Unsupported Cell Type. Double-Click to inspect/edit the content.

Weather Forcasting

```
import pandas as pd #importing panda
import numpy as np #importing numpy
import seaborn as sns #importing seaborn
from numpy import math #This module provides access to the mathematical functions
```

```
datasets=pd.read_csv("C:\\Users\\Somu\\Downloads\\WEATHER FORCASTING (Project).csv") #Read a comma-separated values (csv) fi
X = datasets.iloc[:, :-1].values # : - Represent all the rows , :-1 - Taking all the columns from (n-1),-2
Y = datasets.iloc[:, 4].values # : - Represent all the rows  , 4 - upto 5th column
datasets
```

	Temperature (C)	Humidity	Visibility (km)	Precip	Wind Speed (km/h)	Loud Cover	Apparent Temperature (C)
0	9.472222	0.89	15.8263	rain	14.1197	0	7.388889
1	9.355556	0.86	15.8263	rain	14.2646	0	7.227778
2	9.377778	0.89	14.9569	rain	3.9284	0	9.377778
3	8.288889	0.83	15.8263	rain	14.1036	0	5.944444
4	8.755556	0.83	15.8263	rain	11.0446	0	6.977778
...	...	...	...	...	...	...	...
94	7.827778	0.72	15.8263	rain	13.8943	0	5.405556
95	7.855556	0.72	15.0052	rain	9.8049	0	6.122222
96	7.316667	0.75	15.8746	rain	6.6654	0	6.211111
97	7.244444	0.75	15.8746	rain	7.1162	0	6.005556
98	5.438889	0.88	9.9820	rain	3.7191	0	5.438889

99 rows × 7 columns

```
#To view top 10 Weather records
datasets.head(10)
```

	Temperature (C)	Humidity	Visibility (km)	Precip	Wind Speed (km/h)	Loud Cover	Apparent Temperature (C)
0	9.472222	0.89	15.8263	rain	14.1197	0	7.388889
1	9.355556	0.86	15.8263	rain	14.2646	0	7.227778
2	9.377778	0.89	14.9569	rain	3.9284	0	9.377778
3	8.288889	0.83	15.8263	rain	14.1036	0	5.944444
4	8.755556	0.83	15.8263	rain	11.0446	0	6.977778
5	9.222222	0.85	14.9569	rain	13.9587	0	7.111111
6	7.733333	0.95	9.9820	rain	12.3648	0	5.522222
7	8.772222	0.89	9.9820	rain	14.1519	0	6.527778
8	10.822222	0.82	9.9820	rain	11.3183	0	10.822222
9	13.772222	0.72	9.9820	rain	12.5258	0	13.772222

```
# To select Data by Label
Weather=datasets.loc[25]
Weather
```

```
Temperature (C)             9.91111
Humidity                       0.66
Visibility (km)             15.8263
Precip                         rain
Wind Speed (km/h)           17.2109
Loud Cover                        0
Apparent Temperature (C)    7.56667
Name: 25, dtype: object
```

```
datasets.info() #This method prints information about a DataFrame including the index dtype and columns, non-null values and
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99 entries, 0 to 98
```

```
Data columns (total 7 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Temperature (C)           99 non-null     float64
 1   Humidity                  99 non-null     float64
 2   Visibility (km)           99 non-null     float64
 3   Precip                    99 non-null     object 
 4   Wind Speed (km/h)         99 non-null     float64
 5   Loud Cover                99 non-null     int64  
 6   Apparent Temperature (C)  99 non-null     float64
dtypes: float64(5), int64(1), object(1)
memory usage: 5.5+ KB
```

```
# check datatypes
datasets.dtypes
```

```
Temperature (C)             float64
Humidity                    float64
Visibility (km)             float64
Precip                       object
Wind Speed (km/h)           float64
Loud Cover                    int64
Apparent Temperature (C)    float64
dtype: object
```

```
rows, col=datasets.shape #Return a tuple representing the dimensionality of the DataFrame
print("Rows : %s, column : %s" % (rows,col) ) #Convert a number or string to an integer,  or return 0 if no arguments are gi
```

```
Rows : 99, column : 7
```

```
print(datasets.columns) #Immutable ndarray implementing an ordered, sliceable set. The basic object storing axis labels for 
```

```
Index(['Temperature (C)', 'Humidity', 'Visibility (km)', 'Precip',
       'Wind Speed (km/h)', 'Loud Cover', 'Apparent Temperature (C)'],
      dtype='object')
```

```
#Categorical variables:
categorical = datasets.select_dtypes(include = ["object"]).keys()
print(categorical)
```

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

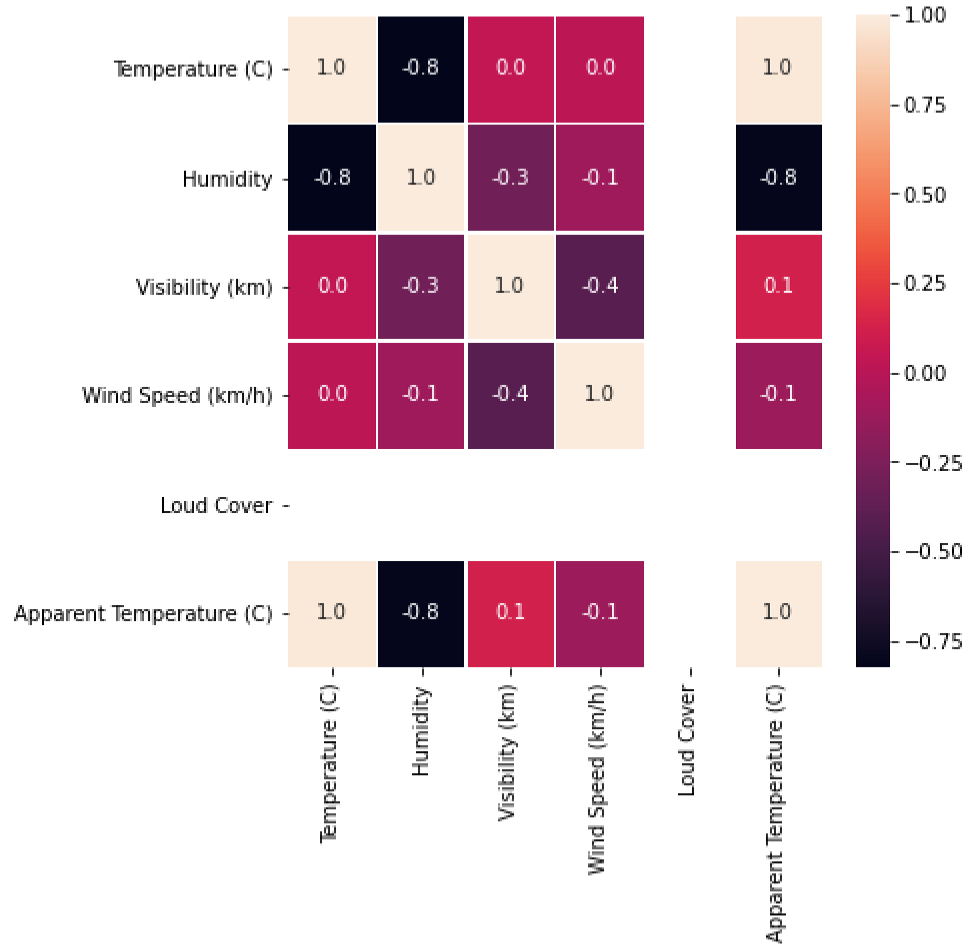
```
#Quantitative variables:
quantitative = datasets.select_dtypes(include = ["int64","float64"]).keys()
print(quantitative)
```

```
Index(['Temperature (C)', 'Humidity', 'Visibility (km)', 'Wind Speed (km/h)',
       'Loud Cover', 'Apparent Temperature (C)'],
      dtype='object')
```

```
corr=datasets.corr() #Compute pairwise correlation of columns, excluding NA/null values.
print(corr)
```

```
                          Temperature (C)  Humidity  Visibility (km)  \
Temperature (C)                  1.000000 -0.825571         0.048935   
Humidity                        -0.825571  1.000000        -0.304906   
Visibility (km)                  0.048935 -0.304906         1.000000   
Wind Speed (km/h)                0.026648 -0.099384        -0.401903   
Loud Cover                            NaN       NaN              NaN   
Apparent Temperature (C)         0.978764 -0.805435         0.125471   

                          Wind Speed (km/h)  Loud Cover  \
Temperature (C)                    0.026648         NaN   
Humidity                          -0.099384         NaN   
Visibility (km)                   -0.401903         NaN   
Wind Speed (km/h)                  1.000000         NaN   
Loud Cover                              NaN         NaN   
Apparent Temperature (C)          -0.112981         NaN   

                          Apparent Temperature (C)  
Temperature (C)                           0.978764  
Humidity                                 -0.805435  
Visibility (km)                           0.125471  
Wind Speed (km/h)                        -0.112981  
Loud Cover                                     NaN  
Apparent Temperature (C)                  1.000000  
```

```
f,ax = plt.subplots(figsize=(6, 6))
sns.heatmap(datasets.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax) ## Thick values take the relationship to identify 
#heatmap - Plot rectangular data as a color-encoded matrix.
plt.show()
```




```
#Quantitative variables. Missing values
datasets[quantitative].describe()
```

	Temperature (C)	Humidity	Visibility (km)	Wind Speed (km/h)	Loud Cover	Apparent Temperature (C)
count	99.000000	99.000000	99.000000	99.000000	99.0	99.000000
mean	11.730135	0.742525	10.866036	16.583651	0.0	10.566835
std	4.223573	0.159542	3.449905	7.658931	0.0	5.395910
min	5.438889	0.360000	2.656500	0.644000	0.0	1.494444
25%	8.200000	0.660000	9.982000	11.117050	0.0	5.652778
50%	10.694444	0.770000	10.851400	15.584800	0.0	10.694444
75%	15.058333	0.860000	12.751200	22.588300	0.0	15.058333
max	21.183333	0.990000	15.874600	32.167800	0.0	21.183333

```
from matplotlib.pyplot import rcParams #A dictionary object including validation.
rcParams['figure.figsize'] = 9, 9 #Validating functions are defined and associated with rc parameters in :mod:`matplotlib.rc
datasets[quantitative].hist()
```

```
array([[<AxesSubplot:title={'center':'Temperature (C)'}>,
        <AxesSubplot:title={'center':'Humidity'}>],
       [<AxesSubplot:title={'center':'Visibility (km)'}>,
        <AxesSubplot:title={'center':'Wind Speed (km/h)'}>],
       [<AxesSubplot:title={'center':'Loud Cover'}>,
```

```
datasets.isnull().any() #Detect missing values.
```

```
Temperature (C)             False
Humidity                    False
Visibility (km)             False
Precip                      False
Wind Speed (km/h)           False
Loud Cover                  False
Apparent Temperature (C)    False
dtype: bool
```

```
#'Loud Cover' takes values zero. We drop it
datasets=datasets.drop('Loud Cover',axis=1) #Drop specified labels from rows or columns.
#Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names. Wh
```

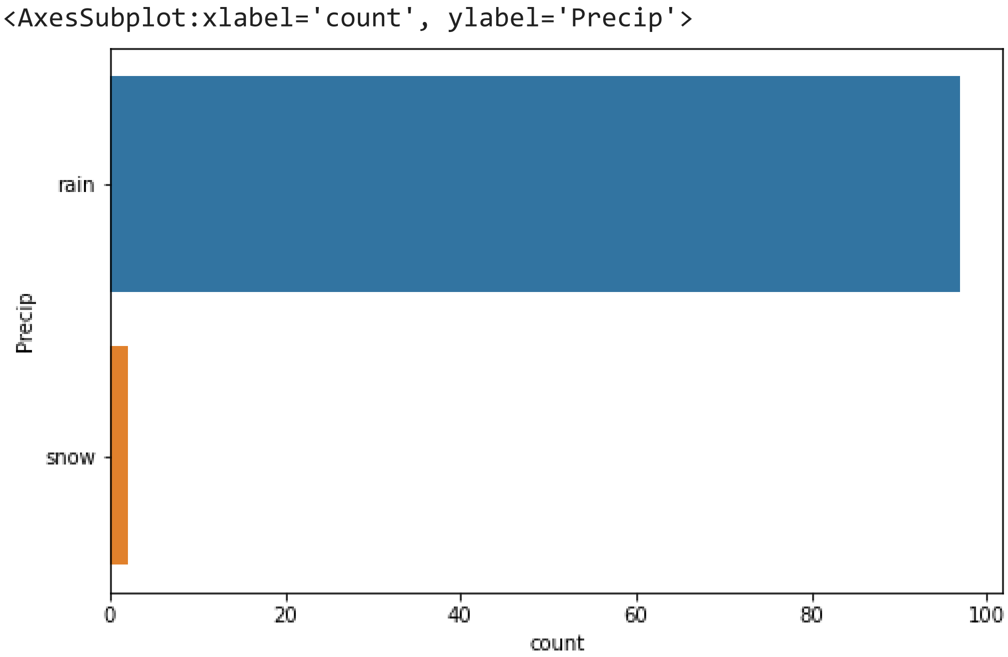
```
datasets.fillna(method='ffill', inplace=True) #Fill NA/NaN values using the specified method
```

```
# Calculate total number of cells in dataframe
totalCells = np.product(datasets.shape) #Return a tuple representing the dimensionality of the DataFrame.

# Count number of missing values per column
missingCount = datasets.isnull().sum() #Detect missing values

# Calculate total number of missing values
totalMissing = missingCount.sum()

# Calculate percentage of missing values
print("The weather history dataset contains", round(((totalMissing/totalCells) * 100), 2), "%", "missing values.")
```

```
The weather history dataset contains 0.0 % missing values.
```

```
rcParams['figure.figsize'] = 8, 5 #A dictionary object including validation.
sns.countplot(y=datasets['Precip']) #Show the counts of observations in each categorical bin using bars.
```

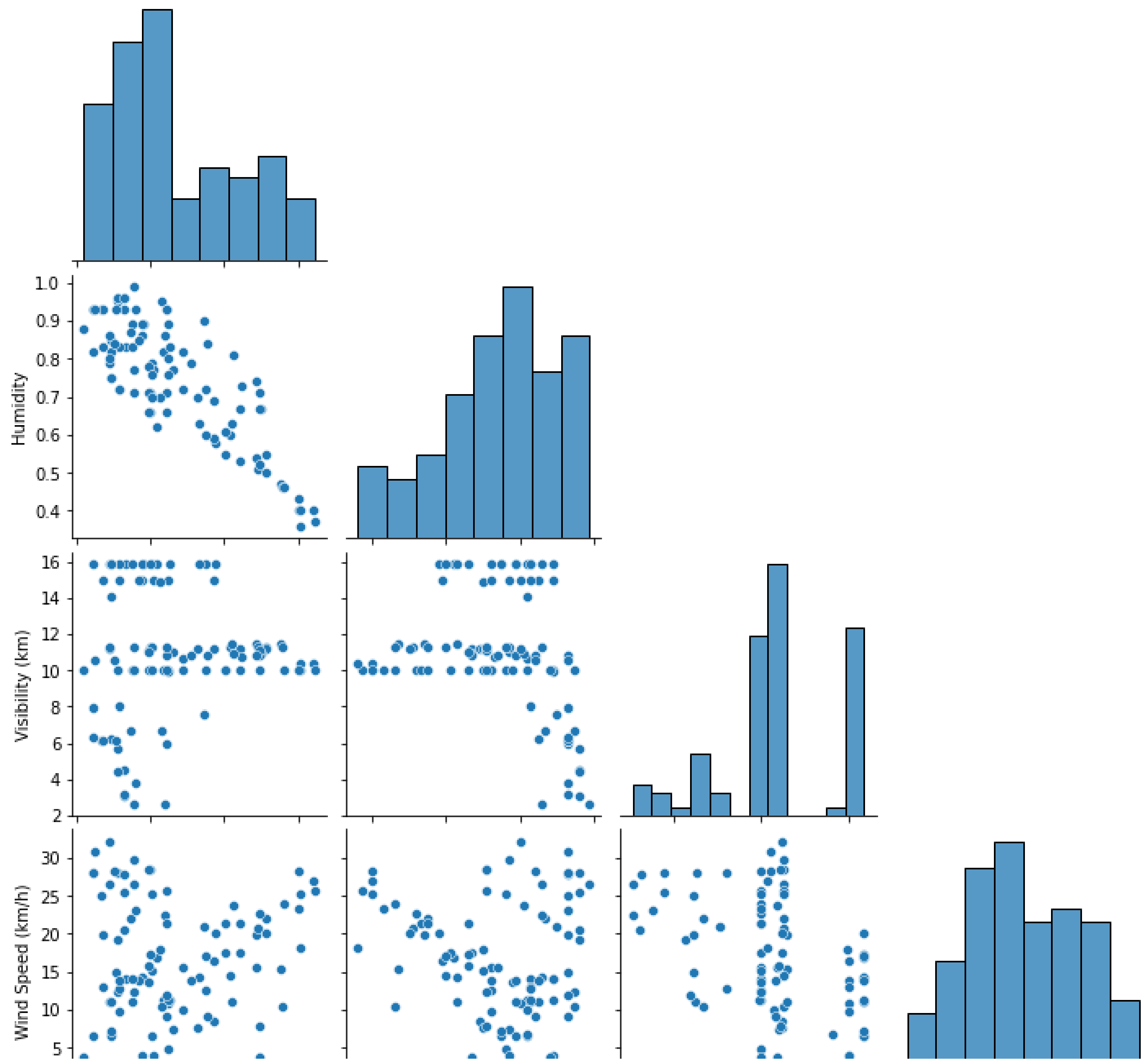
```
<AxesSubplot:xlabel='count', ylabel='Precip'>
```




```
datasets['Precip'].value_counts(dropna=False)
```

```
rain    97
snow     2
Name: Precip, dtype: int64
```

```
sns.pairplot(datasets,corner=True); #Two-dimensional, size-mutable, potentially heterogeneous tabular data.
# To find the relationship between different type of Column
```



Multilinear Regression

```
pd.get_dummies(datasets.Precip ) #One-dimensional ndarray with axis labels (including time series).
#A dummy variable is a binary variable that indicates whether a separate categorical variable takes on a specific value.
```

	rain	snow
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0
...	...	...
94	1	0
95	1	0
96	1	0
97	1	0
98	1	0

99 rows × 2 columns

```
# Label Encoder - Convert multiple String Value to the numerical Value
# One Hot Encoder - Dividing Value into column
```

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder #ncode categorical features as a one-hot numeric array.
labelencoder_X = LabelEncoder() #Encode target labels with value between 0 and n_classes-1.
#convert string values to float integers
X[:, 3] = labelencoder_X.fit_transform(X[:, 3]) #Fit label encoder and return encoded labels
```

```
print(X[:, 3])
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
X = datasets["Apparent Temperature (C)"].values.reshape(-1,1)
```

```
Y = datasets["Temperature (C)"].values.reshape(-1,1)
```

```
from sklearn.model_selection import train_test_split #Split arrays or matrices into random train and test subsets
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.5, random_state = 0) #The data will not be changed be
```

```
from sklearn.linear_model import LinearRegression #Ordinary least squares Linear Regression.
regressor = LinearRegression()
```

```
regressor.fit(X_Train, Y_Train)# Fit linear model.
# Training
```

```
LinearRegression()
```

```
Y_Pred = regressor.predict(X_Test) #Predict using the linear model.
#Predict the data using X_Test
```

```
from sklearn.metrics import mean_squared_error,r2_score #R^2 (coefficient of determination) regression score function.
print("mean squared error",mean_squared_error(Y_Test,Y_Pred)) #estimator measures the average of error squares
#Points how much far from regression line mean squared error is used
print("root square",r2_score(Y_Test,Y_Pred)) #Root Square is finding the accuracy , about how much Data is accurate
math.sqrt(mean_squared_error(Y_Test,Y_Pred)) #Return the square root of x
```

```
mean squared error 0.820642394464359
root square 0.9569110317047951
0.9058931473768631
```

Simple Linear Regression

```
from sklearn.model_selection import train_test_split #Split arrays or matrices into random train and test subsets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/3, random_state = 0)
```

```
from sklearn.linear_model import LinearRegression #Ordinary least squares Linear Regression.
regressor = LinearRegression()
regressor.fit(X_train, Y_train) #Fit linear model.
```

```
LinearRegression()
```

```
Y_pred = regressor.predict(X_test) # Predict using the linear model.
```

```
print(regressor.coef_) #Prints the values to a stream, or to sys.stdout by default.
```

```
[[0.78073585]]
```

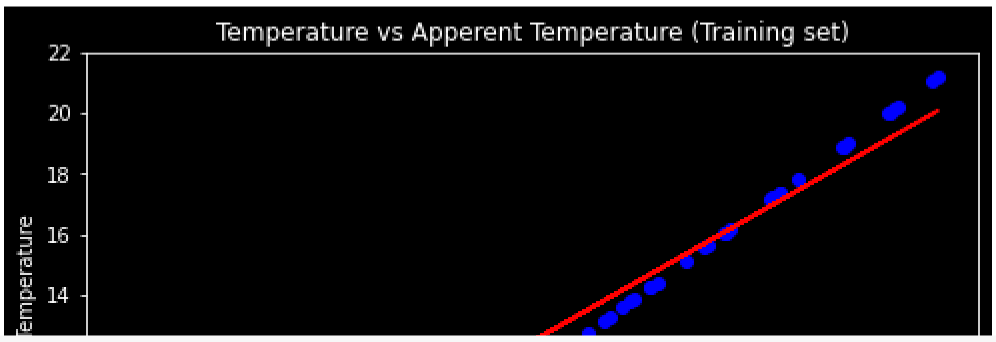
```
print(regressor.intercept_) #Double-precision floating-point number type, compatible with Python `float`
```

```
[3.55327436]
```

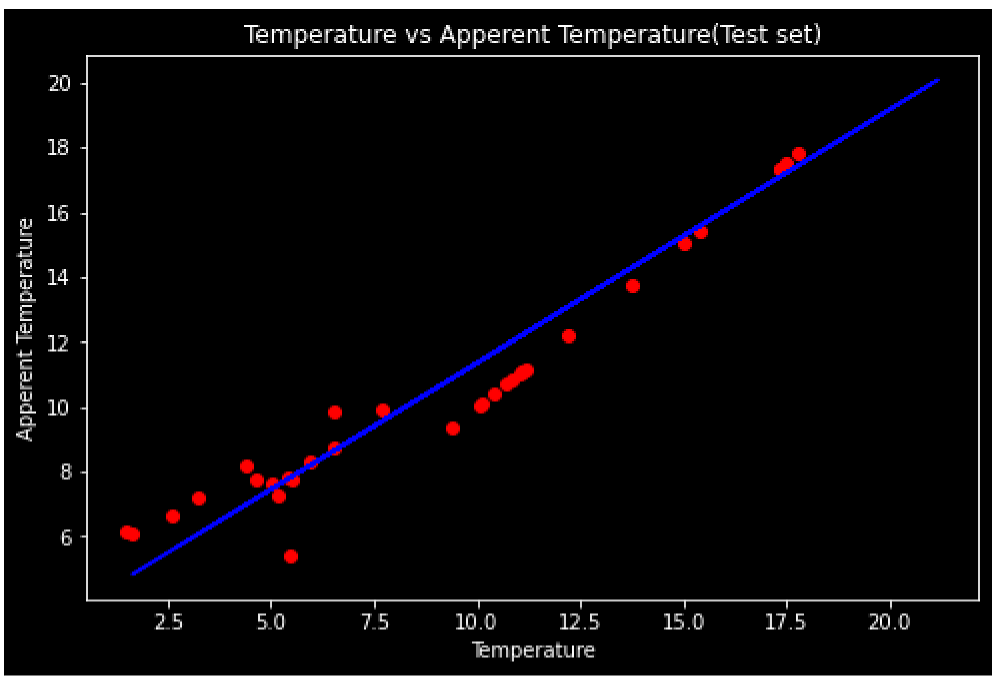
```
from sklearn.metrics import r2_score #R^2 (coefficient of determination) regression score function.
r2_score(Y_test,Y_pred)
```

```
0.9209023074598142
```

```
 # plotting scatter plot for the Training dataset
plt.style.use('dark_background') #ploting with bg color black
plt.scatter(X_train, Y_train, color = 'blue') #A scatter plot of *y* vs. *x* with varying marker size and/or color.
plt.plot(X_train, regressor.predict(X_train), color = 'red') #Plot y versus x as lines and/or markers
plt.title('Temperature vs Apperent Temperature (Training set)') #Set a title for the axes.
plt.xlabel('Temperature') #Set the label for the x-axis.
plt.ylabel("Apperent Temperature") #Set the label for the y-axis
plt.show() #Display all open figures.
```




```
# plotting scatter plot for the Testing dataset
plt.scatter(X_test, Y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Temperature vs Apperent Temperature(Test set)')
plt.xlabel('Temperature')
plt.ylabel('Apperent Temperature')
plt.show()
```



Polynomial Regression

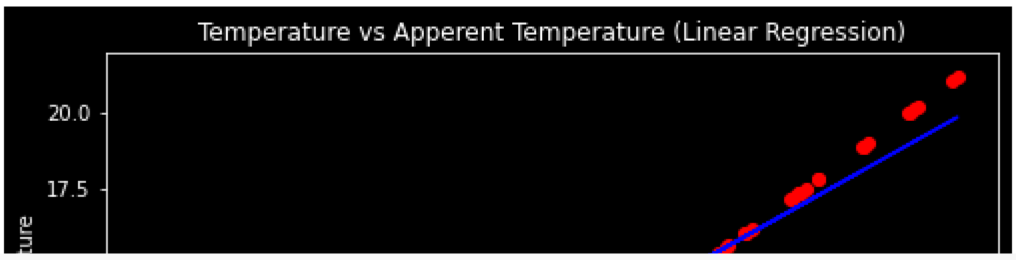
```
#Training the Linear Regression model on the whole dataset
from sklearn.linear_model import LinearRegression #sklearn is a Python module integrating classical machine
#LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between th
lin_reg = LinearRegression()
lin_reg.fit(X, Y) #Fit linear model
```

```
LinearRegression()
```

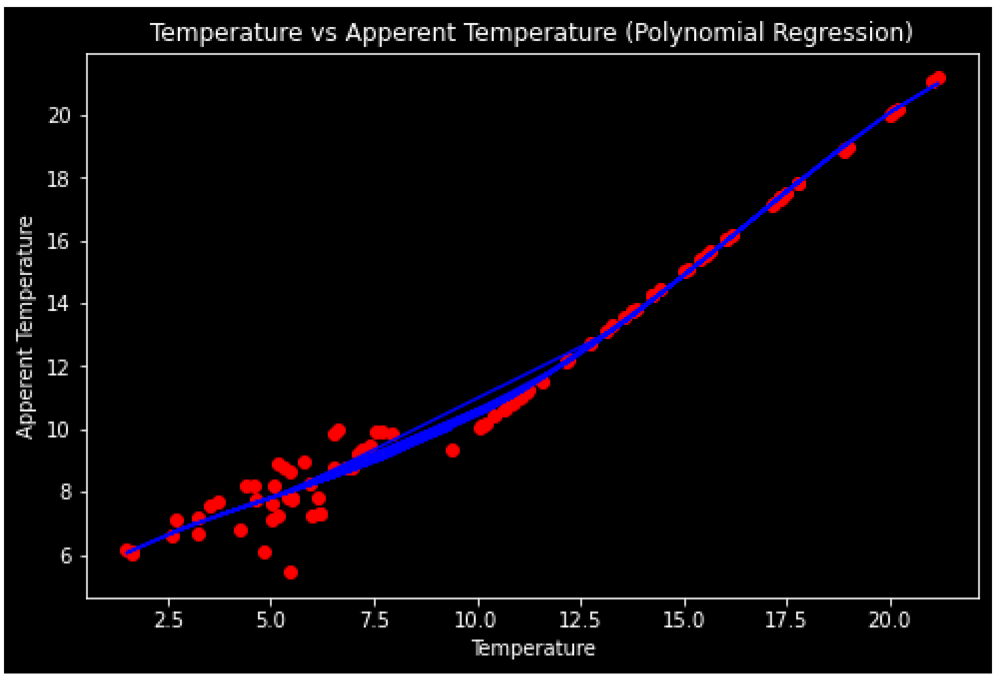
```
#Training the Polynomial Regression model on the whole dataset
#In Polynomial Linear Regression , line of graph change in the form of curve to predict accurately depending on degrees
from sklearn.preprocessing import PolynomialFeatures #Generate polynomial and interaction features.
poly_reg = PolynomialFeatures(degree = 4) #Generate a new feature matrix consisting of all polynomial combinations of the fe
X_poly = poly_reg.fit_transform(X) #Fit to data, then transform it
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, Y) #Fit linear model
```

```
LinearRegression()
```

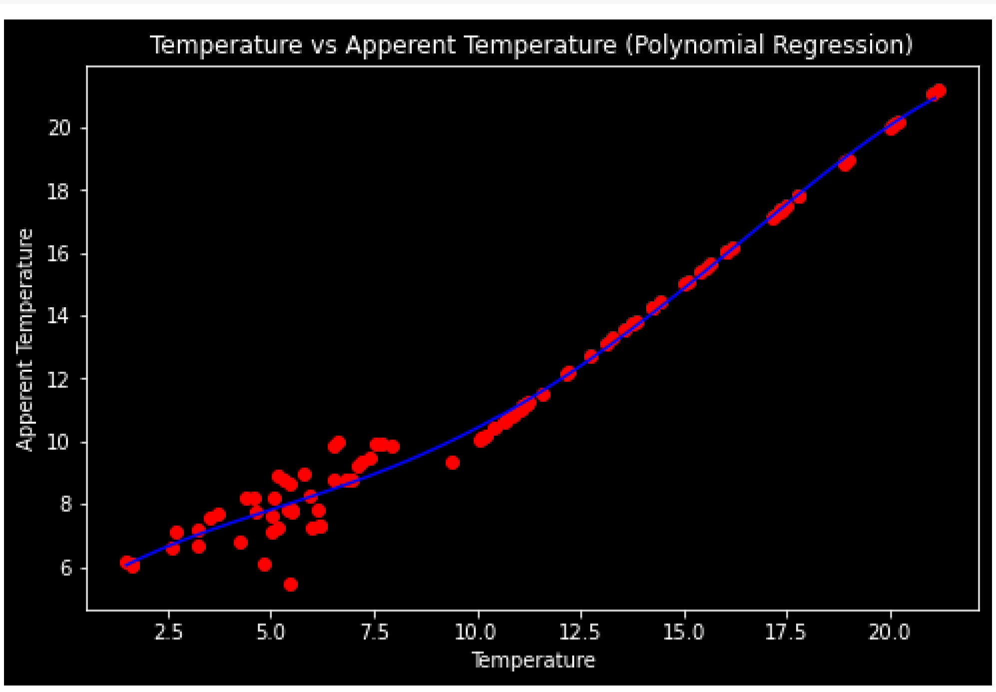
```
#Visualising the Linear Regression Results
plt.scatter(X, Y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title('Temperature vs Apperent Temperature (Linear Regression)')
plt.xlabel('Temperature')
plt.ylabel('Apperent Temperature')
plt.show()
```




```
#Visualising the Polynomial Regression results
plt.scatter(X, Y, color = 'red') #A scatter plot of *y* vs. *x* with varying marker size and/or color.
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue') #Plot y versus x as lines and/or markers
plt.title('Temperature vs Apperent Temperature (Polynomial Regression)') #Set a title for the axes
plt.xlabel('Temperature') #Set the label for the x-axis
plt.ylabel('Apperent Temperature')
plt.show() #Display all open figures
```




```
#Visualising the Polynomial Regression results (for higher resolutionand smoother curve)
X_grid = np.arange(min(X), max(X), 0.1) #Return evenly spaced values within a given interval.
X_grid = X_grid.reshape((len(X_grid), 1)) #Returns an array containing the same data with a new shape.
plt.scatter(X, Y, color = 'red') #A scatter plot of *y* vs. *x* with varying marker size and/or color.
plt.plot(X_grid, lin_reg_2.predict(poly_reg.fit_transform(X_grid)), color = 'blue') #Plot y versus x as lines and/or markers
plt.title('Temperature vs Apperent Temperature (Polynomial Regression)') #Set a title for the axes
plt.xlabel('Temperature') #Set the label for the x-axis
plt.ylabel('Apperent Temperature') #Set the label for the y-axis.
plt.show() #Display all open figures
```




```
#Predicting a new result with Linear Regression
lin_reg.predict([[6.5]]) #Predict using the linear model.
```

```
array([[8.61447618]])
```

```
#Predicting a new result with Polynomial Regression
lin_reg_2.predict(poly_reg.fit_transform([[6.5]])) #Predict using the linear mode and Fit to data, then transform it.
```

```
array([[8.47256823]])
```

Decision Tree Regression

```
#Training the Decision Tree Regression model on the whole dataset
from sklearn.tree import DecisionTreeRegressor #A decision tree regressor
regressor = DecisionTreeRegressor(random_state = 0)
```

```
regressor.fit(X, Y) #Build a decision tree regressor from the training set (X, y)
```

```
DecisionTreeRegressor(random_state=0)
```

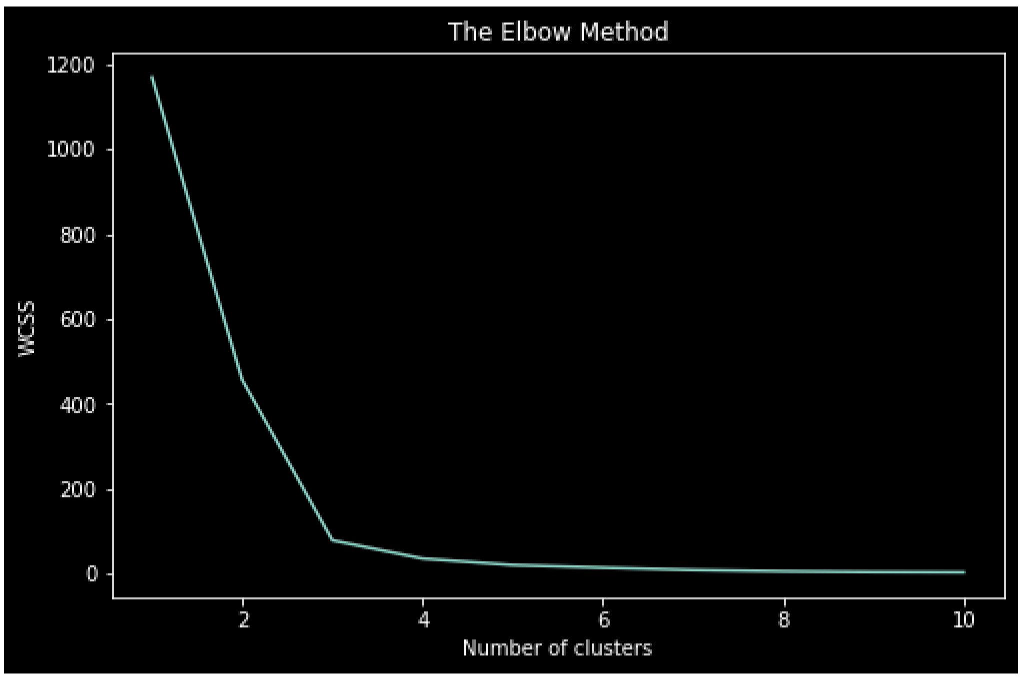
```
#Predicting a new result
regressor.predict([[7.5]]) #Predict class or regression value for X
```

```
array([9.91111111])
```

K-Means Clustering

```
datasets=pd.read_csv("C:\\Users\\Somu\\Downloads\\WEATHER FORCASTING (Project).csv")
X = datasets.iloc[:, [1, 2]].values
```

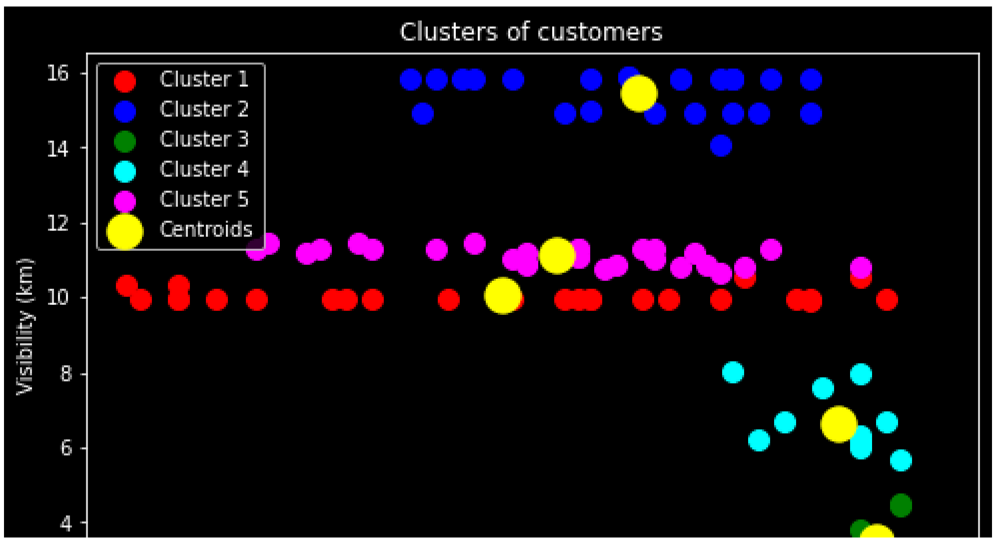
```
#Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
     kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)#each cluster
     kmeans.fit(X)#data set of x value
     wcss.append(kmeans.inertia_)#sse values is stored inwcss
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```




```
#Training the K-Means model on the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
# Each and every cluster number represent what is the Humidity
#Numbers represent clusting numbers
y_kmeans = kmeans.fit_predict(X) #To identify the y- value
y_kmeans
```

```
array([1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4,
       0, 0, 1, 1, 1, 1, 3, 1, 0, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3,
       2, 2, 2, 2, 2, 3, 2, 2, 1, 3, 3, 3, 3, 3, 0, 4, 4, 0, 4, 4, 4, 4,
       4, 0, 1, 0, 4, 0, 1, 1, 1, 1, 0])
```

```
#Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
#0 represent first cluster , other 0 represent annual income and 1 represent pending score , s=100 total number of points
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow',label="Centroids")
plt.title('Clusters of customers')
plt.xlabel('Humidity')
plt.ylabel('Visibility (km)')
plt.legend()
plt.show()
```



Classifications

```
datasets=pd.read_csv("C:\\Users\\Somu\\Downloads\\WEATHER FORCASTING 1 (Project).csv")#Read a comma-separated values (csv) f:
x = datasets.iloc[:, :-1].values # : - Represent all the rows , :-1 - Taking all the columns from (n-1),-2
y = datasets.iloc[:, 4].values # : - Represent all the rows  , 4 - upto 5th column
datasets
```

	Temperature (C)	Humidity	Visibility (km)	Precip	Wind Speed (km/h)	Loud Cover	Apparent Temperature (C)
0	9.472222	0.89	15.8263	1	14.1197	0	7.388889
1	9.355556	0.86	15.8263	1	14.2646	0	7.227778
2	9.377778	0.89	14.9569	1	3.9284	0	9.377778
3	8.288889	0.83	15.8263	1	14.1036	0	5.944444
4	8.755556	0.83	15.8263	1	11.0446	0	6.977778
...	...	...	...	...	...	...	...
94	7.827778	0.72	15.8263	1	13.8943	0	5.405556
95	7.855556	0.72	15.0052	1	9.8049	0	6.122222
96	7.316667	0.75	15.8746	1	6.6654	0	6.211111
97	7.244444	0.75	15.8746	1	7.1162	0	6.005556
98	5.438889	0.88	9.9820	1	3.7191	0	5.438889

99 rows × 7 columns

LOGISTICS REGRESSION

```
y=datasets.iloc[:,-4]
x= datasets.iloc[:,[3,5]]
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0) #One-dimensional ndarray with axis
st_x= StandardScaler() #Standardise the Value , Set the range of Dataset
x_train= st_x.fit_transform(x_train)##Two-dimensional, size-mutable, potentially heterogeneous tabular data.
x_test= st_x.transform(x_test)
#scaler = MinMaxScaler()
#X_train= scaler.fit_transform(X_train)
#X_test= scaler.fit_transform(X_test)
Training_Accuracy=[]
Testing_Accuracy=[]
model=[]
```

```
from sklearn.model_selection import train_test_split #importing train_test_split from sklearn.preprocessing
x_Train, x_Test, y_Train, y_Test = train_test_split(x, y, train_size=0.7,test_size = 0.3, random_state=0) #assigned testsize
```

```
from sklearn.linear_model import LogisticRegression #importing logistic regression
from sklearn import metrics #import matrices
logreg = LogisticRegression() #classifier.
logreg.fit(x_Train, y_Train) # Fitted estimator.
```

```
LogisticRegression()
```

```
#Predicting the Test set results
y_pred = logreg.predict(x_test)
```

```
y_pred = logreg.predict(x_test)
```

```
x_pred = logreg.predict(x_Train) #predicting x values based on traning features
y_pred = logreg.predict(x_Test) #predicting y values based on x_test
Training_Accuracy.append(metrics.accuracy_score(y_Train, x_pred))#add traning accuracy to the list
Testing_Accuracy.append(metrics.accuracy_score(y_Test, y_pred))#add testing accuracy to the list
model.append('LogisticRegression')#add the specified name
print("Training Accuracy:",metrics.accuracy_score(y_Train, x_pred)) #print traning accuracy
print("Testing Accuracy:",metrics.accuracy_score(y_Test, y_pred))#print testing accuracy
```

```
Training Accuracy: 0.9710144927536232
Testing Accuracy: 1.0
```

SupportVectorMachine

```
yy=datasets.iloc[:,-4]
xx= datasets.iloc[:,[3,5]]
```

```
xx_train, xx_test, yy_train, yy_test= train_test_split(xx, yy, test_size= 0.25, random_state=0) #One-dimensional ndarray with
st_x= StandardScaler() #Standardise the Value , Set the range of Dataset
xx_train= st_x.fit_transform(xx_train)##Two-dimensional, size-mutable, potentially heterogeneous tabular data.
xx_test= st_x.transform(xx_test)
#scaler = MinMaxScaler()
#X_train= scaler.fit_transform(X_train)
#X_test= scaler.fit_transform(X_test)
Training_Accuracy=[]
Testing_Accuracy=[]
model=[]
```

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

```
SVC(kernel='linear')
```

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

```
#Predicting the Test set results
y_pred=svclassifier.predict(xx_test)
```

```
Training_Accuracy.append(metrics.accuracy_score(yy_train, x_pred))
Testing_Accuracy.append(metrics.accuracy_score(yy_test, y_pred))
model.append('SVM')
print("Training Accuracy:",metrics.accuracy_score(yy_train, x_pred))
print("Testing Accuracy:",metrics.accuracy_score(yy_test, y_pred))
```

```
Training Accuracy: 1.0
Testing Accuracy: 1.0
```

KNN

```
b=datasets.iloc[:,-4] #[row,column start number: column end number(n-1)]
a= datasets.iloc[:,[3,5]]
```

```
a_train, a_test, b_train, b_test= train_test_split(a, b, test_size= 0.25, random_state=0) #One-dimensional ndarray with axis
st_x= StandardScaler() #Standardise the Value , Set the range of Dataset
a_train= st_x.fit_transform(a_train)##Two-dimensional, size-mutable, potentially heterogeneous tabular data.
a_test= st_x.transform(a_test)
#scaler = MinMaxScaler()
#X_train= scaler.fit_transform(X_train)
#X_test= scaler.fit_transform(X_test)
Training_Accuracy=[]
Testing_Accuracy=[]
model=[]
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(a_train, b_train)
```

```
KNeighborsClassifier()
```

```
y_pred = knn.predict(a_test)
```

```
from sklearn import metrics
x_pred = knn.predict(a_train)
y_pred = knn.predict(a_test)
Training_Accuracy.append(metrics.accuracy_score(b_train, x_pred))
Testing_Accuracy.append(metrics.accuracy_score(b_test, y_pred))
model.append('KNeighbors')
print("Training Accuracy:",metrics.accuracy_score(b_train, x_pred))
print("Testing Accuracy:",metrics.accuracy_score(b_test, y_pred))
```

```
Training Accuracy: 0.972972972972973
Testing Accuracy: 1.0
```

DECISION TREE CLASSIFICATION

```
Y=datasets.iloc[:,-4] #[row,column start number: column end number(n-1)]
X= datasets.iloc[:,[3,5]]
```

```
X_train, X_test, Y_train, Y_test= train_test_split(X, Y, test_size= 0.25, random_state=0) #One-dimensional ndarray with axis
st_x= StandardScaler() #Standardise the Value , Set the range of Dataset
X_train= st_x.fit_transform(X_train)##Two-dimensional, size-mutable, potentially heterogeneous tabular data.
X_test= st_x.transform(X_test)
#scaler = MinMaxScaler()
#X_train= scaler.fit_transform(X_train)
#X_test= scaler.fit_transform(X_test)
Training_Accuracy=[]
Testing_Accuracy=[]
model=[]
```

```
#Training the Decision Tree Classification model on the Training set
from sklearn.tree import DecisionTreeClassifier #A decision tree classifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, Y_train)
```

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

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

```
y_pred = classifier.predict(X_test)
x_pred = classifier.predict(X_train)
Training_Accuracy.append(metrics.accuracy_score(Y_train, x_pred)) #add traning accuracy to the list
Testing_Accuracy.append(metrics.accuracy_score(Y_test, y_pred))#add testing accuracy to the list
model.append('DecisionTree')#add the specified name
print("Training Accuracy:",metrics.accuracy_score(Y_train, x_pred))#print traning accuracy
print("Testing Accuracy:",metrics.accuracy_score(Y_test, y_pred))#print testing accuracy
```

```
Training Accuracy: 1.0
Testing Accuracy: 1.0
```

Random Forest Classification

```
#Training the Random Forest Classification model on the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=20)
classifier.fit(X_train, Y_train)
```

```
RandomForestClassifier(n_estimators=20)
```

```
y_pred = classifier.predict(X_test) #prediction
x_pred = classifier.predict(X_train)
Training_Accuracy.append(metrics.accuracy_score(y_train, x_pred))
Testing_Accuracy.append(r2_score(y_test,y_pred))
model.append('RandomForest')
print("Training Accuracy:",metrics.accuracy_score(y_train, x_pred))
print("Testing Accuracy:",r2_score(y_test,y_pred))
```

```
Training Accuracy: 1.0
Testing Accuracy: 1.0
```

```
#Predicting the Test set results
```

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

```
#Graph showing Training and Testing accuracy of Decision Tree and Random Forest Classification
# set width of bar
barWidth = 0.10
# set height of bar
fig = plt.gcf(); #create new figure
fig.set_facecolor("black")
fig.set_size_inches(15,10);
# Set position of bar on X axis
r1 = np.arange(len(Training_Accuracy))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]

# Make the plot
plt.bar(r1,Training_Accuracy, color='springgreen', width=barWidth, edgecolor='yellow', label='Training_Accuracy')
plt.bar(r2,Testing_Accuracy, color='darkblue', width=barWidth, edgecolor='lime', label='Testing_Accuracy')
# Add xticks on the middle of the group bars
plt.xlabel('Models', fontweight='bold')
plt.ylabel('Accuracy', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(Training_Accuracy))], model)
# Create legend & Show graphic
plt.legend()
plt.show()#show barplot
```