॥वसुधैव कुटुम्बकम्॥

SYMBIOSIS INSTITUTE OF TECHNOLOGY

PUNE, MAHARASHTRA

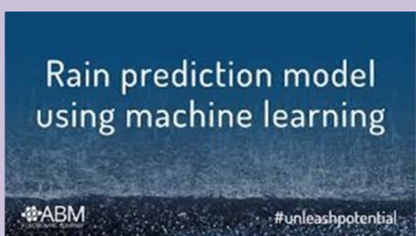
SKILL DEVELOPMENT MINI PROJECT 1

TITLE – RAINFALL PREDICTION USING MACHINE LEARNING

Under the guidance of
SEEMA PATIL MAM

SUBMITTED BY –

Himank Jain	**18070122027**



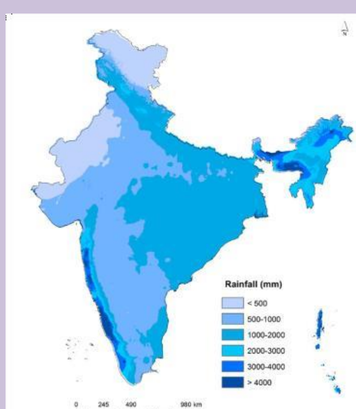


TABLE OF CONTENTS

I. INTRODUCTION

Rainfall forecasting is very important because heavy and irregular rainfall can have many impacts like destruction of crops and farms, damage of property so a better forecasting model is essential for an early warning that can minimize risks to life and property and also managing the agricultural farms in better way. This prediction mainly helps farmers and also water resources can be utilized efficiently. Rainfall prediction is a challenging task and the results should be accurate. There are many hardware devices for predicting rainfall by using the weather conditions like temperature, humidity, pressure. These traditional methods cannot work in an efficient way so by using machine learning techniques we can produce accurate results. We can just do it by having the historical data analysis of rainfall and can predict the rainfall for future seasons. We can apply many techniques like classification, regression according to the requirements and also, we can calculate the error between the actual and prediction and also the accuracy. Different techniques produce different accuracies so it is important to choose the right algorithm and model it according to the requirements.

II. PROBLEM STATEMENT

Rainfall Prediction is the application of science and technology to predict the amount of rainfall over a region. It is important to exactly determine the rainfall for effective use of water resources, crop productivity and pre-planning of water structures. Rainfall prediction is important as heavy rainfall can lead to many disasters. The prediction helps people to take preventive measures and moreover the prediction should be accurate.

The aim of this project is to analyse historical data related to rainfall and use that data to develop a model using machine learning algorithm and that can predict whether it will rain tomorrow or not.

III. SCOPE

There are two types of prediction short term rainfall prediction and long-term rainfall. Prediction mostly short-term prediction can give us the accurate result. The scope of this project covers short-term prediction. On the basis of selected dataset our model will predict whether it will rain tomorrow or not.

Our dataset consists of the following features –

Date, Location, MinTemp, MaxTemp, Rainfall, Evaporation, Sunshine, WindGustDir, WindGustSpeed, WindDir9am, WindDir3pm, WindSpeed9am, WindSpeed3pm, Humidity9am, Humidity3pm, Pressure9am, Pressure3pm, Cloud9am, Cloud3pm, Temp9am, Temp3pm, RainTodaysort, RainTomorrow

IV. LITERATURE SURVEY

AUTHOR	JOURNAL / YEAR	TITLE	METHODOLOGY	KEY FINDINGS	LIMITATIONS
R Vijayan, V Mareeswari	INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH 06 JUNE 2020	Estimating Rainfall Prediction using Machine Learning Techniques on a Dataset	The classification model used in this work comprises of four phases: selection of the correct dataset, preprocessing, prediction and results simulation. • Exploring Data Analysis – During this progression playing out some enlightening examination and deciding the objective variable • Data Preprocessing – Includes formatting, cleaning and sampling • Feature Extraction – The extraction of a function is a method of reduction of attributes. Unlike the selection of features that rank the current attributes according to their predictive significance, the extraction of features transforms the attributes • Applying Algorithms – In this stage different machine learning algorithms are applied and evaluated using confusion matrix and classification report	• In this study, 12 years of historical weather data from 1 December 2005 until 31 November 2017 are used for prediction • Three algorithms were considered for prediction – • Random forest, SVM and Logistic Regression • Random Forest and logistic regression gave best results with accuracy of 82%	• Accuracy can be improved by hyperparameter tuning • A model based on neural network can be used for better perfromance
Ayisha Siddiqua L, Senthil kumar	International Journal of Recent Technology and Engineering (IJRTE) December 2019	Rainfall Prediction Using Machine Learning Algorithms	Before moving to building a model for prediction, steps such as EDA, pre-processing, visualization are performed. The following algorithms are considered for prediction – • SVM use to solve the classification problems which find the best fit line between the classes which also known as the hyperplane. Distance from the hyperplane corresponds to the confidence of prediction • Navie Bayes classifier is based on the probability theorem which is Bayes theorem. It is a powerful algorithm for predictive modelling. • Random forest is supervised learning method in which a classification tree is generated. In this algorithm input data vector put in each tree of the forest to classify a new object from an input feature vectors.	• This Paper has presented a supervised rainfall learning model which used machine learning algorithms to classify rainfall data. • For prediction 3 algorithms were considered- SVM, Random Forest and Naïve Bayes • Random Forest yielded highest accuracy of 65%.	• Scope is limited to Rainfall prediction • A hybrid model is suggested to improve accuracy and performance • Future work includes Storm predictions and Crop prediction.

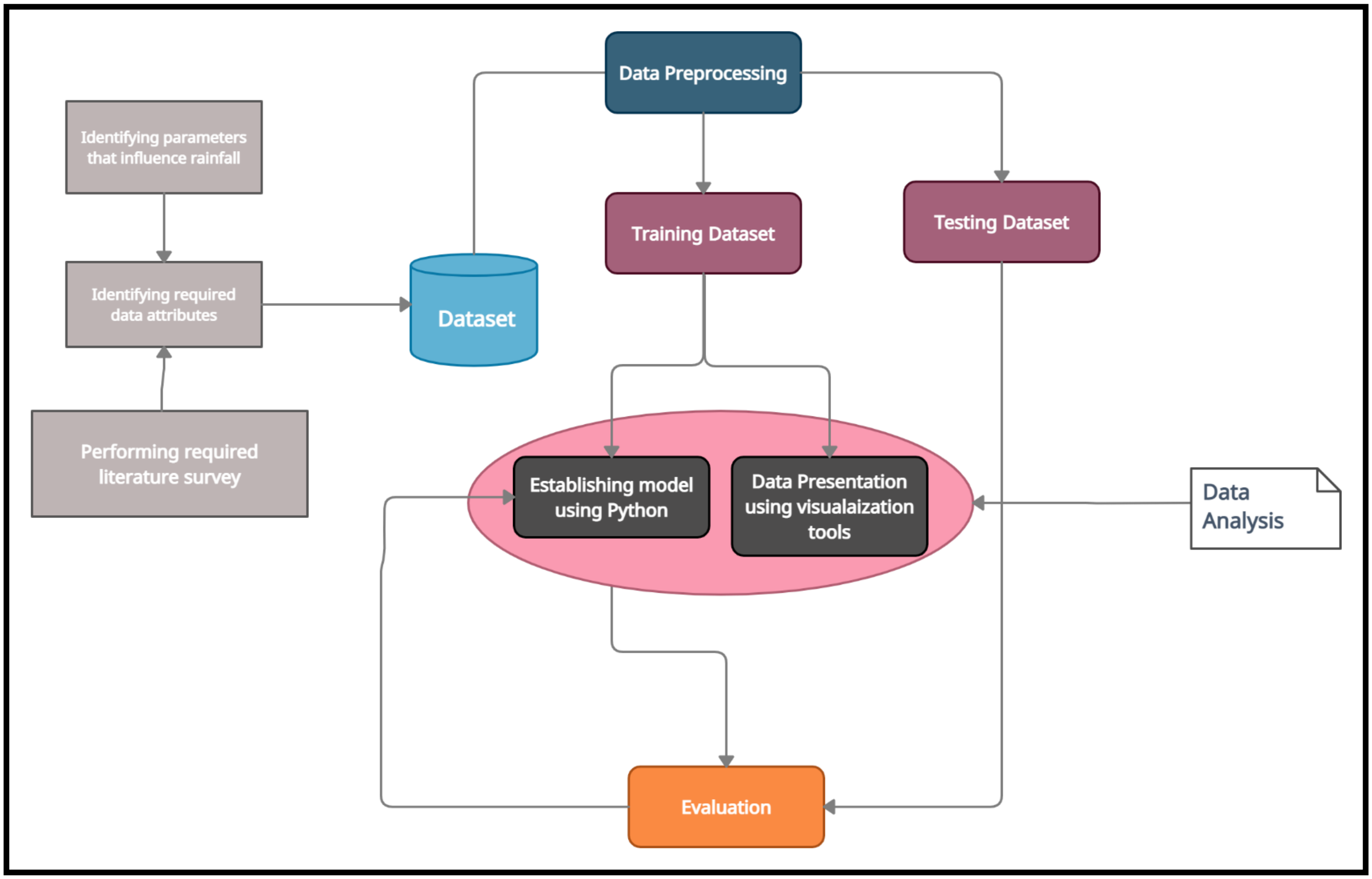
V. METHODOLOGY –

Dataset Description –

This dataset contains about 10 years of daily weather observations from many locations across Australia.
The dataset consists of 23 attributes. RainTomorrow is the target variable to predict. It means -- did it rain the next day, Yes or No? This column is Yes if the rain for that day was 1mm or more.

- Date - The date of observation
- Location - The common name of the location of the weather station
- MinTemp - The minimum temperature in degrees celsius
- MaxTemp - The maximum temperature in degrees celsius
- Rainfall - The amount of rainfall recorded for the day in mm
- Evaporation - The so-called Class A pan evaporation (mm) in the 24 hours to 9am
- Sunshine - The number of hours of bright sunshine in the day.
- WindGustDir - The direction of the strongest wind gust in the 24 hours to midnight
- WindGustSpeed - The speed (km/h) of the strongest wind gust in the 24 hours to midnight
- WindDir9am - Direction of the wind at 9am
- WindDir3pm - Direction of the wind at 3pm
- WindSpeed9am - Wind speed (km/hr) averaged over 10 minutes prior to 9am
- WindSpeed3pm - Wind speed (km/hr) averaged over 10 minutes prior to 3pm
- Humidity9am - Humidity (percent) at 9am
- Humidity3pm - Humidity (percent) at 3pm
- Pressure9am - Atmospheric pressure (hpa) reduced to mean sea level at 9am
- Pressure3pm - Atmospheric pressure (hpa) reduced to mean sea level at 3pm
- Cloud9am - Fraction of sky obscured by cloud at 9am. This is measured in "oktas", which are a unit of eigths. It records how many eigths of the sky are obscured by cloud. A 0 measure indicates completely clear sky whilst an 8 indicates that it is completely overcast.
- Cloud3pm - Fraction of sky obscured by cloud (in "oktas": eighths) at 3pm.
- Temp9am - Temperature (degrees C) at 9am
- Temp3pm - Temperature (degrees C) at 3pm
- RainTodaysort - 1 if precipitation (mm) in the 24 hours to 9am exceeds 1mm, otherwise 0
- RainTomorrow - The amount of next day rain in mm. A kind of measure of the "risk".

ARCHITECTURE –



STEPS INVOLVED –

1.) DATA PREPROCESSING –

Missing Values –

- Identifying the count and percentage of missing values in each column
- Missing Values Handled by Random Sample imputation to maintain the variance
- Removing null values of continuous features and filling it with median of the continuous feature
- Removing null values of discrete features and filling it with mode of the continuous feature

Categorical values –

- Converting RainTomorrow and RainToday features to 0/1.
- Converting wind direction to numerical feature
- Categorical Values like location, wind direction are handled by using Target guided encoding

Removing Outliers –

- Outliers are handled using IQR and boxplot

Over Sampling –

- We have an imbalanced dataset which is why we use over sampling (smote) to create a balanced dataset

2.) <u>DATA VISUALIZATION –</u>

- Heat map – To show the correlation between various features of dataset
- Distribution plot – To see how different features of dataset are distributed
- Count plot – To check the count of 0's and 1's in RainTomorrow and RainToday columns.
- Box plot – To check the presence of outliers which were found in several features.
- Bar graph – Used to display number of records for each location
- Pie Chart - Used to display percentage of values in RainToday column. Also used to display RainTomorrow feature location wise. One pie chart to show Wind gust Direction
- Scatter plot – Used to display Maximum temperature location-wise.
- Lmplot – Regression Analysis between Humidity vs Rainfall, Cloud vs Rainfall and Wind speed vs Rainfall

3.) <u>DATA MODELING AND EVALUATION –</u>

Our weather dataset can be called an imbalanced dataset as the number of 0 or ‘No’ in RainTomorrow feature is far more greater than number of 1 or ‘Yes’. What this means is that more than 50% of our dataset says that there wont be any rain tomorrow. This creates a bias towards our prediction. If we create a model with this dataset, it is very much likely that our model will predict “No” to RainTomorrow, hence reducing our accuracy. To solve this issue we use oversampling. Smote or Synthetic minority oversampling technique is an approach that generates sample duplicate values for minority class eventually giving us a balanced dataset.

Once we have our balanced dataset, we apply our 3 considered algorithms –

Algorithm	Accuracy Score	AUC
CatBoost classifier	0.86	0.89
Random Forest	0.84	0.88
Logistic Regression	0.77	0.85

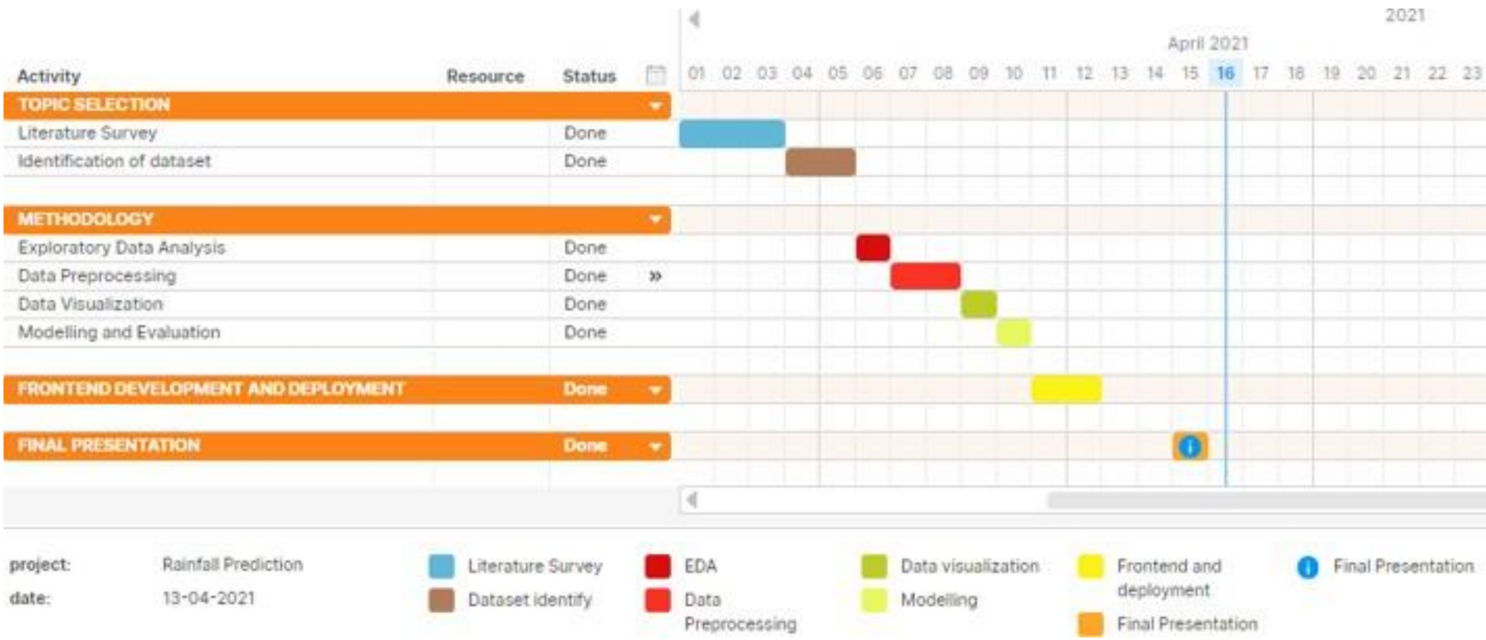
RESULTS – CATBOOST CLASSIFIER GAVE BEST RESULTS WITH ACCURACY OF 86%.

VI. TECHNICAL REQUIREMENTS –

HARDWARE	
Processor	Intel i3 or above
Hard disk	1GB or more
RAM	2GB (minimum)

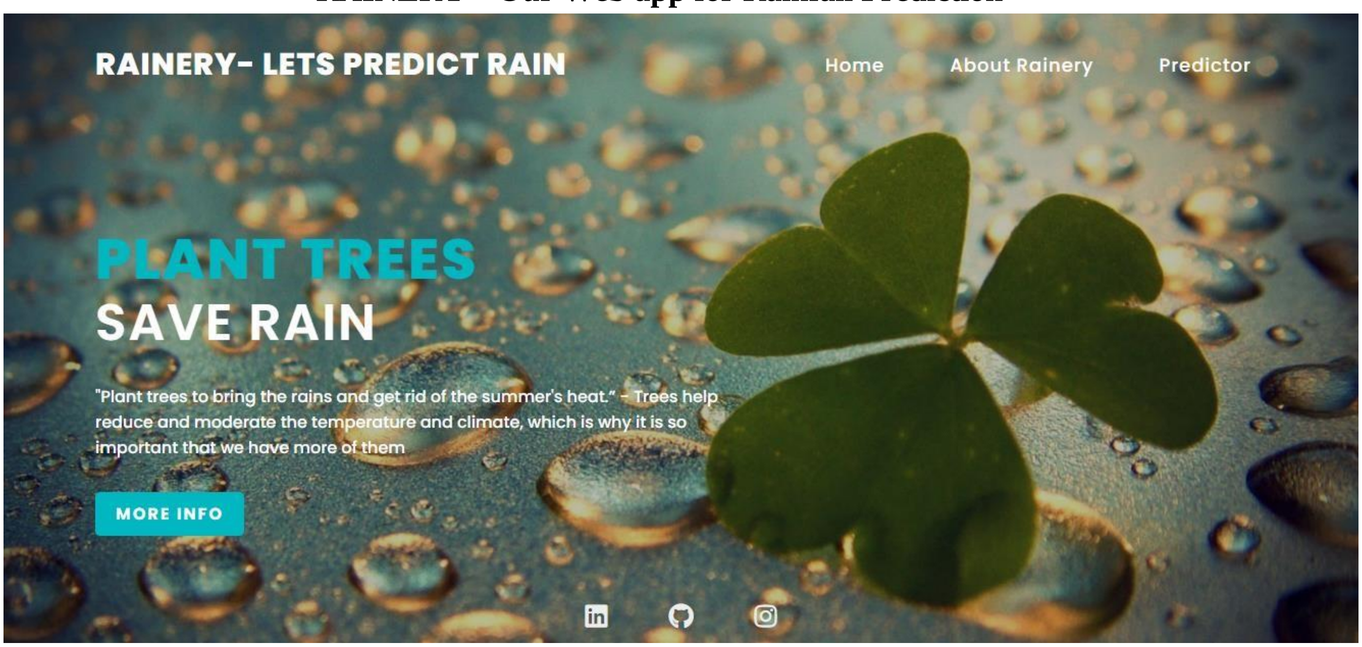
SOFTWARE	
Operating System	Windows 10 or Linux (Recommended)
Programming Language	Python 3.8
Editor	Jupyter Notebook
Python Libraries required	Numpy, Pandas, Matplotlib, Seaborn, Scikit-Learn, CatBoost, Imb-Learn, Flask

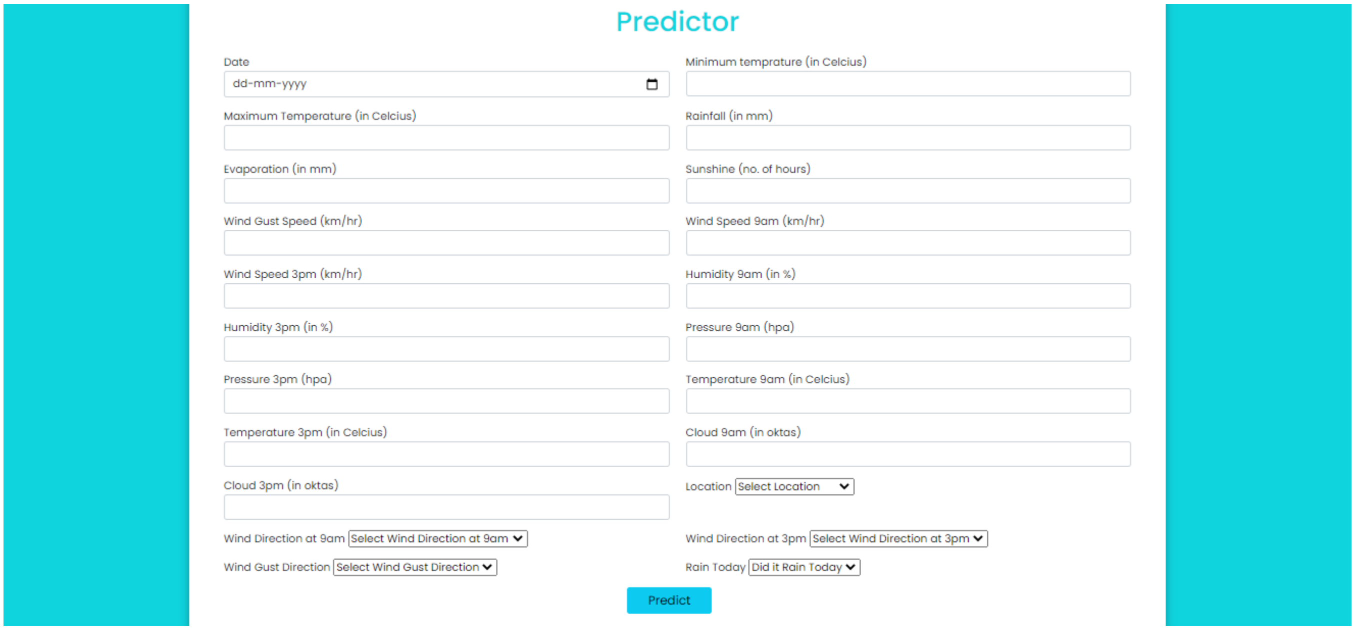
VII. TIMELINE –



VIII. SCREENSHOTS

RAINERY – Our Web app for Rainfall Prediction




Predictor
Date
dd-mm-yyyy
Minimum temprature (in Celcius)
Maximum Temperature (in Celcius)
Rainfall (in mm)
Evaporation (in mm)
Sunshine (no. of hours)
Wind Gust Speed (km/hr)
Wind Speed 9am (km/hr)
Wind Speed 3pm (km/hr)
Humidity 9am (in %)
Humidity 3pm (in %)
Pressure 9am (hpa)
Pressure 3pm (hpa)
Temperature 9am (in Celcius)
Temperature 3pm (in Celcius)
Cloud 9am (in oktas)
Cloud 3pm (in oktas)
Location Select Location
Wind Direction at 9am Select Wind Direction at 9am
Wind Direction at 3pm Select Wind Direction at 3pm
Wind Gust Direction Select Wind Gust Direction
Rain Today Did it Rain Today
Predict

OUTPUT


RAINY DAY
Tomorrow is going to be rainy day. So enjoy yourselves with a cup of coffee and hot snack

Exploratory Data Analysis and Data Preprocessing:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	WNW	20.0	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	WSW	4.0	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	WSW	19.0	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	E	11.0	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	NW	7.0	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           145460 non-null  object
 1   Location       145460 non-null  object
 2   MinTemp        143975 non-null  float64
 3   MaxTemp        144199 non-null  float64
 4   Rainfall       142199 non-null  float64
 5   Evaporation    82670 non-null   float64
 6   Sunshine       75625 non-null   float64
 7   WindGustDir    135134 non-null  object
 8   WindGustSpeed  135197 non-null  float64
 9   WindDir9am     134894 non-null  object
 10  WindDir3pm     141232 non-null  object
 11  WindSpeed9am   143693 non-null  float64
 12  WindSpeed3pm   142398 non-null  float64
 13  Humidity9am    142806 non-null  float64
 14  Humidity3pm    140953 non-null  float64
 15  Pressure9am    130395 non-null  float64
 16  Pressure3pm    130432 non-null  float64
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
 20  Temp3pm        141851 non-null  float64
 21  RainToday      142199 non-null  object
 22  RainTomorrow   142193 non-null  object
dtypes: float64(16), object(7)
memory usage: 25.5+ MB
```

Here we have listed information about our data frame including the type of columns it contains, data types and memory usage.

```
df.describe()
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3
count	143975.000000	144199.000000	142199.000000	82670.000000	75625.000000	135197.000000	143693.000000	142398.000000	142806.000000	140953.0000
mean	12.194034	23.221348	2.360918	5.468232	7.611178	40.035230	14.043426	18.662657	68.880831	51.539
std	6.398495	7.119049	8.478060	4.193704	3.785483	13.607062	8.915375	8.809800	19.029164	20.795
min	-8.500000	-4.800000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	0.000000	0.0000
25%	7.600000	17.900000	0.000000	2.600000	4.800000	31.000000	7.000000	13.000000	57.000000	37.0000
50%	12.000000	22.600000	0.000000	4.800000	8.400000	39.000000	13.000000	19.000000	70.000000	52.0000
75%	16.900000	28.200000	0.800000	7.400000	10.600000	48.000000	19.000000	24.000000	83.000000	66.0000
max	33.900000	48.100000	371.000000	145.000000	14.500000	135.000000	130.000000	87.000000	100.000000	100.0000

The above result lists the statistical summary of all the numerical variables present in the dataset. The different rows contain values corresponding to the statistical measure described by the first column (without heading). The different columns represent the different numerical columns included in the dataset.

```
print(len(df.index))
df.duplicated().sum()
```

```
145460

0
```

The first result displays the total number of rows in the dataset which is 145460. The second result is the number of duplicated rows in the dataset. There are no duplicate rows present in the dataset.

```
numerical_feature = [feature for feature in df.columns if df[feature].dtypes != 'O']
discrete_feature=[feature for feature in numerical_feature if len(df[feature].unique())<25]
continuous_feature = [feature for feature in numerical_feature if feature not in discrete_feature]
categorical_feature = [feature for feature in df.columns if feature not in numerical_feature]
print("Numerical Features Count {}".format(len(numerical_feature)))
print("Discrete feature Count {}".format(len(discrete_feature)))
print("Continuous feature Count {}".format(len(continuous_feature)))
print("Categorical feature Count {}".format(len(categorical_feature)))
```

```
Numerical Features Count 16
Discrete feature Count 2
Continuous feature Count 14
Categorical feature Count 7
```

Here we get count of different types of variables used in the dataset. This is helpful as these variables are dealt differently in data preprocessing.

```python
print(numerical_feature)
```

```
['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am',
'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm']
```

```python
print(discrete_feature)
```

```
['Cloud9am', 'Cloud3pm']
```

```python
print(continuous_feature)
```

```
['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am',
'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Temp9am', 'Temp3pm']
```

```python
print(categorical_feature)
```

```
['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday', 'RainTomorrow']
```

Here we have classified these different variables into their domain categories to get them ready for data preprocessing in the next step as each of them needs to be handled differently.

```python
# Handle Missing Values
df.isnull().sum()*100/len(df)
```

```
Date              0.000000
Location          0.000000
MinTemp           1.020899
MaxTemp           0.866905
Rainfall          2.241853
Evaporation      43.166506
Sunshine         48.009762
WindGustDir       7.098859
WindGustSpeed     7.055548
WindDir9am        7.263853
WindDir3pm        2.906641
WindSpeed9am      1.214767
WindSpeed3pm      2.105046
Humidity9am       1.824557
Humidity3pm       3.098446
Pressure9am      10.356799
Pressure3pm      10.331363
Cloud9am         38.421559
Cloud3pm         40.807095
Temp9am           1.214767
Temp3pm           2.481094
RainToday         2.241853
RainTomorrow      2.245978
dtype: float64
```

The percentage of missing values for each column of the dataset is displayed. It is worth noting that some of the columns contain almost 50% empty values.

```python
def randomsampleimputation(df, variable):
    df[variable]=df[variable]
    random_sample=df[variable].dropna().sample(df[variable].isnull().sum(),random_state=0)
    random_sample.index=df[df[variable].isnull()].index
    df.loc[df[variable].isnull(),variable]=random_sample
```

randomsampleimputation function takes a variable as input along with the dataframe and imputes the missing values of the variable with the help of random sampling.
In this technique, a sample is chosen randomly from the already filled in values of the same variable. Choosing such a sample is done in an unbiased way.

```
randomsampleimputation(df, "Cloud9am")
randomsampleimputation(df, "Cloud3pm")
randomsampleimputation(df, "Evaporation")
randomsampleimputation(df, "Sunshine")
```

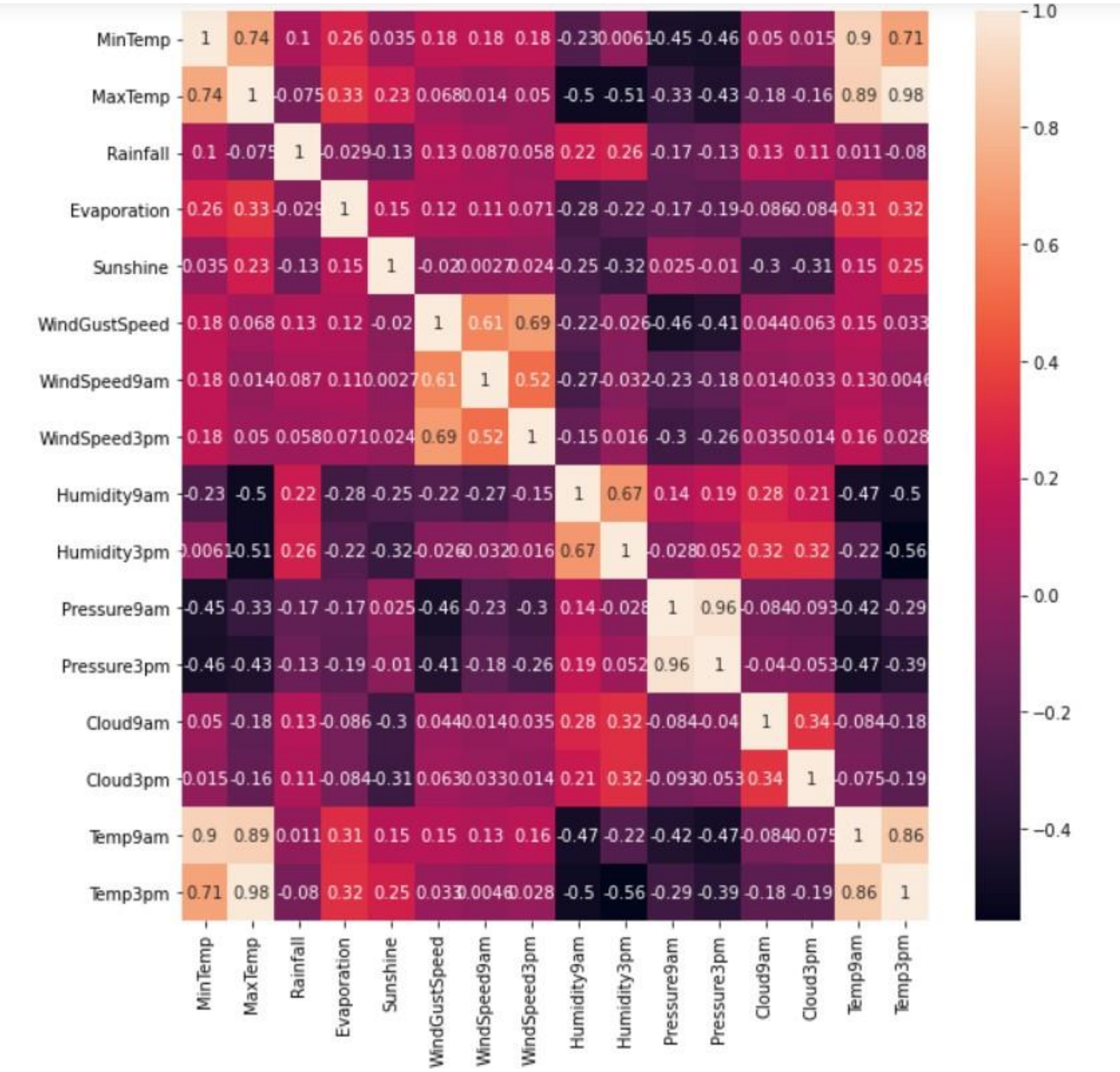
Here, random sampling is done for the variables - "Cloud9am", "Cloud3pm", "Evaporation" and "Sunshine".

```
corrmat = df.corr()
corrmat
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressur
MinTemp	1.000000	0.736555	0.103938	0.264243	0.034965	0.177415	0.175064	0.175173	-0.232899	0.006089	-0.45
MaxTemp	0.736555	1.000000	-0.074992	0.330018	0.234574	0.067615	0.014450	0.050300	-0.504110	-0.508855	-0.33
Rainfall	0.103938	-0.074992	1.000000	-0.028819	-0.126713	0.133659	0.087338	0.057887	0.224405	0.255755	-0.16
Evaporation	0.264243	0.330018	-0.028819	1.000000	0.152067	0.115676	0.108256	0.071081	-0.284180	-0.219553	-0.17
Sunshine	0.034965	0.234574	-0.126713	0.152067	1.000000	-0.019548	0.002740	0.024379	-0.251279	-0.322692	0.02
WindGustSpeed	0.177415	0.067615	0.133659	0.115676	-0.019548	1.000000	0.605303	0.686307	-0.215070	-0.026327	-0.45
WindSpeed9am	0.175064	0.014450	0.087338	0.108256	0.002740	0.605303	1.000000	0.519547	-0.270858	-0.031614	-0.22
WindSpeed3pm	0.175173	0.050300	0.057887	0.071081	0.024379	0.686307	0.519547	1.000000	-0.145525	0.016432	-0.29
Humidity9am	-0.232899	-0.504110	0.224405	-0.284180	-0.251279	-0.215070	-0.270858	-0.145525	1.000000	0.666949	0.13
Humidity3pm	0.006089	-0.508855	0.255755	-0.219553	-0.322692	-0.026327	-0.031614	0.016432	0.666949	1.000000	-0.02
Pressure9am	-0.450970	-0.332061	-0.168154	-0.170029	0.025139	-0.458744	-0.228743	-0.296351	0.139442	-0.027544	1.00
Pressure3pm	-0.461292	-0.427167	-0.126534	-0.185339	-0.010022	-0.413749	-0.175817	-0.255439	0.186858	0.051997	0.96
Cloud9am	0.049552	-0.175388	0.134936	-0.085556	-0.299892	0.044177	0.014133	0.035089	0.277955	0.316882	-0.08
Cloud3pm	0.014919	-0.162153	0.112153	-0.083914	-0.309680	0.063227	0.032828	0.013988	0.211440	0.317951	-0.09
Temp9am	0.901821	0.887210	0.011192	0.311943	0.147188	0.150150	0.128545	0.163030	-0.471354	-0.221019	-0.42
Temp3pm	0.708906	0.984503	-0.079657	0.322375	0.249469	0.032748	0.004569	0.027778	-0.498399	-0.557841	-0.28

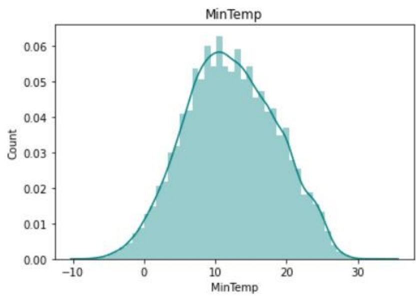
corrmat stores the correlation matrix of the dataframe df. The above output displays correlation matrix showing correlation coefficients between each of the numerical variables present in the dataset. These correlation coefficients convey the strength of the relationship between respective variables.

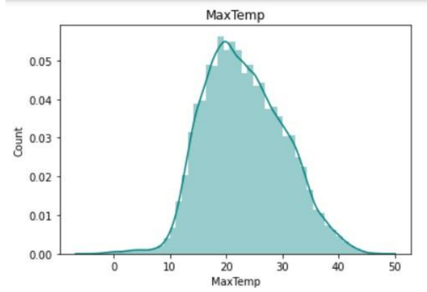
```
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(corrmat,annot=True)
```

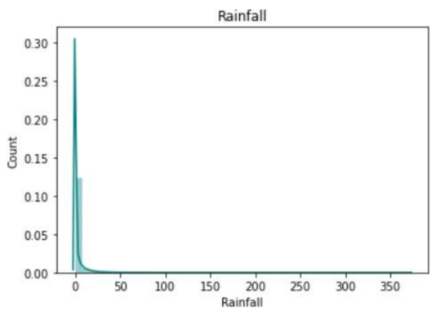


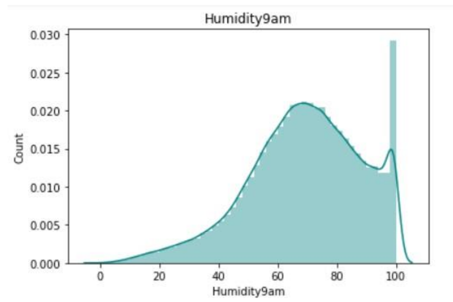
The above heatmap magnitude of correlation between the variables using a perceptually uniform color scale. Darker the color, more negative is the value and vice versa.

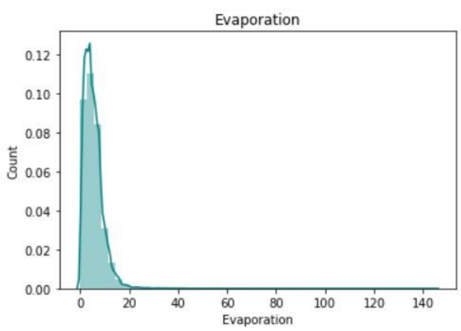
```python
for feature in continuous_feature:
    data=df.copy()
    sns.distplot(df[feature],color='teal')
    plt.xlabel(feature)
    plt.ylabel("Count")
    plt.title(feature)
    plt.figure(figsize=(15,15))
    plt.show()
```

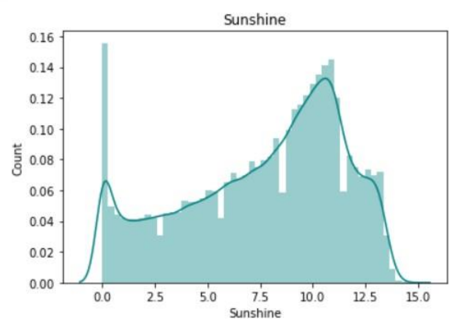


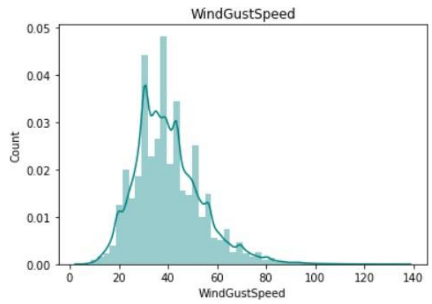


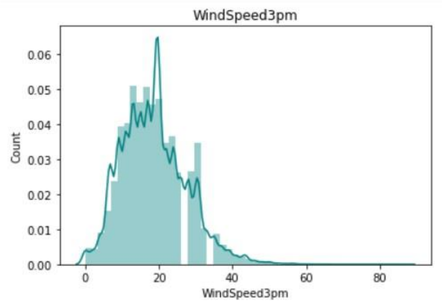






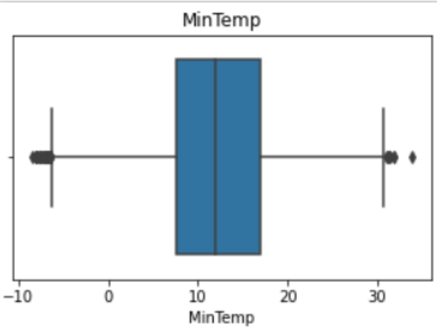


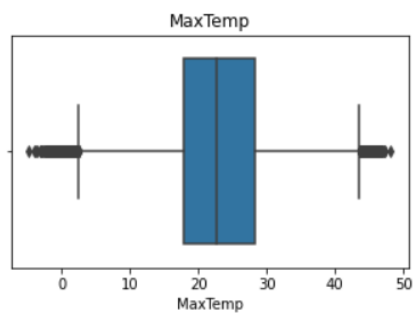


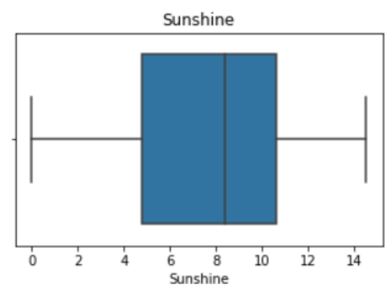


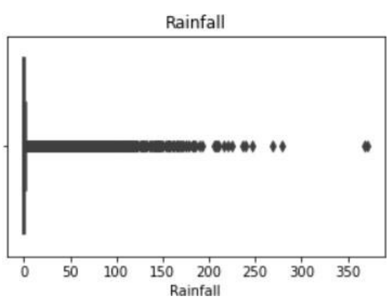
The line drawn over this histogram aids in visualization of the univariate distribution of data in the given column.

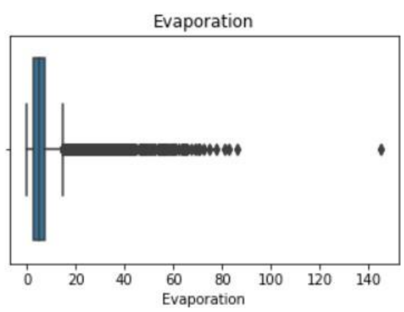
```
#A for loop is used to plot a boxplot for all the continuous features to see the outliers
for feature in continuous_feature:
    plt.figure(figsize=(5,3))
    data=df.copy()
    sns.boxplot(data[feature])
    plt.title(feature)
    plt.figure(figsize=(15,15))
```

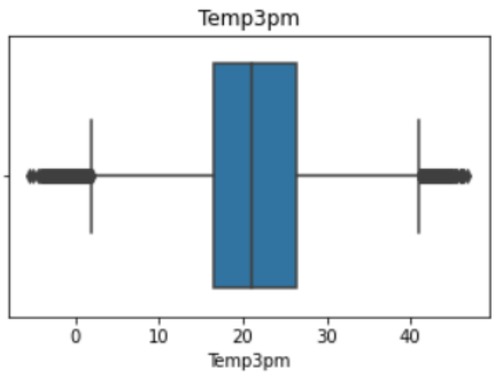


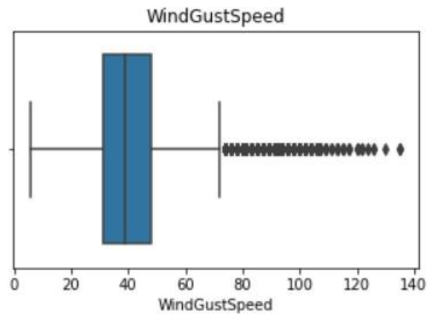


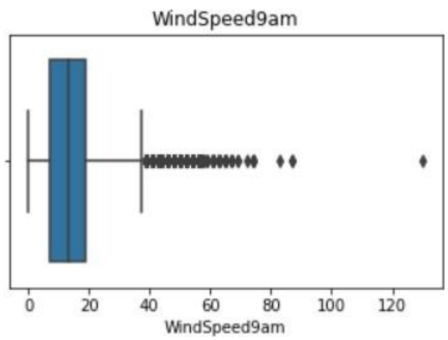


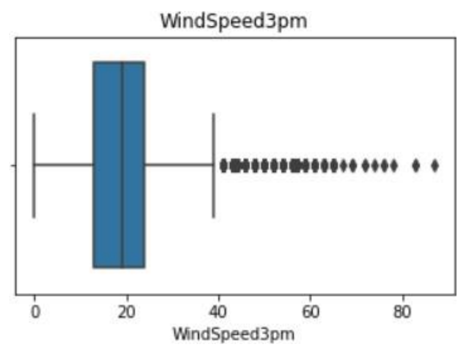












Before attempting to remove the outliers, these box plots have been plotted as they provide handy information of the outliers in a particular data subset.

Box plot marks five numerical data points: Minimum, First quartile, Median, Third quartile, Maximum. The minimum point (q1 - (1.5 * IQR)) and maximum point (q3 + (1.5 * IQR)) are the points beyond which outliers are present.

The above charts inform about the presence of outliers in a particular column along with the extent to which they outlie beyond the minimum and maximum point.

```python
for feature in continuous_feature:
    if(df[feature].isnull().sum()*100/len(df))>0:
        df[feature] = df[feature].fillna(df[feature].median())
```

```python
def mode_nan(df,variable):
    mode=df[variable].value_counts().index[0]
    df[variable].fillna(mode,inplace=True)
mode_nan(df,"Cloud9am")
mode_nan(df,"Cloud3pm")
```

mode_nan is a function which takes a column variable along with its dataframe as input to replace the missing values in the column with the mode of current present values in the same column.

Missing values of Cloud9am and Cloud3pm hence get replaced with the mode of the present values in respective columns.

```
df["RainToday"] = pd.get_dummies(df["RainToday"], drop_first = True)
df["RainTomorrow"] = pd.get_dummies(df["RainTomorrow"], drop_first = True)
```

Since the columns "RainToday" and "RainTomorrow" are categorical columns they have been converted into dummy variables.

```
windgustdir = {'NNW':0, 'NW':1, 'WNW':2, 'N':3, 'W':4, 'WSW':5, 'NNE':6, 'S':7, 'SSW':8, 'SW':9, 'SSE':10,
       'NE':11, 'SE':12, 'ESE':13, 'ENE':14, 'E':15}
winddir9am = {'NNW':0, 'N':1, 'NW':2, 'NNE':3, 'WNW':4, 'W':5, 'WSW':6, 'SW':7, 'SSW':8, 'NE':9, 'S':10,
       'SSE':11, 'ENE':12, 'SE':13, 'ESE':14, 'E':15}
winddir3pm = {'NW':0, 'NNW':1, 'N':2, 'WNW':3, 'W':4, 'NNE':5, 'WSW':6, 'SSW':7, 'S':8, 'SW':9, 'SE':10,
       'NE':11, 'SSE':12, 'ENE':13, 'E':14, 'ESE':15}
df["WindGustDir"] = df["WindGustDir"].map(windgustdir)
df["WindDir9am"] = df["WindDir9am"].map(winddir9am)
df["WindDir3pm"] = df["WindDir3pm"].map(winddir3pm)
```

In the three categorical columns WindGustDir, WindDir9am and WindDir9pm, each of the unique values in the column is mapped to a number. These numbers range from 0 to no_of_unique_values-1.

```
df.isnull().sum()*100/len(df)
```

```
Date             0.0
Location         0.0
MinTemp          0.0
MaxTemp          0.0
Rainfall         0.0
Evaporation      0.0
Sunshine         0.0
WindGustDir      0.0
WindGustSpeed    0.0
WindDir9am       0.0
WindDir3pm       0.0
WindSpeed9am     0.0
WindSpeed3pm     0.0
Humidity9am      0.0
Humidity3pm      0.0
Pressure9am      0.0
Pressure3pm      0.0
Cloud9am         0.0
Cloud3pm         0.0
Temp9am          0.0
Temp3pm          0.0
RainToday        0.0
RainTomorrow     0.0
dtype: float64
```

The above result shows the percentage of missing values for each column in the dataframe. All the values are 0 which indicates that all the missing values have been treated.

```
df1 = df.groupby(["Location"])["RainTomorrow"].value_counts().sort_values().unstack()
```

The dataframe df1 is assigned to contain the number of total values which indicate whether it will rain tomorrow or not. These values are stored location wise.

```
df1
```

RainTomorrow	0	1
Location		
Adelaide	2505	688
Albany	2138	902
Albury	2422	618
AliceSprings	2796	244
BadgerysCreek	2426	583
Ballarat	2259	781
Bendigo	2478	562
Brisbane	2484	709
Cairns	2090	950
Canberra	2807	629
Cobar	2623	386
CoffsHarbour	2140	869
Dartmoor	2087	922
Darwin	2341	852
GoldCoast	2265	775
Hobart	2432	761
Katherine	1313	265
Launceston	2341	699
Melbourne	2557	636
MelbourneAirport	2356	653
Moree	2615	394
MountGambier	2120	920
MountGinini	2221	819
Newcastle	2308	731
Nhil	1336	242
NorahHead	2196	808
NorfolkIsland	2090	919
Nuriootpa	2417	592
PearceRAAF	2504	505
Penrith	2444	595
Perth	2548	645
PerthAirport	2442	567
Portland	1914	1095
Richmond	2449	560
Sale	2366	643
SalmonGums	2529	472
Sydney	2479	865
SydneyAirport	2235	774
Townsville	2521	519
Tuggeranong	2471	568
Uluru	1462	116
WaggaWagga	2473	536
Walpole	2057	949
Watsonia	2271	738
Williamtown	2309	700
Witchcliffe	2130	879
Wollongong	2327	713
Woomera	2807	202

The above results display the number of results whether it will RainTomorrow location wise.1 indicates Yes (It will rain tomorrow) while 0 indicates No (It won't rain tomorrow).

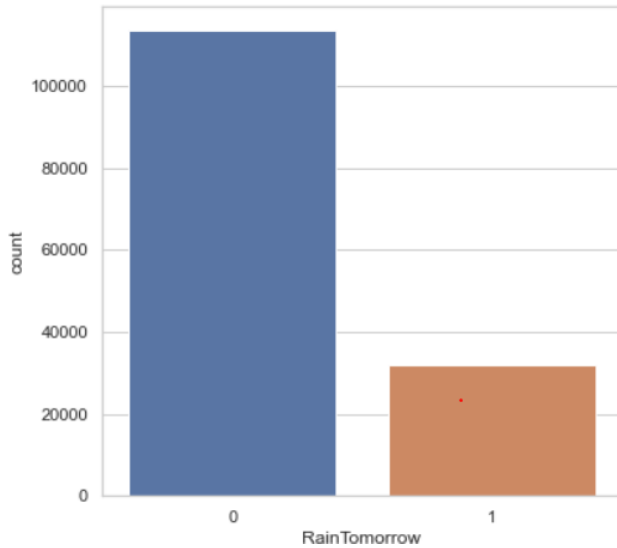
```
location = {'Portland':1, 'Cairns':2, 'Walpole':3, 'Dartmoor':4, 'MountGambier':5,
       'NorfolkIsland':6, 'Albany':7, 'Witchcliffe':8, 'CoffsHarbour':9, 'Sydney':10,
       'Darwin':11, 'MountGinini':12, 'NorahHead':13, 'Ballarat':14, 'GoldCoast':15,
       'SydneyAirport':16, 'Hobart':17, 'Watsonia':18, 'Newcastle':19, 'Wollongong':20,
       'Brisbane':21, 'Williamtown':22, 'Launceston':23, 'Adelaide':24, 'MelbourneAirport':25,
       'Perth':26, 'Sale':27, 'Melbourne':28, 'Canberra':29, 'Albury':30, 'Penrith':31,
       'Nuriootpa':32, 'BadgerysCreek':33, 'Tuggeranong':34, 'PerthAirport':35, 'Bendigo':36,
       'Richmond':37, 'WaggaWagga':38, 'Townsville':39, 'PearceRAAF':40, 'SalmonGums':41,
       'Moree':42, 'Cobar':43, 'Mildura':44, 'Katherine':45, 'AliceSprings':46, 'Nhil':47,
       'Woomera':48, 'Uluru':49}
df["Location"] = df["Location"].map(location)
```

Each of the 49 locations have been mapped to an integer for further convenience. These numbers range from 1 to 49.

```
df["Date"] = pd.to_datetime(df["Date"], format = "%Y-%m-%dT", errors = "coerce")
df["Date_month"] = df["Date"].dt.month
df["Date_day"] = df["Date"].dt.day
```

Two additional columns are created for the dataset which contain the day and month to which the record belongs. These values are stored in the column "Date_month" and "Date_day" respectively.

```
sns.countplot(df["RainTomorrow"])
```



The above plot simply plots the count of the record "RainTomorrow". 0 signifies that it won't rain tomorrow while 1 signifies it will rain tomorrow.

```
IQR=df.MinTemp.quantile(0.75)-df.MinTemp.quantile(0.25)
lower_bridge=df.MinTemp.quantile(0.25)-(IQR*1.5)
upper_bridge=df.MinTemp.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
-5.950000000000002 30.450000000000003
```

```
df.loc[df['MinTemp']>=30.45,'MinTemp']=30.45
df.loc[df['MinTemp']<=-5.95,'MinTemp']=-5.95
```

```
IQR=df.MaxTemp.quantile(0.75)-df.MaxTemp.quantile(0.25)
lower_bridge=df.MaxTemp.quantile(0.25)-(IQR*1.5)
upper_bridge=df.MaxTemp.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
2.700000000000001 43.5
```

```
df.loc[df['MaxTemp']>=43.5,'MaxTemp']=43.5
df.loc[df['MaxTemp']<=2.7,'MaxTemp']=2.7
```

```
IQR=df.Rainfall.quantile(0.75)-df.Rainfall.quantile(0.25)
lower_bridge=df.Rainfall.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Rainfall.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
-0.8999999999999999 1.5
```

```
df.loc[df['Rainfall']>=1.5,'Rainfall']=1.5
df.loc[df['Rainfall']<=-0.89,'Rainfall']=-0.89
```

```
IQR=df.Evaporation.quantile(0.75)-df.Evaporation.quantile(0.25)
lower_bridge=df.Evaporation.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Evaporation.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
-4.600000000000001 14.600000000000001
```

```
df.loc[df['Evaporation']>=14.6,'Evaporation']=14.6
df.loc[df['Evaporation']<=-4.6,'Evaporation']=-4.6
```

```
IQR=df.WindGustSpeed.quantile(0.75)-df.WindGustSpeed.quantile(0.25)
lower_bridge=df.WindGustSpeed.quantile(0.25)-(IQR*1.5)
upper_bridge=df.WindGustSpeed.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
8.5 68.5
```

```
df.loc[df['WindGustSpeed']>=68.5,'WindGustSpeed']=68.5
df.loc[df['WindGustSpeed']<=8.5,'WindGustSpeed']=8.5
```

```
IQR=df.WindSpeed9am.quantile(0.75)-df.WindSpeed9am.quantile(0.25)
lower_bridge=df.WindSpeed9am.quantile(0.25)-(IQR*1.5)
upper_bridge=df.WindSpeed9am.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
-11.0 37.0
```

```
df.loc[df['WindSpeed9am']>=37,'WindSpeed9am']=37
df.loc[df['WindSpeed9am']<=-11,'WindSpeed9am']=-11
```

```
IQR=df.WindSpeed3pm.quantile(0.75)-df.WindSpeed3pm.quantile(0.25)
lower_bridge=df.WindSpeed3pm.quantile(0.25)-(IQR*1.5)
upper_bridge=df.WindSpeed3pm.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
-3.5 40.5
```

```
df.loc[df['WindSpeed3pm']>40.5,'WindSpeed3pm']=40.5
df.loc[df['WindSpeed3pm']<=-3.5,'WindSpeed3pm']=-3.5
```

```
IQR=df.Humidity9am.quantile(0.75)-df.Humidity9am.quantile(0.25)
lower_bridge=df.Humidity9am.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Humidity9am.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
18.0 122.0
```

```
df.loc[df['Humidity9am']>=122,'Humidity9am']=122
df.loc[df['Humidity9am']<=18,'Humidity9am']=18
```

```
IQR=df.Pressure9am.quantile(0.75)-df.Pressure9am.quantile(0.25)
lower_bridge=df.Pressure9am.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Pressure9am.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
1001.0500000000001 1034.25
```

```
df.loc[df['Pressure9am']>=1034.25,'Pressure9am']=1034.25
df.loc[df['Pressure9am']<=1001.05,'Pressure9am']=1001.05
```

```
IQR=df.Pressure3pm.quantile(0.75)-df.Pressure3pm.quantile(0.25)
lower_bridge=df.Pressure3pm.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Pressure3pm.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
998.6500000000001 1031.85
```

```
df.loc[df['Pressure3pm']>=1031.85,'Pressure3pm']=1031.85
df.loc[df['Pressure3pm']<=998.65,'Pressure3pm']=998.65
```

```
IQR=df.Temp9am.quantile(0.75)-df.Temp9am.quantile(0.25)
lower_bridge=df.Temp9am.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Temp9am.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
-1.4999999999999982 35.3
```

```
df.loc[df['Temp9am']>=35.3,'Temp9am']=35.3
df.loc[df['Temp9am']<=-1.49,'Temp9am']=-1.49
```

```
IQR=df.Temp3pm.quantile(0.75)-df.Temp3pm.quantile(0.25)
lower_bridge=df.Temp3pm.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Temp3pm.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
2.4499999999999993 40.45
```

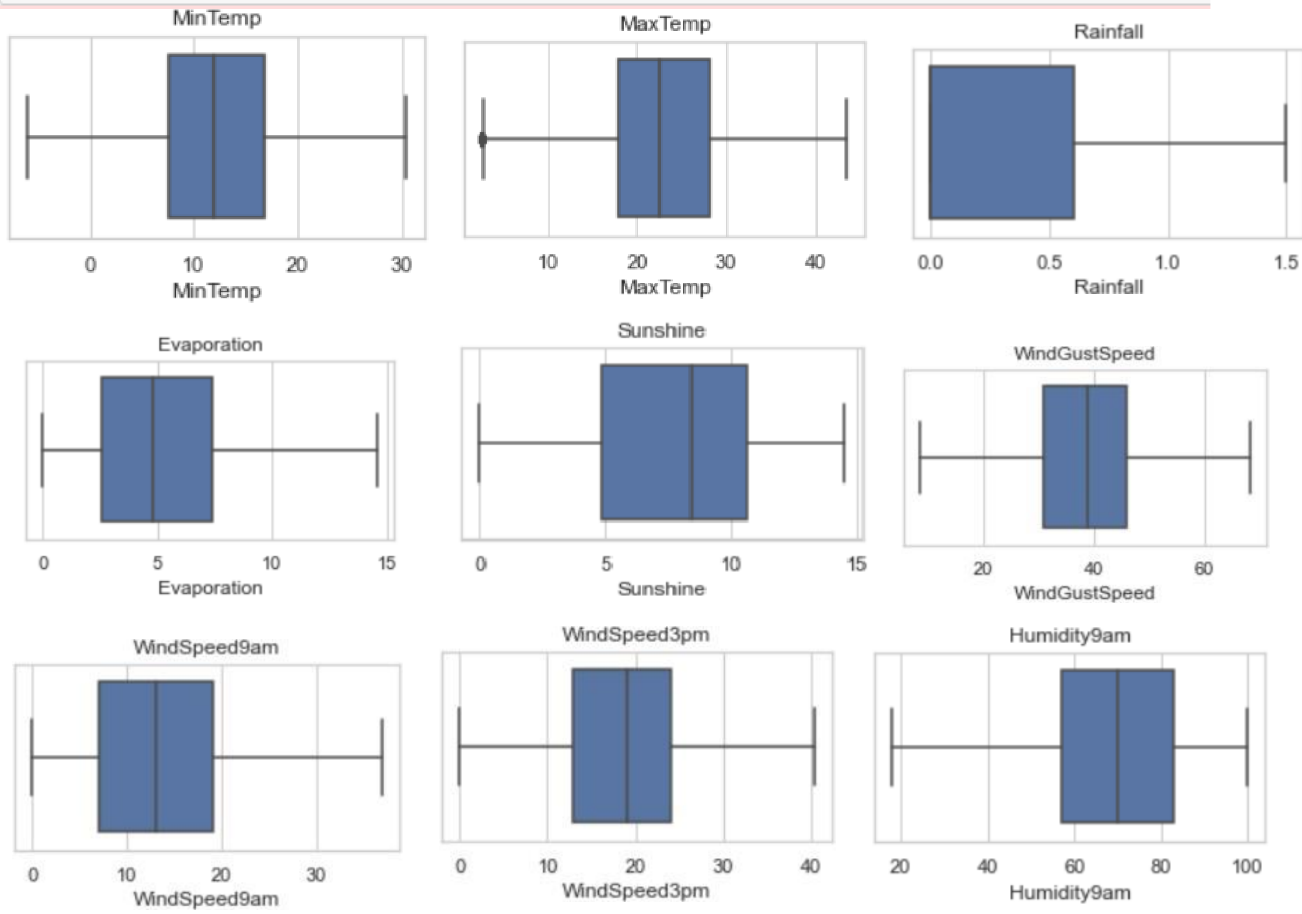
```
df.loc[df['Temp3pm']>=40.45,'Temp3pm']=40.45
df.loc[df['Temp3pm']<=2.45,'Temp3pm']=2.45
```

As seen in one of the previous plotted graphs, the dataset contains many outliers which would withhold the accuracy of the prediction. In order to improve the accuracy, these outliers need to be treated.

upper_bridge and lower_bridge are calculated. For each of the above variables, these values contain what we refer to the maximum and minimum point in the box plot. As indicated by the name, they mark the minimum value and maximum value which further define the range in which the values are to be present for the variable.

Consequently, we calculate upper_bridge and lower_bridge for each of the variables. All the outliers beyond the maximum point are reassigned the value contained in upper_bridge while those lying beyond minimum point are assigned the value of lower_bridge.

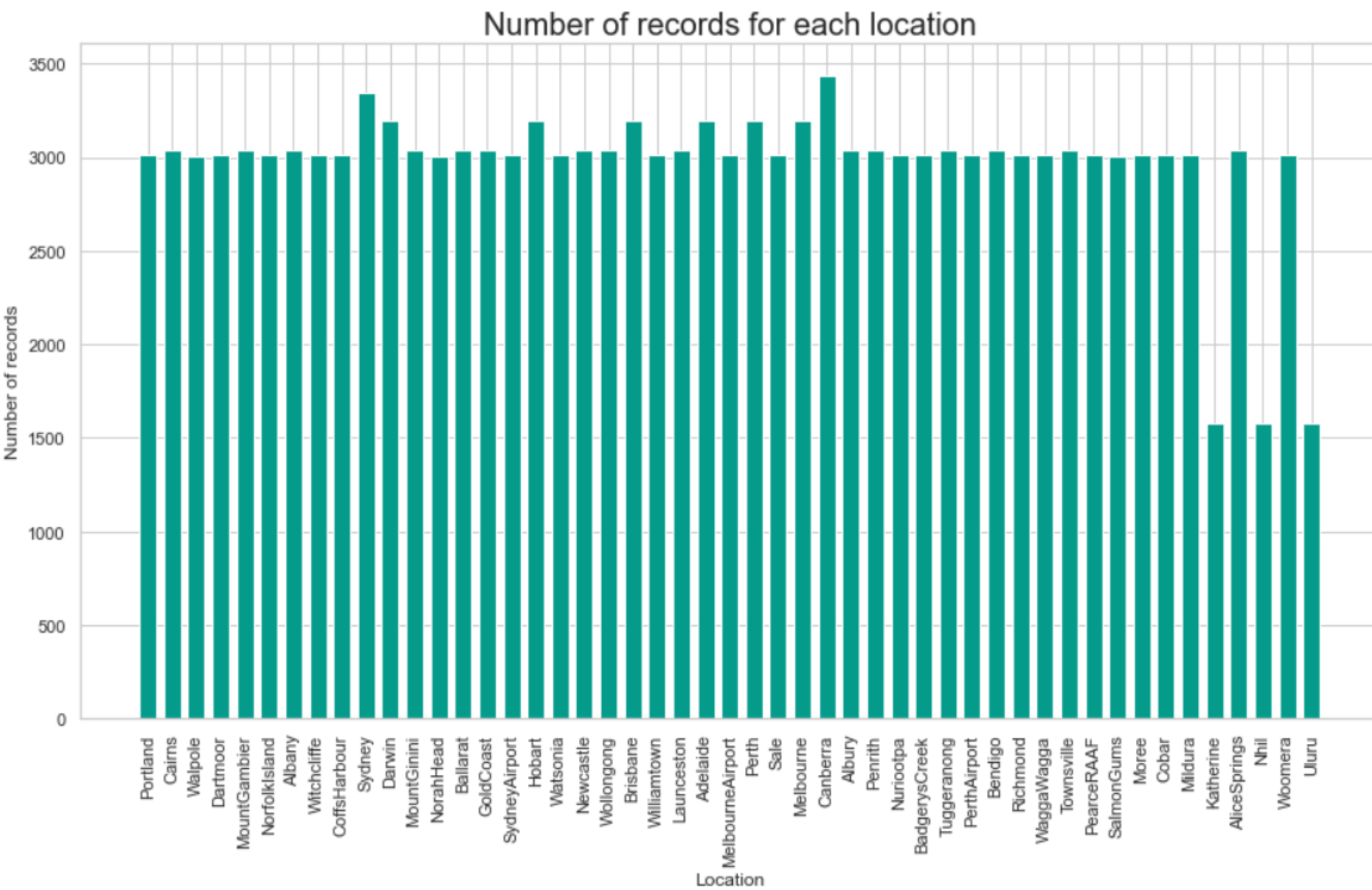
```
for feature in continuous_feature:
    data=df.copy()
    sns.boxplot(data[feature])
    plt.title(feature)
    plt.figure(figsize=(15,15))
```



The above plots clearly indicate that all the outliers have been removed(treated) as there are no points to the left and right of minimum and maximum point respectively.

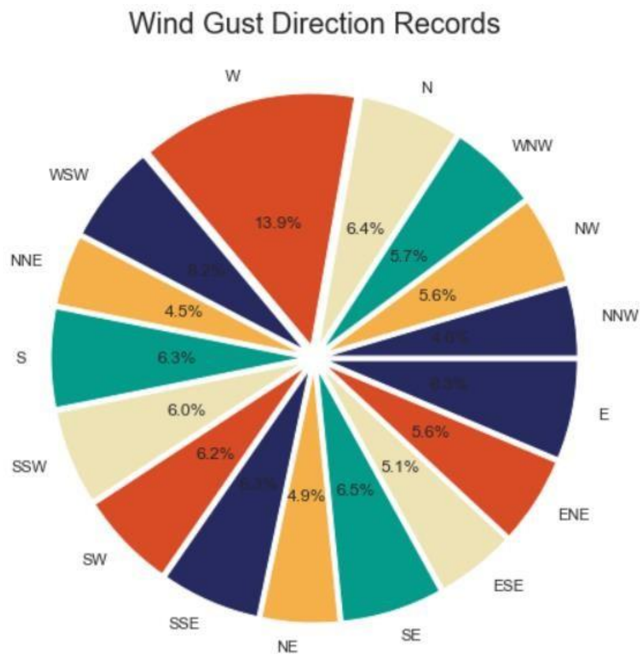
Data Visualization:

```python
plt.rcParams["figure.figsize"] = (15,8)

colorPalette=["#272961","#F5B049","#059B8B","#EDE3B4","#D84B25"]
x=list(locations.values())
print(x)
dictCount=dict(df.groupby("Location")["Rainfall"].count())
y=list(dictCount.values())
print(y)
plt.xticks(rotation=90)
plt.bar(x, y, color =colorPalette[2],width = 0.7)
plt.xlabel("Location",fontsize=12)
plt.ylabel("Number of records",fontsize=12)
plt.title("Number of records for each location",fontsize=20)
```



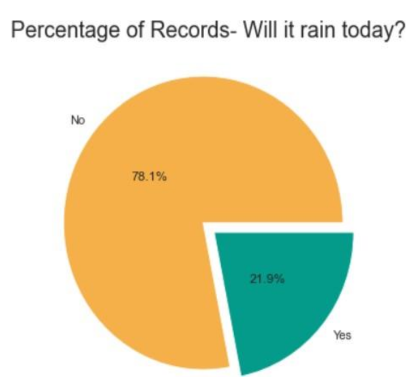
```
windgustdir = {'NNW':0, 'NW':1, 'WNW':2, 'N':3, 'W':4, 'WSW':5, 'NNE':6, 'S':7, 'SSW':8, 'SW':9, 'SSE':10,
        'NE':11, 'SE':12, 'ESE':13, 'ENE':14, 'E':15}
```

```
plt.rcParams["figure.figsize"] = (8,8)
pieData=df.groupby("WindGustDir")["WindGustDir"].count()
labels=windgustdir.keys()
plt.pie(x=pieData, autopct="%.1f%%", explode=[0.05]*len(pieData), labels=labels, pctdistance=0.5,colors=colorPalette)
plt.title("Wind Gust Direction Records",fontsize=20)
```



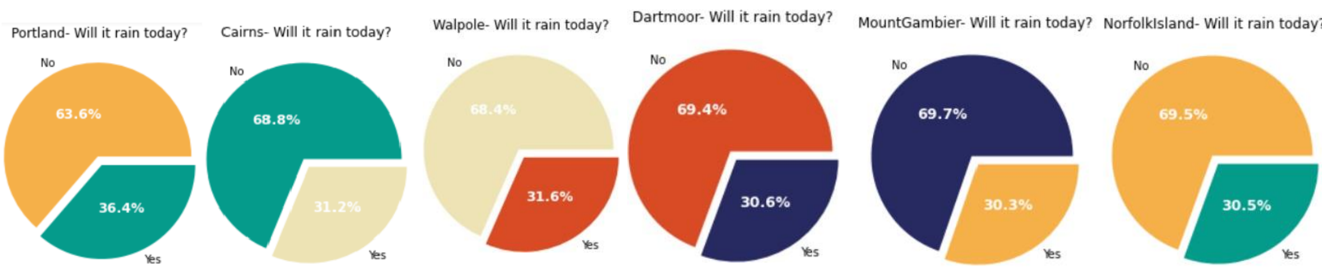
The plot indicates the percentage of each record as stored in the dataset by the column "WindGustDir". The labels are the unique values found in the column and indicate the direction of wind gust.

```
plt.rcParams["figure.figsize"] = (6,6)
pieData=df.groupby("RainToday")["RainToday"].count()
pieData
labels=["No","Yes"]
plt.pie(x=pieData, autopct="%.1f%%", explode=[0.05]*len(pieData),labels=labels, pctdistance=0.5,colors=colorPalette[1:])
plt.title("Percentage of Records- Will it rain today?",fontsize=20)
plt.show()
```



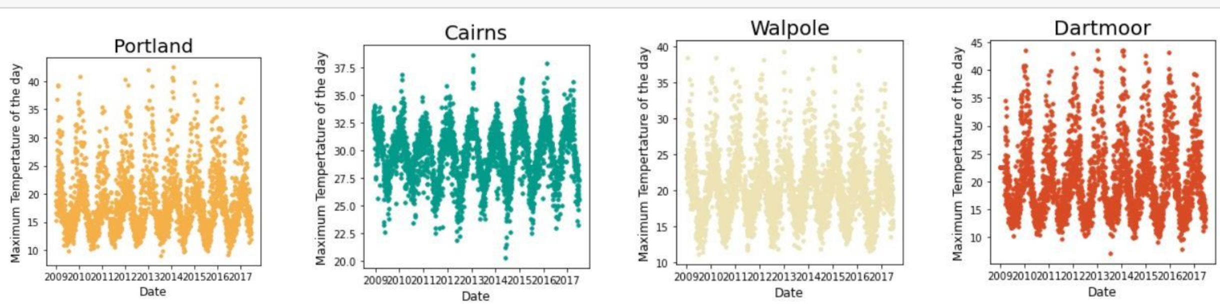
The percentage of records that hold the result of whether it would rain today or not is shown by the above plot.
It is worth noting that the percentage of data that contain the result “Yes” is very less than the percentage of data that contain “No” as the result.

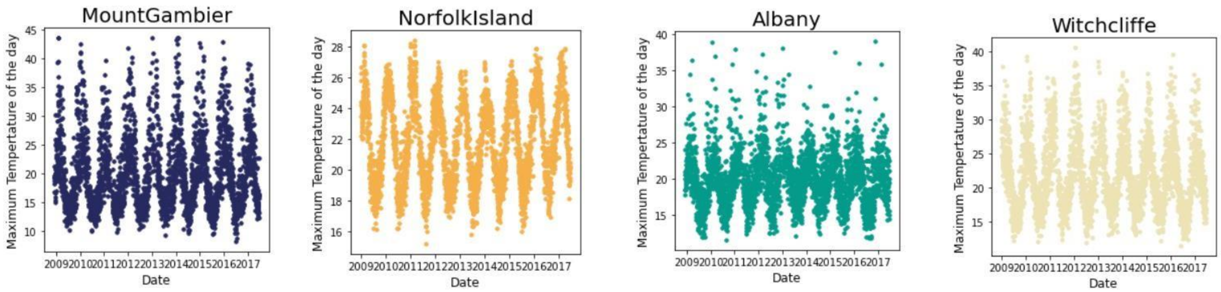
```
plt.rcParams["figure.figsize"] = (6,6)
for i in range(1,50):
    pieData=df[(df["Location"]==i)].groupby("RainToday")["RainToday"].count()
    labels=["No","Yes"]
    _, _, autopcts=plt.pie(x=pieData, autopct="%.1f%%", explode=[0.05]*len(pieData),labels=labels, pctdistance=0.5,colors=[colorP
    plt.setp(autopcts, **{'color':'white', 'weight':'bold', 'fontsize':12.5})
    plt.title(locations[i]+"- Will it rain today?",fontsize=18)
    plt.show()
```



The code snippet plots percentage of rain records for **each location.**
For each location, the count of the number of records whether it will rain today or not is found out.
For every location, the records which indicate the surety of rainfall today are very less compared to the contrary. Hence, this has to be taken care of before evaluation.

```
plt.rcParams["figure.figsize"] = (4,4)

for i in range(1,50):
    plt.scatter(df[(df["Location"]==i)]["Date"],df[(df["Location"]==i)]["MaxTemp"],color=colorPalette[i%5],s=12)
    plt.xlabel("Date",fontsize=12)
    plt.ylabel("Maximum Tempertature of the day",fontsize=12)
    plt.title(locations[i],fontsize=20)
    plt.show()
```

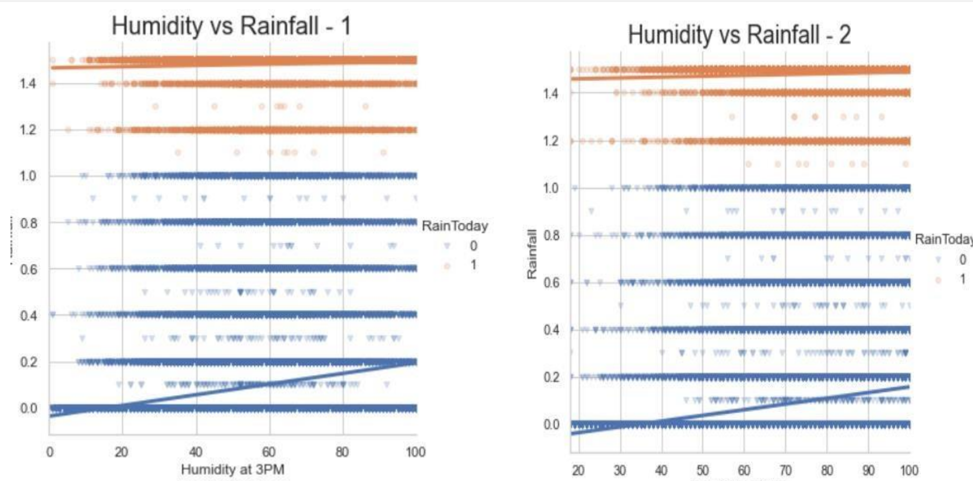




To get an overview how uniform is the record we have, we plot maximum temperature against the day on which it is produced. These plots are produced for **all 49 locations.**
The plots specifies uniformity of data which further indicates its accuracy.
The points high above may indicate summer season while the lower ones may indicate winter season.

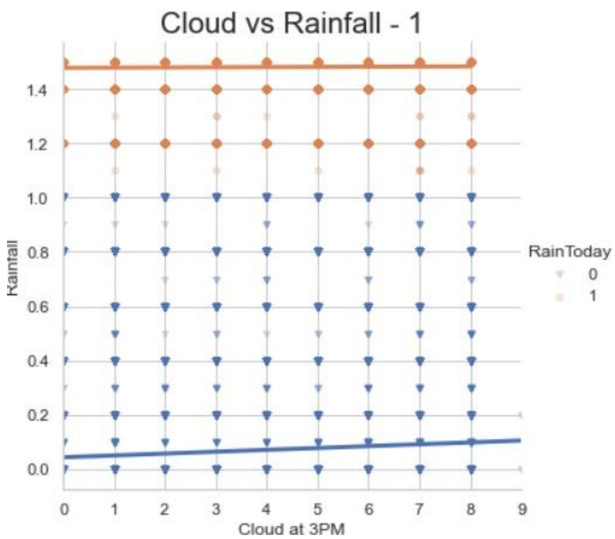
<u>Regression Analysis:</u>

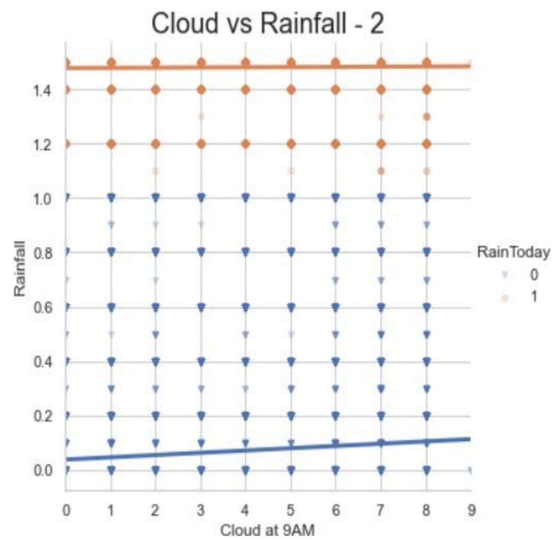
```
sns.set(rc={'figure.figsize':(20,15)})
sns.set_style('whitegrid')
sns.color_palette("hls", 8)
sns.lmplot(x ='Humidity3pm', y ='Rainfall', data = df,hue="RainToday" ,markers =['v', 'o'], scatter_kws ={'s':20, 'alpha': 0.2},
plt.xlabel("Humidity at 3PM",fontsize=12)
plt.ylabel("Rainfall",fontsize=12)
plt.title("Humidity vs Rainfall - 1",fontsize=20)
plt.show()

sns.lmplot(x ='Humidity9am', y ='Rainfall', data = df,hue="RainToday" ,markers =['v', 'o'], scatter_kws ={'s':20, 'alpha': 0.2},
plt.xlabel("Humidity at 9AM",fontsize=12)
plt.ylabel("Rainfall",fontsize=12)
plt.title("Humidity vs Rainfall - 2",fontsize=20)
plt.show()
```



These graphs plot the regression lines between Humidity and Rainfall. The "hue" parameter helps plot these for two differently recorded data based on:
It will rain today
It won't rain today

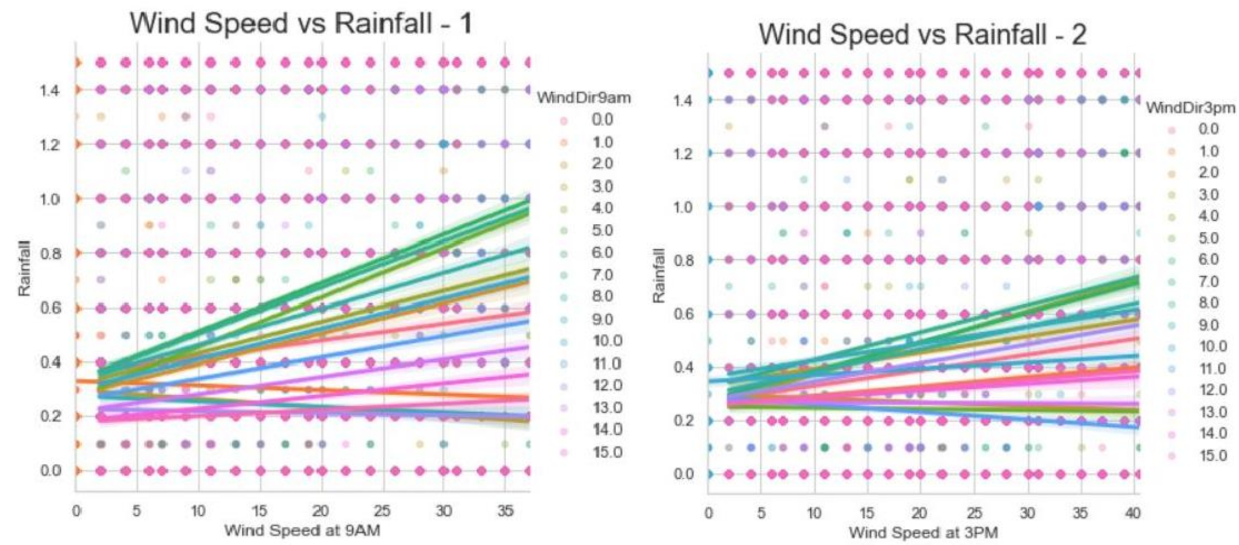
```
sns.lmplot(x ='Cloud3pm', y ='Rainfall', data = df,hue="RainToday" ,markers =['v', 'o'], scatter_kws ={'s':20, 'alpha': 0.2},line

plt.xlabel("Cloud at 3PM",fontsize=12)
plt.ylabel("Rainfall",fontsize=12)
plt.title("Cloud vs Rainfall - 1",fontsize=20)
plt.show()

sns.lmplot(x ='Cloud9am', y ='Rainfall', data = df,hue="RainToday" ,markers =['v', 'o'], scatter_kws ={'s':20, 'alpha': 0.2},line

plt.xlabel("Cloud at 9AM",fontsize=12)
plt.ylabel("Rainfall",fontsize=12)
plt.title("Cloud vs Rainfall - 2",fontsize=20)
plt.show()
```





The graph plots the regression line between cloud cover and rainfall. The data plotted has been divided in the same way as mentioned in the last graph.
The plot indicates a greater cloud cover would bring more rainfall in general.

```
sns.lmplot(x ='WindSpeed9am', y ='Rainfall', data = df,hue="WindDir9am" , scatter_kws ={'s':20, 'alpha': 0.3},line_kws={'lw': 3})
plt.xlabel("Wind Speed at 9AM",fontsize=12)
plt.ylabel("Rainfall",fontsize=12)
plt.title("Wind Speed vs Rainfall - 1",fontsize=20)
plt.show()

sns.lmplot(x ='WindSpeed3pm', y ='Rainfall', data = df,hue="WindDir3pm" , scatter_kws ={'s':20, 'alpha': 0.3},line_kws={'lw': 3})
plt.xlabel("Wind Speed at 3PM",fontsize=12)
plt.ylabel("Rainfall",fontsize=12)
plt.title("Wind Speed vs Rainfall - 2",fontsize=20)
plt.show()
```



The above graph marks the regression lines between wind speed and rainfall during different times of the day. The first one keeps track of morning data while the second one records afternoon data.
Different colors indicate different directions.
It is also observed that directions in which the wind blows also play an important role along with its speed. Some wind directions contribute to positive slope while some to negative.

MODELLING AND EVALUATION

```
X = df.drop(["RainTomorrow", "Date"], axis=1)
Y = df["RainTomorrow"]
```

The variable X contains the dataframe which holds the records based on which prediction is to be made. Y contains the dataframe from which values are to be predicted: RainTomorrow.
The columns "RainTomorrow" and "Date" are dropped from the dataframe contained in X as "RainTomorrow" is the variable to be predicted and "Date" has been previously added to the dataframe seperately in the form of day and month.

```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size =0.2, stratify = Y, random_state = 0)
```

The dataset is split into train dataset and test dataset. 80% of the data is now part of train dataset leaving the rest for test dataset.
The parameter "Stratify" makes sure that the proportion of values in the sample will be the same as the proportion of values in Y, i.e. the proportion of the different values of RainTomorrow is same in test and train dataset.

```
sm=SMOTE(random_state=0)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)
print("The number of classes before fit {}".format(Counter(y_train)))
print("The number of classes after fit {}".format(Counter(y_train_res)))
```

```
The number of classes before fit Counter({0: 90866, 1: 25502})
The number of classes after fit Counter({0: 90866, 1: 90866})
```

The above snippet helps convert the imbalanced dataset to a balanced one by oversampling the minority class.
SMOTE stands for Synthetic Minority Oversampling TEchnique. Specifically, a random example from the minority class is first chosen. Then k of the nearest neighbors for that example are found (typically k=5). A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point. The adds more classes to the minority class such that the number of classes become equal.

(a) CatBoost

```
cat = CatBoostClassifier(iterations=2000, eval_metric = "AUC")
cat.fit(X_train_res, y_train_res)
```

```
Learning rate set to 0.050311
0:      total: 101ms    remaining: 3m 21s
1:      total: 182ms    remaining: 3m 2s
2:      total: 270ms    remaining: 2m 59s
3:      total: 325ms    remaining: 2m 42s
4:      total: 370ms    remaining: 2m 27s
5:      total: 417ms    remaining: 2m 18s
6:      total: 464ms    remaining: 2m 11s
7:      total: 506ms    remaining: 2m 5s
8:      total: 550ms    remaining: 2m 1s
9:      total: 598ms    remaining: 1m 58s
10:     total: 653ms    remaining: 1m 58s
11:     total: 699ms    remaining: 1m 55s
12:     total: 742ms    remaining: 1m 53s
13:     total: 798ms    remaining: 1m 53s
14:     total: 841ms    remaining: 1m 51s
15:     total: 890ms    remaining: 1m 50s
16:     total: 941ms    remaining: 1m 49s
17:     total: 981ms    remaining: 1m 48s
```

This snippet runs for 2000 iterations each time training our model

CatBoostClassifier is used for training and applying problems for classification problems.
fit is the method of the class CatBoostClassifier which is used to train a model.

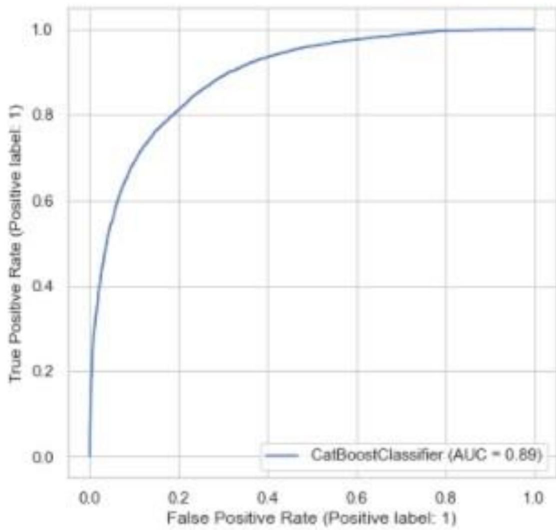
```
y_pred = cat.predict(X_test)
print('Confusion Matrix: -')
print(confusion_matrix(y_test,y_pred))
print('Accuracy score: -')
print(accuracy_score(y_test,y_pred))
print('Classification report: -')
print(classification_report(y_test,y_pred))
```

```
Confusion Matrix: -
[[21506  1211]
 [ 2800  3575]]
Accuracy score: -
0.8621270452358036
Classification report: -
              precision    recall  f1-score   support

           0       0.88      0.95      0.91     22717
           1       0.75      0.56      0.64      6375

    accuracy                           0.86     29092
   macro avg       0.82      0.75      0.78     29092
weighted avg       0.85      0.86      0.85     29092
```

The above output displays the quality of prediction as predicted by the algorithm.

```
plt.rcParams["figure.figsize"] = (6,6)
metrics.plot_roc_curve(cat, X_test, y_test)
metrics.roc_auc_score(y_test, y_pred, average=None)
```

```
0.7537381092332166
```



The above ROC curve shows performance of the CatBoost Classifier as it serves as a probability curve.
The label indicates the Area under Curve and has a high value of 0.89. Higher the area under is the curve, better is the prediction made by the classifier.

(b) Random Forest

```
rf=RandomForestClassifier()
rf.fit(X_train_res,y_train_res)
```

```
RandomForestClassifier()
```

Here, we use Random Forest Classifier to train our model. It uses a number of decision tree classifiers on several sub samples of dataset and uses average.

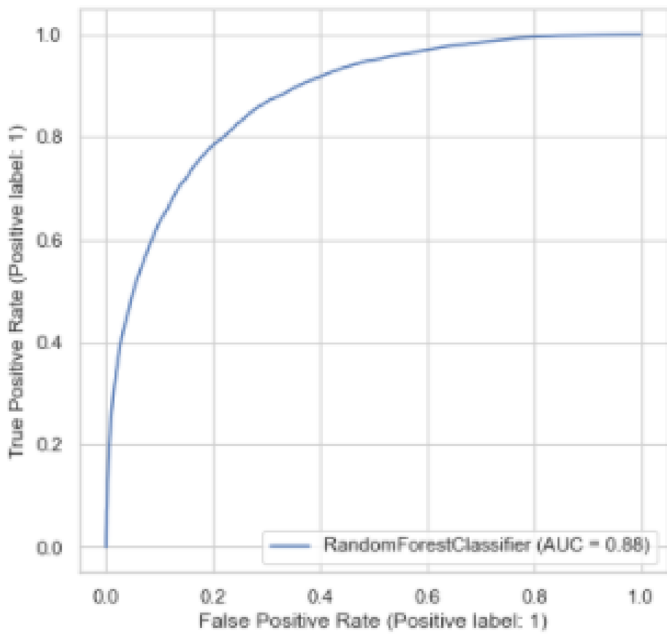
```
y_pred1 = rf.predict(X_test)
print('Confusion Matrix: -')
print(confusion_matrix(y_test,y_pred1))
print('Accuracy score: -')
print(accuracy_score(y_test,y_pred1))
print('Classification report: -')
print(classification_report(y_test,y_pred1))
```

```
Confusion Matrix: -
[[20633  2084]
 [ 2457  3918]]
Accuracy score: -
0.8439089784133095
Classification report: -
              precision    recall  f1-score   support

           0       0.89      0.91      0.90     22717
           1       0.65      0.61      0.63      6375

    accuracy                           0.84     29092
   macro avg       0.77      0.76      0.77     29092
weighted avg       0.84      0.84      0.84     29092
```

The report displays the quality of prediction made by Random Forest Classifier.

```
metrics.plot_roc_curve(rf, X_test, y_test)
metrics.roc_auc_score(y_test, y_pred1, average=None)
```

```
0.7614253849798933
```



It follows up that the area under the ROC curve of the prediction made by the Random Forest Classifier is 0.88.

```
logreg = LogisticRegression()
logreg.fit(X_train_res, y_train_res)
```

```
c:\python\python38\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning:
s=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression()
```

Logistic Regression is used to train the model. It uses a logistic function to model binary dependent variables.

Logistic Regression is used to train the model. It uses a logistic function to model binary dependent variables.

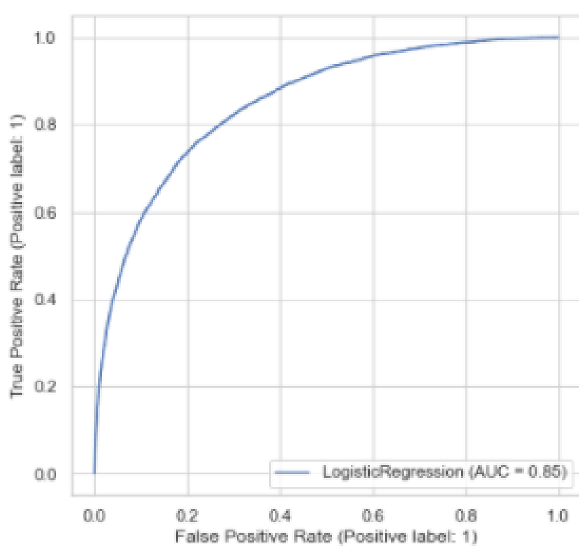
```
y_pred2 = logreg.predict(X_test)
print('Confusion Matrix: -')
print(confusion_matrix(y_test,y_pred2))
print('Accuracy score: -')
print(accuracy_score(y_test,y_pred2))
print('Classification report: -')
print(classification_report(y_test,y_pred2))
```

```
Confusion Matrix: -
[[17626  5091]
 [ 1514  4861]]
Accuracy score: -
0.7729616389385398
Classification report: -
              precision    recall  f1-score   support

           0       0.92      0.78      0.84     22717
           1       0.49      0.76      0.60      6375

    accuracy                           0.77     29092
   macro avg       0.70      0.77      0.72     29092
weighted avg       0.83      0.77      0.79     29092
```

Here is the information about the prediction made by using Logistic Regression.

```
metrics.plot_roc_curve(logreg, X_test, y_test)
metrics.roc_auc_score(y_test, y_pred2, average=None)
```

```
0.7692022541639801
```



It is noted that the area under ROC curve is 0.85.

Out of the three models used, CatBoost classifier gives the best prediction result.