**                                                      Blog By : Ravinder Singh**

**Rain Prediction –Weather forecasting**

Weather forecasting is the application of science and technology to predict the conditions of the atmosphere for a given location and time. Weather forecasts are made by collecting quantitative data about the current state of the atmosphere at a given place and using meteorology to project how the atmosphere will change. Rain Dataset is to predict whether or not it will rain tomorrow. The Dataset contains about 10 years of daily weather observations of different locations in Australia. Here, predict two things:

**Problem Statement:  
Design a predictive model with the use of machine learning algorithms to predict how much rainfall could be there.**

**Dataset Description:**

Number of columns: 23

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.

WindGustDi r- 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 3p

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.

Cloud3pm -Fraction of sky obscured by cloud

Temp9am-Temperature (degrees C) at 9am

Temp3pm -Temperature (degrees C) at 3pm

RainToday -Boolean: 1 if precipitation (mm) in the 24 hours to 9am exceeds 1mm, otherwise 0

RainTomorrow -The amount of next day rain in mm. Used to create response variable . A kind of measure of the "risk".

Goal : Predict the amount of Rainfall

**Approach Used  to Solve In a Structured Manner**

**Step 1: Importing basic libraries**

Importing Libraries the we are going to use for the project pandas, numpy for Data Manipulation and seaborn and matplotlib for visualization of the Data

*import pandas as pd*

*import numpy as np*

*import seaborn as sns*

*import matplotlib.pyplot as plt*

*%matplotlib inline*

**Step 2:Reading our data**

Reading the csv file directly from the github link .Using the pandas command .

*rain\_data=pd.read\_csv("https://raw.githubusercontent.com/dsrscientist/dataset3/main/weatherAUS.csv)*

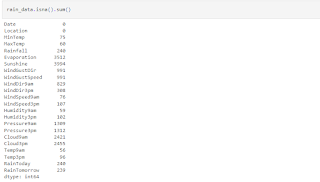
*rain\_data.head(2)*

[https://blogger.googleusercontent.com/img/a/AVvXsEhMVPpYSssNo_pL9RtO84W2koz_9SGZtec52pClzHCTEQYW1J5SNPBcubP6gkxnvKjCFUWQIrI8w2CR7WC0-4fviRl_9FQUk03UeIGvx27OjZhBrhwOgGBKgbVr7yevlfqv_8E6UkTULXCasOXmXp8hDFCBEZTx0vJWtqTEFOOqEV1ujhkXRKBltiDk=w320-h62](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

**Step 3: Missing Value  Imputation**

Checking for the missing values in the Data so that we could understand more about the data as there are many missing values we will impute the missing values first

*rain\_data.isnull().sum()*

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

Dropping rows with NAN values in Rainfall because as we have to find these value

*rain\_data.drop(rain\_data[rain\_data['Rainfall'].isna()].index,inplace=True)*

**Mapping**Mapping Categorial Columns so that every column that has the same values will have same code.We could have used label Encoder for this but that might have used 2 different code for the same values in different Columns

*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}*

*rain\_data["WindGustDir"] = rain\_data["WindGustDir"].map(windgustdir)*

*rain\_data["WindDir9am"] = rain\_data["WindDir9am"].map(winddir9am)*

*rain\_data["WindDir3pm"] = rain\_data["WindDir3pm"].map(winddir3pm)*

**Label Encoding**Label Encoding refers to converting the labels into a numeric form so as to convert them into the machine-readable form. Machine learning algorithms can then decide in a better way how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.Also It's done before the missing value imputation as mice works on Integer values not on the string values

*import LabelEncoder  
column=['Date\_month', 'Date\_Year','RainToday','Location','RainTomorrow']  
from sklearn.preprocessing import LabelEncoder  
for col in column:  
    le =LabelEncoder()  
    rain\_data[col]=le.fit\_transform(rain\_data[col].astype('str'))*

**Imputing the Missing Values**  
Iterative Imputer is a multivariate imputing strategy that models a column with the missing values (target variable) as a function of other features (predictor variables) in a round-robin fashion and uses that estimate for imputation.Its is a power-ful package and can save you some time but it increases the multi-collinearity in the data.(Multi-Collinearity discussed in the the later section)

*!pip install fancyimpute*

*importing the MICE from fancyimpute library*

*from fancyimpute import IterativeImputer*

Calling the  MICE Class

*mice\_imputer = IterativeImputer()*

 Imputing the missing value with Mice Imputer

*df= mice\_imputer.fit\_transform(rain\_data)*

*rain\_data=pd.DataFrame(data=mice\_imputer.fit\_transform(rain\_data), columns=rain\_data.columns, index=rain\_data.index)*

**Step 4: Data Exploration**

Data analysis utilising visual methods is called exploratory data analysis (EDA). With the use of statistical summaries and graphical representations, it is used to identify trends, patterns, or to verify assumptions.

**For Data Exploration Discrete Variable are countable infinite amount of time while numerical variable are to much in number to count**

*num\_var = [feature for feature in rain\_data.columns if rain\_data[feature].dtypes != 'O']*

*discrete\_var = [feature for feature in num\_var if len(rain\_data[feature].unique()) <= 25]*

*cont\_var = [feature for feature in num\_var if feature not in discrete\_var]*

*cat\_var = [feature for feature in rain\_data.columns if feature not in num\_var]*

**Distant Plot  for the various Continous Variable**

*for feature in cont\_var:*

*data=rain\_data.copy()*

*sns.distplot(data[feature])*

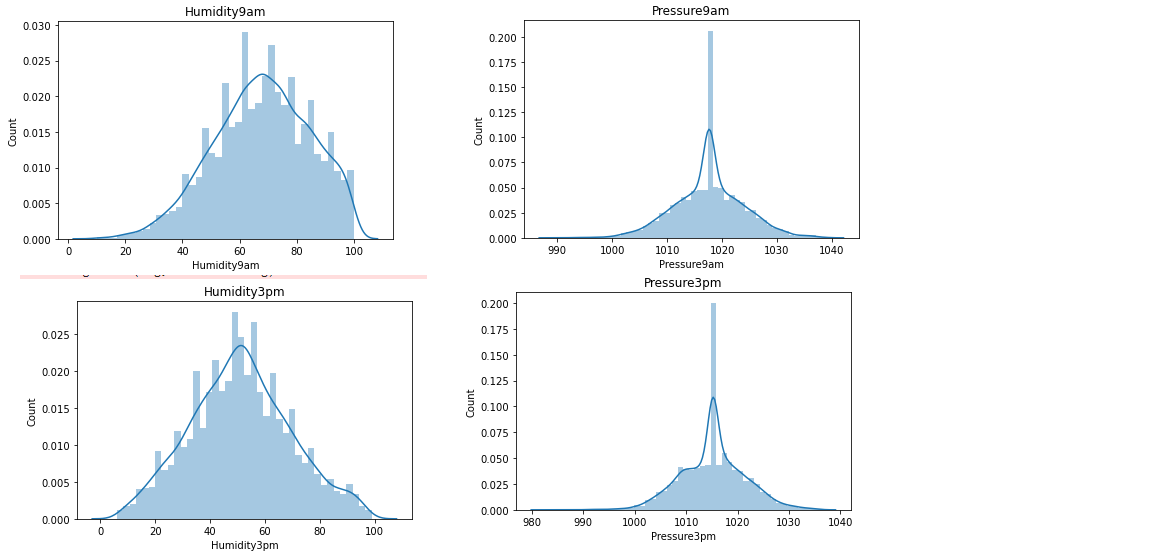
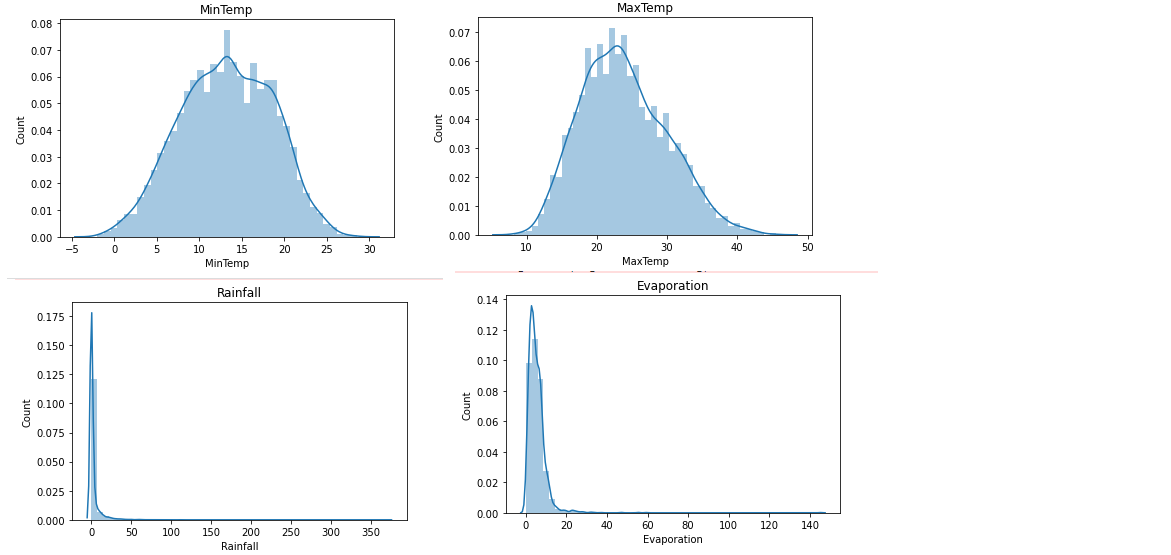
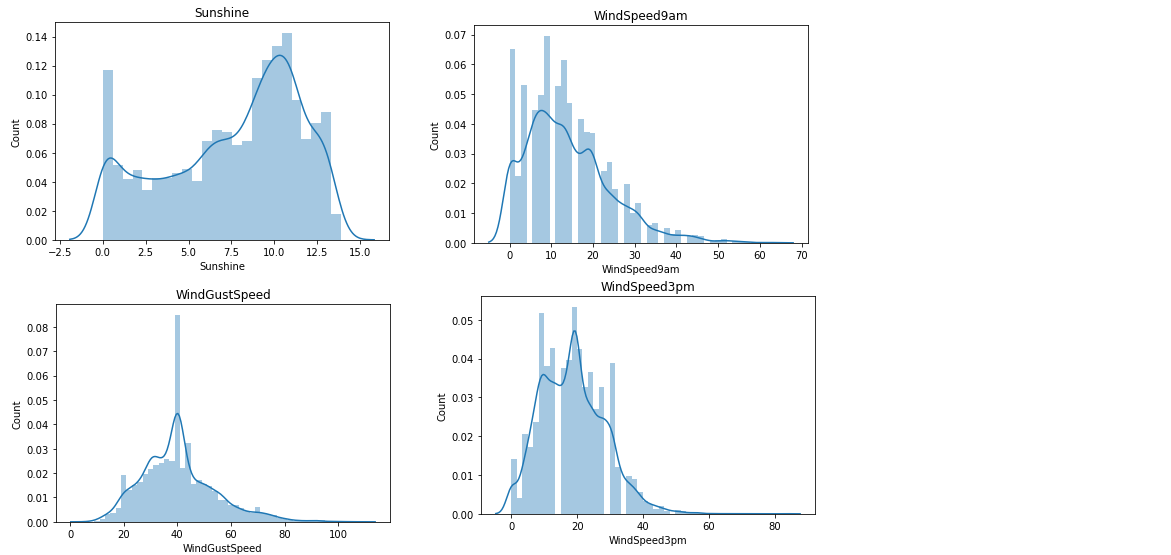
*plt.xlabel(feature)*

*plt.ylabel("Count")*

*plt.title(feature)*

*plt.figure(figsize=(15,15))*

*plt.show()*



**Plotting Q-Q Plot**

*import scipy.stats as stats*

*import pylab*

*def plot\_curve(df,feature):*

*plt.figure(figsize=(10,6))*

*plt.subplot(1,2,1)*

*df[feature].hist()*

*plt.subplot(1,2,2)*

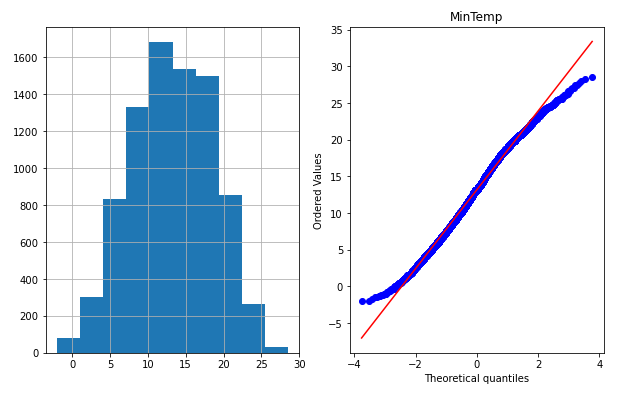
*stats.probplot(df[feature],dist='norm',plot=pylab)*

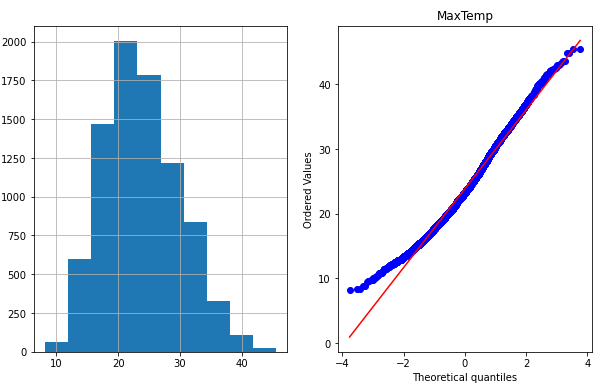
*plt.title(feature)*

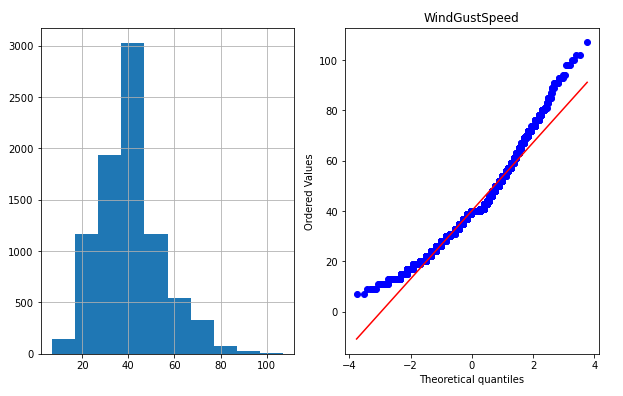
*plt.show()*

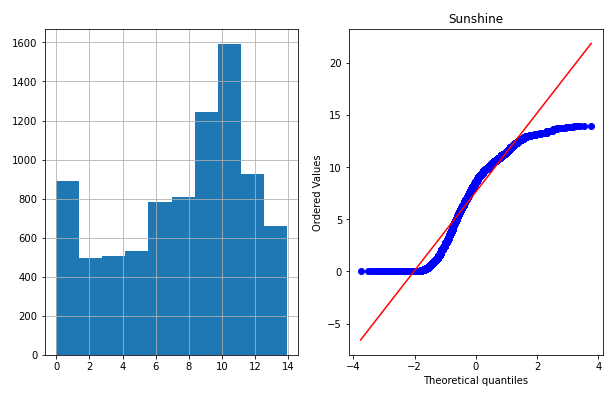
*for i in cont\_var:*

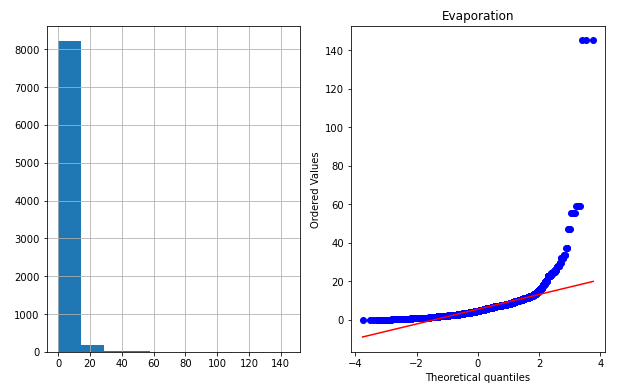
*plot\_curve(rain\_data, i)*

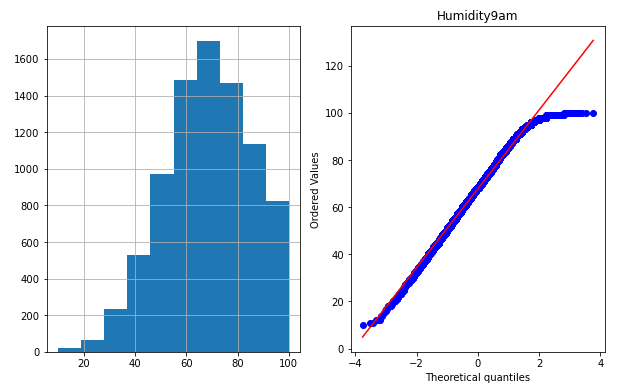
[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

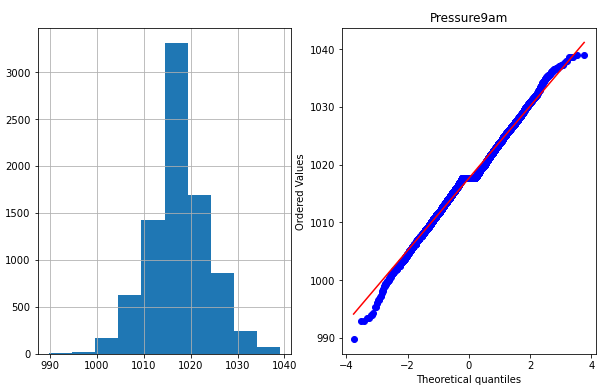
[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

**Step 5: Feature Engineering**

Feature engineering is the process of modifying your data set, including addition, deletion, combination, and mutation, in order to enhance the training of your machine learning model and achieve improved accuracy and performance. Knowledge of the business issue and the data sources available is the foundation for effective feature engineering.

**Creating Average Columns for Wind Speed,Humidity,temperature,humidity,cloud**

*rain\_data['windspeed']=(rain\_data['WindSpeed9am']+rain\_data['WindSpeed3pm'])//2*

*rain\_data.drop(['WindSpeed9am','WindSpeed3pm'],axis=1,inplace=True)*

*rain\_data['Humidity']=(rain\_data['Humidity9am']+rain\_data['Humidity3pm'])//2*

*rain\_data.drop(['Humidity9am','Humidity3pm'],axis=1,inplace=True)*

*rain\_data['Temp']=(rain\_data['Temp9am']+rain\_data['Temp3pm'])//2*

*rain\_data.drop(['Temp9am','Temp3pm'],axis=1,inplace=True)*

*rain\_data['Pressure']=(rain\_data['Pressure9am']+rain\_data['Pressure3pm'])//2*

*rain\_data.drop(['Pressure9am','Pressure3pm'],axis=1,inplace=True)*

*rain\_data['Cloud']=(rain\_data['Cloud9am']+rain\_data['Cloud3pm'])//2*

*rain\_data.drop(['Cloud9am','Cloud3pm'],axis=1,inplace=True)*

*rain\_data["Date"] = pd.to\_datetime(rain\_data["Date"], format = "%Y-%m-%dT", errors = "coerce")*

Creating column for Day, Month ,Year

*rain\_data["Date\_month"] = rain\_data["Date"].dt.month*

*rain\_data["Date\_Year"] = rain\_data["Date"].dt.year*

*rain\_data["Date\_day"] = rain\_data["Date"].dt.day*

*rain\_data.drop(["Date"],axis=1,inplace=True)*

**Step 6: Data PreProcessing for Model**

Checking for Z- Scores

*from scipy.stats import zscore*

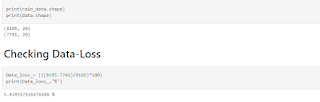
*z\_score=np.abs(zscore(rain\_data))*

Removing data with z score more then 3

*data=rain\_data[(z\_score<3).all(axis=1)]*

*print(rain\_data.shape)*

*print(data.shape)*

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

**Splitting Data**Performing random sampling (SMOTE) to remove skewness in 'RainToday' column as predicting rainfall is pointless if there is no rain

*features= data.drop(["RainToday"], axis=1)*

*target= data["RainToday"]*

*from sklearn.preprocessing import StandardScaler*

*for col in features.columns:*

*SC = StandardScaler()*

*features[col] = SC.fit\_transform(features[col].values.reshape(-1, 1))*

**SMOTE SAMPLING**SMOTE is a method of oversampling that creates artificial samples from the minority class. In order to train the classifier, it is utilised to create a training set that is artificially class-balanced or nearly class-balanced.

*from imblearn import under\_sampling, over\_sampling*

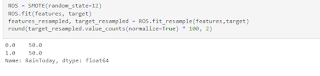
*from imblearn.over\_sampling import SMOTE*

*ROS = SMOTE(random\_state=12)*

*ROS.fit(features, target)*

*features\_resampled, target\_resampled = ROS.fit\_resample(features,target)*

*round(target\_resampled.value\_counts(normalize=True) \* 100, 2)*

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

**Merging Data after removing skewness in the Rain Today Column**

*features\_resampled['RainToday']=0*

*features\_resampled['RainToday']=target\_resampled*

Splitting the Data again as target and features for Model Building

*features= features\_resampled.drop(["Rainfall"], axis=1)*

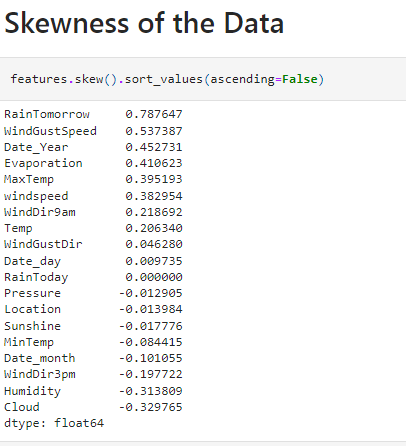
*target= features\_resampled["Rainfall"]*

**Skewness**

While kurtosis assesses whether data have heavy or light tails in a normal distribution, skewness assesses the symmetry or asymmetry of the data distribution. Data may be negatively skewed (pushed to the left) or positively skewed (data-pushed towards the left side)

**Checking for Skewness in the Data**

*features.skew().sort\_values(ascending=False)*

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

**Skew-ness Removal**

Why Skew-ness is a problem?  
The statistical model in skewed data may act as an outlier in the tail region, and we know that outliers have a negative impact on a model's performance, especially in regression-based models.

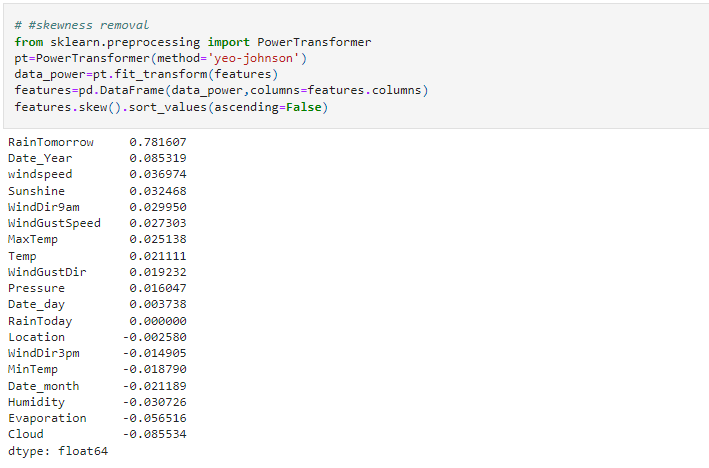
*from sklearn.preprocessing import PowerTransformer*

*pt=PowerTransformer(method='yeo-johnson')*

*data\_power=pt.fit\_transform(features)*

*features=pd.DataFrame(data\_power,columns=features.columns)*

*features.skew().sort\_values(ascending=False)*

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

**Multi-Collinearity**

Anytime an independent variable in a multivariate regression equation has a high correlation with one or more additional independent variables, multi-collinearity exists.

**Why Multi-Collinearity is a Problem?**

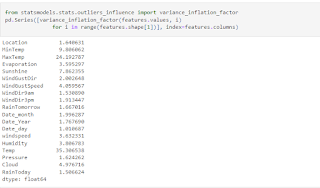
Multi-Collinearity is a concern since it reduces the independent variable's statistical significance. The likelihood that a regression coefficient would be statistically significant decreases, other things being equal, with increasing standard error.

**Checking for Multi-Collinearity**

*from statsmodels.stats.outliers\_influence import variance\_inflation\_factor*

*pd.Series([variance\_inflation\_factor(features.values, i)*

*for i in range(features.shape[1])], index=features.columns)*

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

**Removing Multi-Collinearity**

Dropping Column Temp as the vif score the column is too high.

*features.drop(['Temp'],axis=1,inplace=True)*

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

**Step 7: Model Building**

Here we will simply deploy the various Regression models with default parameters and see which one yields the best result. The models can further be tuned for better performance but are not in the scope of this one article.I'm mention from which I got the best result:

Run this to find out the default score of any Regression algorithm

*from sklearn import datasets, linear\_model, metrics*

*from sklearn import preprocessing*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.linear\_model import LinearRegression*

*#Importing necessary libraries*

*from sklearn.ensemble import RandomForestRegressor*

*from sklearn.linear\_model import LinearRegression*

*from sklearn.metrics import classification\_report*

*from sklearn.model\_selection import cross\_val\_score*

*from sklearn import metrics*

*from sklearn.model\_selection import GridSearchCV*

*from sklearn.linear\_model import Lasso,Ridge*

*from xgboost.sklearn import XGBRegressor*

*from sklearn.model\_selection import train\_test\_split*

*import warnings*

*warnings.filterwarnings("ignore")*

*from sklearn.metrics import r2\_score,mean\_squared\_error,mean\_absolute\_error*

*def build\_model(Features,target,model):*

*Maximum\_Accuracy=0*

*test\_size=[0.2,0.21,0.22,0.23,0.24,0.25,0.26,0.27,0.28,0.29,0.3]*

*over\_fitting=1*

*for j in test\_size:*

*for i in range(100):*

*x\_train, x\_test, y\_train, y\_test = train\_test\_split(Features, target, test\_size=j,*

*random\_state=i)*

*model.fit(x\_train,y\_train)*

*pred= model.predict(x\_test)*

*acc=r2\_score(y\_test,pred)*

*check=model.predict(x\_train)*

*check\_acc=r2\_score(y\_train,check)*

*if acc>Maximum\_Accuracy:*

**Random Forest**

Supervised machine learning algorithms like random forest are frequently employed in classification and regression issues. It creates decision trees from various samples, using their average in the case of regression and majority vote for classification.It works better when the relation of the indepent variables is not linear.

*build\_model(features,target,RandomForestRegressor())*

**Step 8 : Hyper-Parametric Tunning**  
Hyperparameter tuning is an essential part of controlling the behavior of a machine learning model. If we don't correctly tune our hyperparameters, our estimated model parameters produce sub-optimal results, as they don't minimize the loss function. This means our model makes more errors. The more Errors it makes the less useful the model becomes

*x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.2,random\_state=13)*

*estimator = RandomForestRegressor(random\_state=42)*

*param\_grid = {*

*'max\_depth': [10,50,100, None],*

*'min\_samples\_leaf': [1, 2, 4],*

*'n\_estimators': [100,500,1000]*

*}*

*RF= GridSearchCV(estimator, param\_grid,scoring='neg\_mean\_squared\_error')*

*grid\_fit = RF.fit(x\_train, y\_train)*

*rf\_opt = grid\_fit.best\_estimator\_*

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

We are R- squared is 88% approx

**Step 9: Creating Pickle file of the model**

**What is pickling a File?**When serialising and deserializing a Python object structure, pickle is primarily employed. It involves transforming a Python object into a byte stream in order to store it in a file or database, keep programme state consistent across sessions, or send data over the network.

*import pickle*

Saved the trained model in file with given name

*pickle\_out = open("Rain.pkl","wb") # name of my pickle file , wb -write*

*pickle.dump(rf\_opt, pickle\_out)*

*pickle\_out.close()*

**Step 10: Prediction**

Loading the pickeled model for prediction on Y\_test

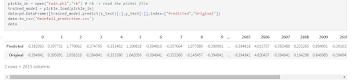
*pickle\_in = open("rain.pkl","rb") # rb -> read the pickel file*

*trained\_model = pickle.load(pickle\_in)*

*data=pd.DataFrame([trained\_model.predict(x\_test)[:],y\_test[:]],index=["Predicted","Original"])*

*data.to\_csv('Rainfall\_prediction.csv')*

*data*

[](https://www.blogger.com/blog/post/edit/3672064354014906488/5978137808132675717)

**Concluding Remarks:**

Random forest Regressor and XGboost Regressor works better for regression problem when the relationship Between them is not Linear and as the loss function reduces make huge difference in the accuracy of the model.  
Linear, Lasso and Ridge Regression work better when the relation of independent variable to target is linear

**Git-Hub Link to Ipynb File:**

https://github.com/Ravinder-Singh-1993/Final-Projects\_Capstone/blob/main/Rain\_prediction\_project/Rain\_Prediction\_project.ipynb