amt=data['Mean Temperature'].mean()
print('Lowest Mean Temp : ',"{:0.1f}".format(lmt),'Degrees')
print('Average Mean Temp : ',"{:0.1f}".format(amt),'Degrees')
print('Highest Mean Temp : ',"{:0.1f}".format(hmt),'Degrees')
```

```
Lowest Mean Temp :  24.5 Degrees
Average Mean Temp :  27.3 Degrees
Highest Mean Temp :  31.6 Degrees
```

In [59]:
```
newdata_SVM.loc[(newdata_SVM['Mean Temperature']>=24.5)&(newdata_SVM['Mean Temperature']<26.3),'Mean Temperature']=25
newdata_SVM.loc[(newdata_SVM['Mean Temperature']>=26.3)&(newdata_SVM['Mean Temperature']<28.3),'Mean Temperature']=27
newdata_SVM.loc[(newdata_SVM['Mean Temperature']>=28.3)&(newdata_SVM['Mean Temperature']<=32.6),'Mean Temperature']=29
```

In [60]:
```
newdata_SVM['Mean Temperature'].value_counts()
```

Out[60]:
```
27.0    212
25.0     53
29.0     47
Name: Mean Temperature, dtype: int64
```

In [61]:
```
X_SVM=np.asanyarray(newdata_DT.iloc[:,0:5])
Y_SVM=np.asanyarray(newdata_DT.iloc[:,5])
```

In [62]:
```
X_train_SVM,X_test_SVM,Y_train_SVM,Y_test_SVM=train_test_split(X_SVM,Y_SVM,test_size=0.3,random_state = 101)
```

In [63]:
```
from sklearn.svm import SVC
trainer_SVM=SVC(kernel='linear')
learner_SVM=trainer_SVM.fit(X_train_SVM,Y_train_SVM)
yac_SVM=Y_test_SVM
yp_SVM=learner_SVM.predict(X_test_SVM)
acs_SVM=accuracy_score(yac_SVM,yp_SVM)*100
jcs_SVM=jaccard_similarity_score(yac_SVM,yp_SVM)*100
```

In [64]:
```
acs_SVM
```

Out[64]:
```
91.48936170212765
```

In [65]:
```
jcs_SVM
```

Out[65]:
```
91.48936170212765
```

Applying Liner regression

This involves finding the best fit line of 2 variables so that one can be used to predict the other. It is a commonly used predictive analysis. It determines the extend to which there is a liner relationship between dependent and one or more independent variables. When
Y= c +b1.x1+b2.x2+…..bn.xn this means there is one dependent and many independent variables. This is multiple regression it is an extension of linear regression involving more than one predictor variable. Data is fit to a multi dimensional surface.

Applying Linear Regression

In [66]:
```
newdata_LR=data.iloc[:,1:7]
newdata_LR.head()
```

Out[66]:

	Cloudiness	Rainfall	Max Temperature	Min Temperature	Humidity	Mean Temperature
0	6.591398	234.3	30.525806	23.122581	83.846774	26.208387
1	7.571429	190.0	30.307143	23.114286	83.157407	26.090000
2	7.043011	299.2	30.548387	23.070968	84.758065	26.055484
3	6.640449	236.4	30.343333	22.910000	84.836207	26.046897
4	6.516129	244.6	31.200000	23.338710	82.225806	26.776774

In [67]:
```
X_LR=np.asanyarray(newdata_LR.iloc[:,0:5])
Y_LR=np.asanyarray(newdata_LR.iloc[:,5])
```

In [68]:
```
X_train_LR,X_test_LR,Y_train_LR,Y_test_LR=train_test_split(X_LR,Y_LR,test_size=0.3,random_state = 101)
```

In [69]:
```
from sklearn.linear_model import LinearRegression
```

In [70]:
```
trainer_LR=LinearRegression(fit_intercept=True)
learner_LR=trainer_LR.fit(X_train_LR,Y_train_LR)
yac_LR=Y_test_LR
yp_LR=learner_LR.predict(X_test_LR)
```

In [71]:
```
print('Mean Absolute Error:', mean_absolute_error(yac_LR,yp_LR)*100)
print('Mean Squared Error:', mean_squared_error(yac_LR,yp_LR)*100)
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(yac_LR,yp_LR))*100)
```

```
Mean Absolute Error: 14.232875046875348
Mean Squared Error: 3.162547558092906
Root Mean Squared Error: 17.78355295798032
```

Applying Support Vector Regression

Support Vector Regression(SVR) is quite different than other Regression models. It uses the Support Vector Machine(SVM, a classification algorithm) algorithm to predict a continuous variable. While other linear regression models try to minimize the error between the predicted and the actual value, Support Vector Regression tries to fit the best line within a predefined or threshold error value. What SVR does in this sense, it tries to classify all the prediction lines in two types, ones that pass through the error boundary(space separated by two parallel lines) and ones that don't. Those lines which do not pass the error boundary are not considered as the difference between the predicted value and the actual value has exceeded the error threshold, $\boldsymbol{\varepsilon}$(epsilon). The lines that pass, are considered for a potential support vector to predict the value of an unknown.

Support Vector Regressor

In [79]:
```
newdata_SVR=data.iloc[:,1:7]
newdata_SVR.head()
```

Out[79]:

	Cloudiness	Rainfall	Max Temperature	Min Temperature	Humidity	Mean Temperature
0	6.591398	234.3	30.525806	23.122581	83.846774	26.208387
1	7.571429	190.0	30.307143	23.114286	83.157407	26.090000
2	7.043011	299.2	30.548387	23.070968	84.758065	26.055484
3	6.640449	236.4	30.343333	22.910000	84.836207	26.046897
4	6.516129	244.6	31.200000	23.338710	82.225806	26.776774

In [80]:
```
X_SVR = np.squeeze(np.asanyarray(newdata_SVR.iloc[:,0:5]))

Y_SVR = np.squeeze(np.asanyarray(newdata_SVR.iloc[:,5]))
```

In [81]:
```
X_train_SVR,X_test_SVR,Y_train_SVR,Y_test_SVR=train_test_split(X_SVR,Y_SVR,test_size=0.3,random_state = 101)
```

In [82]:
```
from sklearn.svm import SVR
```

In [83]:
```
regressor = SVR(kernel='linear',degree=4)
regressor.fit(X_train_SVR,Y_train_SVR)
yp_SVR = regressor.predict(X_test_SVR)
yac_SVR = Y_test_SVR
print('Mean Absolute Error:', mean_absolute_error(yac_SVR,yp_SVR)*100)
print('Mean Squared Error:', mean_squared_error(yac_SVR,yp_SVR)*100)
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(yac_SVR,yp_SVR))*100)
```

```
Mean Absolute Error: 14.193554789482407
Mean Squared Error: 3.182494338189898
Root Mean Squared Error: 17.839546906213446
```

Applying Naïve Bayes Classifier

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Using Bayes theorem, we can find the probability of **A** happening, given that **B** has occurred. Here, **B** is the evidence and **A** is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naive.

Applying Naive Bayes

In [29]:
```
newdata_NB=data.iloc[:,1:7]
newdata_NB.head(1)
```

Out[29]:

	Cloudiness	Rainfall	Max Temperature	Min Temperature	Humidity	Mean Temperature
0	6.591398	234.3	30.525806	23.122581	83.846774	26.208387

In [30]:
```
hmt=data['Mean Temperature'].max()
lmt=data['Mean Temperature'].min()
amt=data['Mean Temperature'].mean()
print('Lowest Mean Temp : ',"{:0.1f}".format(lmt),'Degrees')
print('Average Mean Temp : ',"{:0.1f}".format(amt),'Degrees')
print('Highest Mean Temp : ',"{:0.1f}".format(hmt),'Degrees')
```

```
Lowest Mean Temp :  24.5 Degrees
Average Mean Temp :  27.3 Degrees
Highest Mean Temp :  31.6 Degrees
```

In [31]:
```
newdata_NB.loc[(newdata_NB['Mean Temperature']>=24.5)&(newdata_NB['Mean Temperature']<26.3),'Mean Temperature']=25
newdata_NB.loc[(newdata_NB['Mean Temperature']>=26.3)&(newdata_NB['Mean Temperature']<28.3),'Mean Temperature']=27
newdata_NB.loc[(newdata_NB['Mean Temperature']>=28.3)&(newdata_NB['Mean Temperature']<=32.6),'Mean Temperature']=29
```

In [32]:
```
newdata_NB['Mean Temperature'].value_counts()
```

Out[32]:
```
27.0    212
25.0     53
29.0     47
Name: Mean Temperature, dtype: int64
```

In [33]:
```
X_NB=np.asanyarray(newdata_NB.iloc[:,0:5])
Y_NB=np.asanyarray(newdata_NB.iloc[:,5])
```

In [34]:
```
X_train_NB,X_test_NB,Y_train_NB,Y_test_NB=train_test_split(X_NB,Y_NB,test_size=0.3,random_state = 101)
```

In [35]:
```
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(X_train_NB, Y_train_NB)
yp_NB= model.predict(X_test_NB)
yac_NB=Y_test_NB
acs_NB=accuracy_score(yac_NB,yp_NB)*100
jcs_NB=jaccard_similarity_score(yac_NB,yp_NB)*100
```

In [36]:
```
acs_NB
```

Out[36]:
```
91.48936170212765
```

In [37]:
```
jcs_NB
```

Out[37]:
```
91.48936170212765
```

Applying Ridge Regression

Tikhonov Regularization, colloquially known as ridge regression, is the most commonly used regression algorithm to approximate an answer for an equation with no unique solution. This type of problem is very common in machine learning tasks, where the "best" solution must be chosen using limited data.

Ridge Regression

In [84]:
```
newdata_RR=data.iloc[:,1:7]
newdata_RR.head()
```

Out[84]:

	Cloudiness	Rainfall	Max Temperature	Min Temperature	Humidity	Mean Temperature
0	6.591398	234.3	30.525806	23.122581	83.846774	26.208387
1	7.571429	190.0	30.307143	23.114286	83.157407	26.090000
2	7.043011	299.2	30.548387	23.070968	84.758065	26.055484
3	6.640449	236.4	30.343333	22.910000	84.836207	26.046897
4	6.516129	244.6	31.200000	23.338710	82.225806	26.776774

In [85]:
```
X_RR = np.squeeze(np.asanyarray(newdata_RR.iloc[:,0:5]))

Y_RR = np.squeeze(np.asanyarray(newdata_RR.iloc[:,5]))
```

In [86]:
```
X_train_RR,X_test_RR,Y_train_RR,Y_test_RR=train_test_split(X_RR,Y_RR,test_size=0.3,random_state = 101)
```

In [87]:
```
from sklearn.linear_model import Ridge
```

In [88]:
```
model = Ridge(alpha = 0.5, normalize = False, tol = 0.001,solver ='auto', random_state = 42)
model.fit(X_train_RR, Y_train_RR)
Ridge(alpha=0.5, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=42, solver='auto', tol=0.001)
yac_RR = Y_test_RR
yp_RR = model.predict(X_test_RR)
print('Mean Absolute Error:', mean_absolute_error(yac_RR,yp_RR)*100)
print('Mean Squared Error:', mean_squared_error(yac_RR,yp_RR)*100)
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(yac_RR,yp_RR))*100)
```

```
Mean Absolute Error: 14.25162789035879
Mean Squared Error: 3.1664128943602923
Root Mean Squared Error: 17.794417367141563
```

Results and Comparison

Algorithm used	Mean Absolute error	Mean Squared Error	Root Mean Square Error
Linear Regression	14.232875046875348	3.162547558092	17.783552957980
Polynomial Regression	10.2803978980367	2.630452444134	16.218669625264
Lasso Regression	45.8458246092126	32.1427678675	56.694592217930
Ridge Regression	14.25162789035	3.166412894360	17.794417367141
SVR	14.19355478948240	3.182494338189	17.83954690921344

Algorithm used	Accuracy Score
KNN	73.40425531914893
Naive Bayes	91.48936170212765
Decision Tree	85.1063829787234
SVM	85.1063829787234