

Predictive Modelling – Rain

Prediction in Australia with Python

Objective:

Have you ever got irritated when your meeting gets cancelled due to rain??? Have you got upset when your vacation plan spoiled by rain??? Heavy rainfall can lead to numerous hazards, for example: flooding, including risk to human life, damage to buildings and infrastructure, and loss of crops and livestock. landslides, which can threaten human life, disrupt transport and communications, and cause damage to buildings and infrastructure. To solve these kinds of problems we have demonstrated a solution for this using Logistic Regression. We used Logistic Regression to check the data and proposed a solution to predict rain prediction.

Abstract:

In this project, I will be implementing a predictive model on Rain Dataset to predict whether or not it will rain tomorrow in Australia. The Dataset contains about 10 years of daily weather observations of different locations in Australia. By the end of this article, you will be able to build a predictive model.

Methodology:

➤ Data Preprocessing:

Real-world data is often messy, incomplete, unstructured, inconsistent, redundant, sprinkled with wacky values. So, without deploying any Data Preprocessing techniques, it is almost impossible to gain insights from raw data. So, here we processing a data to cleanup the dataset .

➤ Finding Categorical and Numerical Features in a Data set:

```
categorical_features = [column_name for column_name in rain.columns if rain[column_name].dtype == object]
print("Number of Categorical Features: {}".format(len(categorical_features)))
print("Categorical Features: ", categorical_features)
```

```
Number of Categorical Features: 4
Categorical Features: ['Date', 'Location', 'RainToday', 'RainTomorrow']
```

In [83]:

```
numerical_features = [column_name for column_name in rain.columns if rain[column_name].dtype in [float, int]]
print("Number of Numerical Features: {}".format(len(numerical_features)))
print("Numerical Features: ", numerical_features)
```

```
Number of Numerical Features: 7
Numerical Features: ['MinTemp', 'MaxTemp', 'Rainfall', 'Windspeed', 'Humidity', 'Pressure', 'RISK_MM']
```

➤ Cardinality check for Categorical features:

- Cardinality: The number of unique values in each categorical feature is known as cardinality.

```
for each_feature in categorical_features:
    unique_values = len(rain[each_feature].unique())
    print("Cardinality(no. of unique values) of {} are: {}".format(each_feature, unique_values))
```

```
Cardinality(no. of unique values) of Date are: 3436
Cardinality(no. of unique values) of Location are: 49
Cardinality(no. of unique values) of RainToday are: 3
Cardinality(no. of unique values) of RainTomorrow are: 3
```

In [85]:

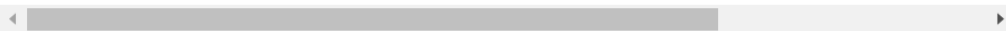
```
rain['Date'] = pd.to_datetime(rain['Date'])
rain['year'] = rain['Date'].dt.year
rain['month'] = rain['Date'].dt.month
rain['day'] = rain['Date'].dt.day
```

In [86]:

```
rain.drop('Date', axis = 1, inplace = True)
rain.head()
```

Out[86]:

	Location	MinTemp	MaxTemp	Rainfall	Windspeed	Humidity	Pressure	RainToday	RISK_M
0	Albury	13.4	22.9	0.6	20.0	71.0	1007.7	No	0
1	Albury	7.4	25.1	0.0	4.0	44.0	1010.6	No	0
2	Albury	12.9	25.7	0.0	19.0	38.0	1007.6	No	0
3	Albury	9.2	28.0	0.0	11.0	45.0	1017.6	No	1
4	Albury	17.5	32.3	1.0	7.0	82.0	1010.8	No	0



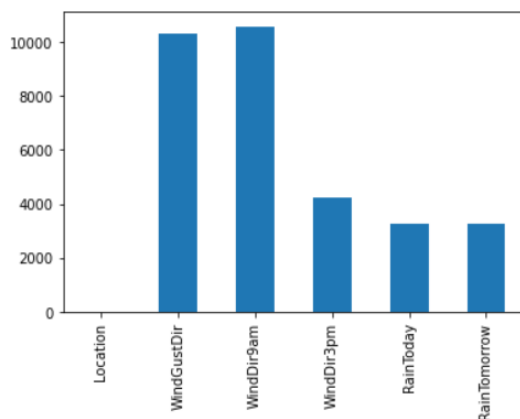
➤ Handling Missing Values:

Machine learning algorithms can't handle missing values and cause problems. If a dataset contains missing values and loaded using pandas, then missing values get replaced with NaN(Not a Number) values. These NaN values can be identified using methods like `isna()` or `isnull()` and they can be imputed using `fillna()`. This process is known as Missing Data Imputation.

```
categorical_features = [column_name for column_name in rain.columns if rain[column_name].dt
rain[categorical_features].isnull().sum()
```

Out[87]:

```
Location      0
RainToday     3261
RainTomorrow  3267
dtype: int64
```



In [88]:

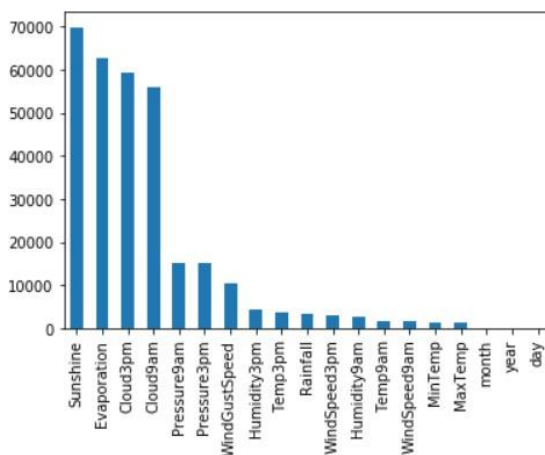
```
categorical_features_with_null = [feature for feature in categorical_features if rain[feature].isnull().sum() > 0]
for each_feature in categorical_features_with_null:
    mode_val = rain[each_feature].mode()[0]
    rain[each_feature].fillna(mode_val, inplace=True)
```

In [89]:

```
numerical_features = [column_name for column_name in rain.columns if rain[column_name].dtype == 'float64' or rain[column_name].dtype == 'int64']
rain[numerical_features].isnull().sum()
```

Out[89]:

```
MinTemp      1485
MaxTemp      1261
Rainfall     3261
Windspeed    1767
Humidity     2654
Pressure     15065
RISK_MM      3267
year          0
month         0
day           0
dtype: int64
```



Missing values in Numerical Features can be imputed using Mean and Median. Mean is sensitive to outliers and median is immune to outliers. If you want to impute the missing values with mean values, then outliers in numerical features need to be addressed properly.

➤ Outliers detection and treatment:

Outlier: An Outlier is an observation that lies an abnormal distance from other values in a given sample. They can be detected using visualization(like boxplots, scatter plots), Z-score, statistical and probabilistic algorithms, etc.

In [92]:

```
features_with_outliers = ['MinTemp', 'MaxTemp', 'Rainfall', 'Windspeed', 'Humidity', 'Pressu']
for feature in features_with_outliers:
    q1 = rain[feature].quantile(0.25)
    q3 = rain[feature].quantile(0.75)
    IQR = q3-q1
    lower_limit = q1 - (IQR*1.5)
    upper_limit = q3 + (IQR*1.5)
    rain.loc[rain[feature]<lower_limit,feature] = lower_limit
    rain.loc[rain[feature]>upper_limit,feature] = upper_limit
```

In [93]:

```
numerical_features_with_null = [feature for feature in numerical_features if rain[feature].isnull().sum() > 0]
for feature in numerical_features_with_null:
    mean_value = rain[feature].mean()
    rain[feature].fillna(mean_value,inplace=True)
```

➤ Exploratory Data Analysis:

Exploratory Data Analysis(EDA) is a technique used to analyze, visualize, investigate, interpret, discover and summarize data

1. Univariate Analysis:

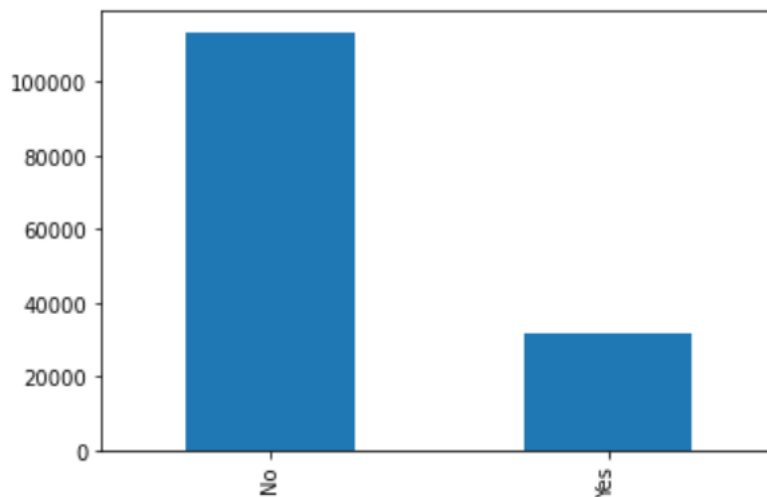
a) Exploring target variable:

In [94]:

```
rain['RainTomorrow'].value_counts().plot(kind='bar')
```

Out[94]:

<AxesSubplot:>



Looks like the Target variable is imbalanced. It has more 'No' values. If data is imbalanced, then it might decrease the performance of the model. As this data is released by the meteorological department of Australia, it doesn't make any sense when we try to balance the target variable, because the truthfulness of data might decrease. So, let me keep it as it is.

2. Bi-variate Analysis:

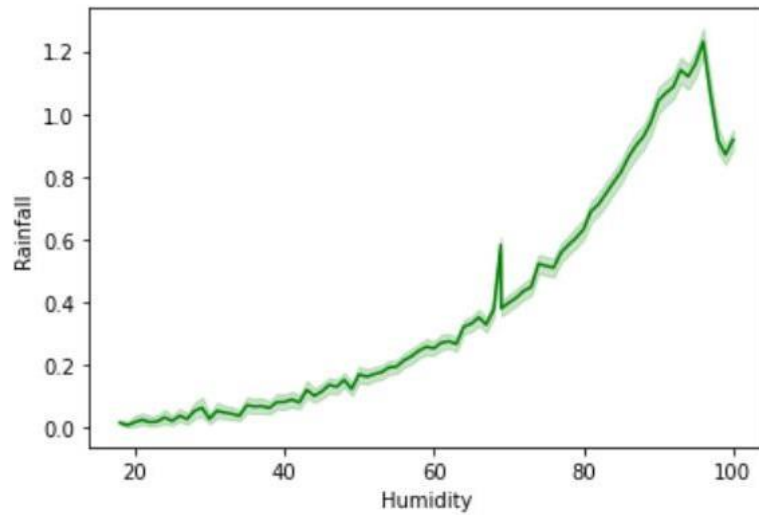
a) Sunshine vs Rainfall:

In [96]:

```
sns.lineplot(data=rain,x='Humidity',y='Rainfall',color='green')
```

Out[96]:

<AxesSubplot:xlabel='Humidity', ylabel='Rainfall'>



In the above line plot, the Sunshine feature is inversely proportional to the Rainfall feature.

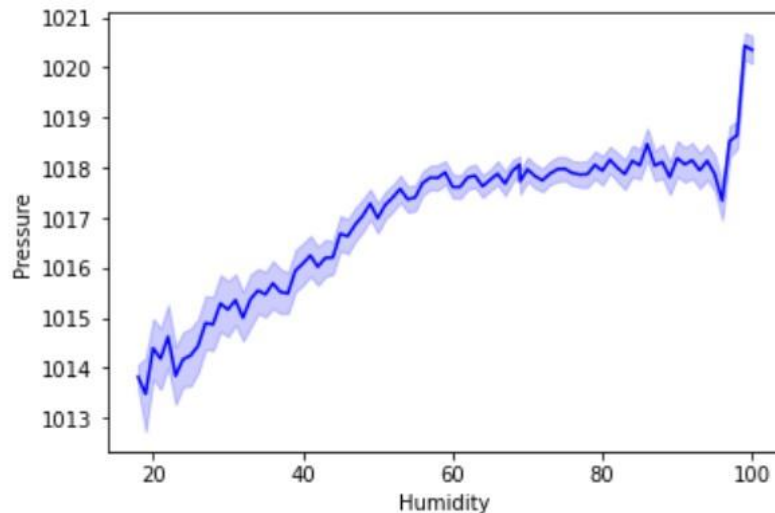
b) Sunshine vs Evaporation:

In [97]:

```
sns.lineplot(data=rain,x='Humidity',y='Pressure',color='blue')
```

Out[97]:

<AxesSubplot:xlabel='Humidity', ylabel='Pressure'>



➤ Encoding of Categorical Features:

Most Machine Learning Algorithms like Logistic Regression, Support Vector Machines, K Nearest Neighbours, etc. can't handle categorical data. Hence, these categorical data need to be converted to numerical data for modeling, which is called Feature Encoding.

There are many feature encoding techniques like One code encoding, label encoding. But in this particular blog, I will be using `replace()` function to encode categorical data to numerical data.

In [98]:

```
def encode_data(feature_name):  
    ...  
    This function takes feature name as a parameter and returns mapping dictionary to repla  
    ...  
    mapping_dict = {}  
    unique_values = list(rain[feature_name].unique())  
    for idx in range(len(unique_values)):  
        mapping_dict[unique_values[idx]] = idx  
    return mapping_dict  
  
rain['RainToday'].replace({'No':0, 'Yes': 1}, inplace = True)  
rain['RainTomorrow'].replace({'No':0, 'Yes': 1}, inplace = True)  
rain['Location'].replace(encode_data('Location'), inplace = True)
```

In [99]:

```
rain.head()
```

Out[99]:

	Location	MinTemp	MaxTemp	Rainfall	Windspeed	Humidity	Pressure	RainToday	RISK_M
0	0	13.4	22.9	0.6	20.0	71.0	1007.7	0	0
1	0	7.4	25.1	0.0	4.0	44.0	1010.6	0	0
2	0	12.9	25.7	0.0	19.0	38.0	1007.6	0	0
3	0	9.2	28.0	0.0	11.0	45.0	1017.6	0	1
4	0	17.5	32.3	1.0	7.0	82.0	1010.8	0	0

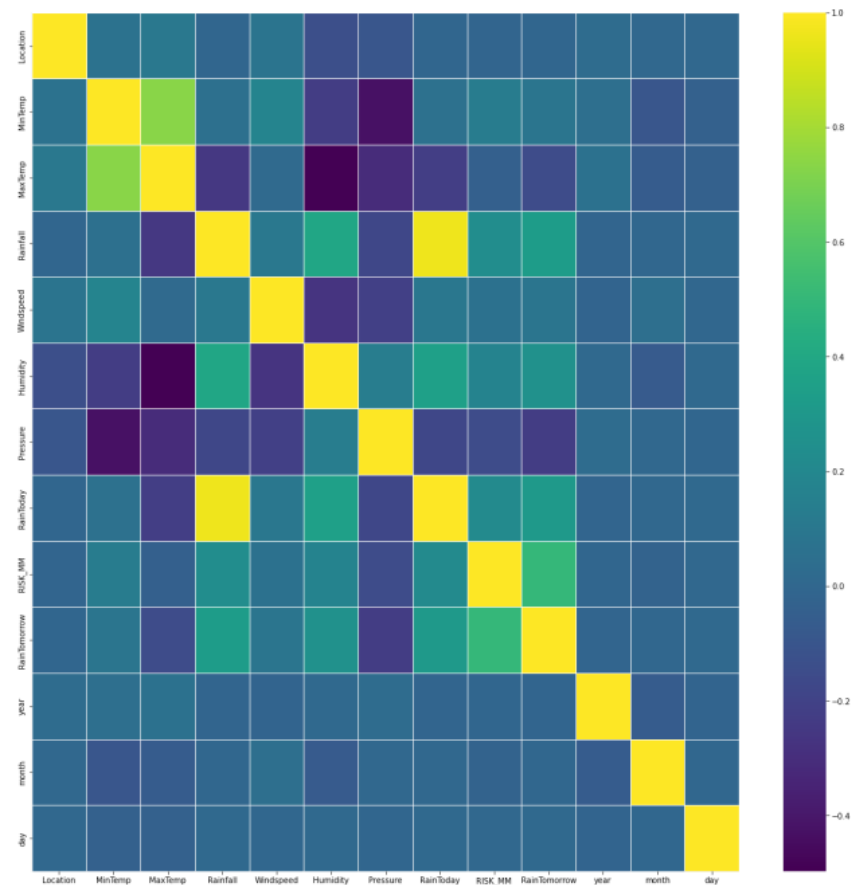
➤ 11. Correlation:

Correlation is a statistic that helps to measure the strength of the relationship between two features. It is used in bivariate analysis. Correlation can be calculated with method `corr()` in pandas.

```
plt.figure(figsize=(20,20))
sns.heatmap(rain.corr(), linewidths=0.5, annot=False, fmt=".2f", cmap = 'viridis')
```

Out[100]:

<AxesSubplot:>



Splitting data into Independent Features and Dependent Features:

For feature importance and feature scaling, we need to split data into independent and dependent features.

In [101]:

```
X = rain.drop(['RainTomorrow'],axis=1)
y = rain['RainTomorrow']
```

In the above code,

- X – Independent Features or Input features
- y – Dependent Features or target label

➤ Feature Importance:

- Machine Learning Model performance depends on features that are used to train a model. Feature importance describes which features are relevant to build a model.
- Feature Importance refers to the techniques that assign a score to input/label features based on how useful they are at predicting a target variable. Feature importance helps in Feature Selection.

We'll be using ExtraTreesRegressor class for Feature Importance. This class implements a meta estimator that fits a number of randomized decision trees on various samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

In [102]:

```
from sklearn.ensemble import ExtraTreesRegressor
etr_model = ExtraTreesRegressor()
etr_model.fit(X,y)
etr_model.feature_importances_
```

Out[102]:

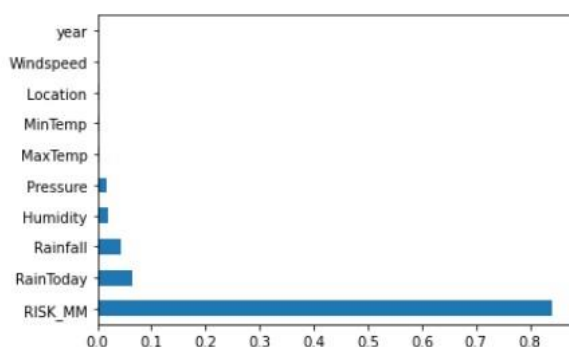
```
array([2.44557085e-03, 2.65351120e-03, 4.74672431e-03, 4.25057697e-02,
       2.29984269e-03, 2.05608199e-02, 1.69119305e-02, 6.37992528e-02,
       8.41515787e-01, 9.19672731e-04, 8.37046740e-04, 8.04071896e-04])
```

In [103]:

```
feature_imp = pd.Series(etr_model.feature_importances_,index=X.columns)
feature_imp.nlargest(10).plot(kind='barh')
```

Out[103]:

<AxesSubplot:>



➤ Splitting Data into training and testing set:

train_test_split() is a method of *model_selection* class used to split data into training and testing sets.

In [104]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state = 0)
```

In [105]:

```
print("Length of Training Data: {}".format(len(X_train)))
print("Length of Testing Data: {}".format(len(X_test)))
```

Length of Training Data: 116368
Length of Testing Data: 29092

➤ Feature Scaling:

Feature Scaling is a technique used to scale, normalize, standardize data in range(0,1). When each column of a dataset has distinct values, then it helps to scale data of each column to a common level. StandardScaler is a class used to implement feature scaling.

In [106]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

In [107]:

```
X_test = scaler.transform(X_test)
```

➤ Model Building:

In this article, I will be using a Logistic Regression algorithm to build a predictive model to predict whether or not it will rain tomorrow in Australia.

- Logistic Regression: It is a statistic-based algorithm used in classification problems. It allows us to predict the probability of an input belongs to a certain category.
- It uses the logit function or sigmoid function as a core.
- According to the Data science community, logistic regression can solve 60% of existing classification problems.

$$h\theta(X) = \frac{1}{1 + e^{- (\beta_0 + \beta_1 X)}}$$

Model Training:

Sklearn library has a module called `linear_model`, which provides `LogisticRegression` class to train a model or a classifier and test it.

```
In [108]:  
from sklearn.linear_model import LogisticRegression  
classifier_logreg = LogisticRegression(solver='liblinear', random_state=0)  
classifier_logreg.fit(X_train, y_train)
```

```
Out[108]:  
LogisticRegression(random_state=0, solver='liblinear')
```

```
In [109]:  
y_pred = classifier_logreg.predict(X_test)  
y_pred
```

```
Out[109]:  
array([0, 1, 0, ..., 1, 0, 0], dtype=int64)
```

`accuracy_score()` is a method used to calculate the accuracy of a model prediction on unseen data.

```
In [110]:  
from sklearn.metrics import accuracy_score  
print("Accuracy Score: {}".format(accuracy_score(y_test, y_pred)))
```

```
Accuracy Score: 0.9527705211054586
```

```
In [111]:  
print("Train Data Score: {}".format(classifier_logreg.score(X_train, y_train)))  
print("Test Data Score: {}".format(classifier_logreg.score(X_test, y_test)))
```

```
Train Data Score: 0.9529423896603877  
Test Data Score: 0.9527705211054586
```

The accuracy Score of training and testing data is comparable and almost equal. So, there is no question of underfitting and overfitting. And the model is generalizing well for new unseen data.

Confusion Matrix:

A Confusion Matrix is used to summarize the performance of the classification problem. It gives a holistic view of how well the model is performing.

```
In [112]:  
from sklearn.metrics import confusion_matrix
```

```
In [113]:  
print(confusion_matrix(y_test,y_pred))
```

```
[[22119  607]  
 [ 767 5599]]
```

Classification-report:

The classification report displays the values of precision, recall, F1 for the model.

```
classification_report:
```

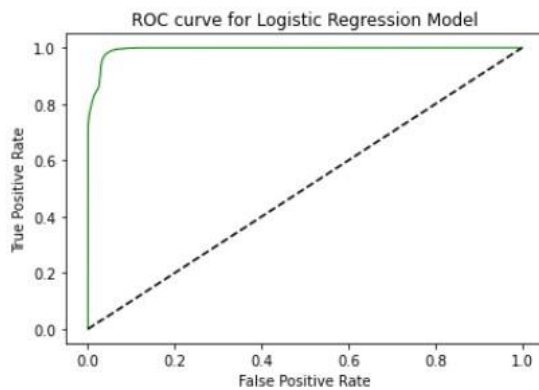
	precision	recall	f1-score	support
0	0.87	0.95	0.90	22726
1	0.72	0.48	0.57	6366
accuracy			0.84	29092
macro avg	0.79	0.71	0.74	29092
weighted avg	0.83	0.84	0.83	29092

Receiver operating characteristic(ROC) curve:

- The ROC curve is an evaluation metric used in binary classification problems to know the performance of the classifier.
- It is a curve plotted between True Positive Rate(TPR) and False Positive Rate(FPR) at various thresholds.
- ROC graph summarizes all the confusion matrices produced at different threshold values.
- ROC curve is used to determine which threshold value is best for Logistic Regression in order to classify classes.

In [114]:

```
y_pred_logreg_proba = classifier_logreg.predict_proba(X_test)
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_logreg_proba[:,1])
plt.figure(figsize=(6,4))
plt.plot(fpr,tpr,'-g',linewidth=1)
plt.plot([0,1], [0,1], 'k--' )
plt.title('ROC curve for Logistic Regression Model')
plt.xlabel("False Positive Rate")
plt.ylabel('True Positive Rate')
plt.show()
```



Cross-Validation:

Let's find out whether model performance can be improved using Cross-Validation Score.

In [115]:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(classifier_logreg, X_train, y_train, cv = 5, scoring='accuracy')
print('Cross-validation scores:{}'.format(scores))
print('Average cross-validation score: {}'.format(scores.mean()))
```

```
Cross-validation scores:[0.95054567 0.95286586 0.95475638 0.95252009 0.95329
352]
Average cross-validation score: 0.9527963032878212
```

The mean accuracy score of cross-validation is almost the same as the original model accuracy score which is 0.95279. So, the accuracy of the model may not be improved using Cross-validation.

Save Model and Scaling object with Pickle:

Pickle is a python module used to serialize and deserialize objects. It is a standard way to store models in machine learning so that they can be used anytime for prediction by unpickling.

Here scaling object is stored in a pickle file, which can be used to standardize real-time and unseen data fed by users for prediction.

In [39]:

```
import pickle

with open('scaler.pkl', 'wb') as file:
    pickle.dump(scaler, file)
```

In [40]:

```
with open('logreg.pkl', 'wb') as file:
    pickle.dump(classifier_logreg, file)
```

➤ Results and Conclusion:

- The logistic Regression model accuracy score is 0.95. The model does a very good job of predicting.
- The model shows no sign of Underfitting or Overfitting. This means the model generalizing well for unseen data.
- The mean accuracy score of cross-validation is almost the same as the original model accuracy score. So, the accuracy of the model may not be improved using Cross-validation.

FOR WEBSITE DEVELOPMENT:

For this we have used streamlit library, and the code is below:

```
import streamlit as st
import joblib
import pandas as pd

import pickle

st.title("Rainfall Prediction")

col1, col2, col3 = st.columns(3)

# getting user input
Location = col1.selectbox("Enter location",["Albury", "BadgerysCreek","Cobar",
"CoffsHarbour","Moree", "Newcastle","NorahHead", "NorfolkIsland","Penrith",
"Richmond","Sydney", "SydneyAirport","WaggaWagga", "Williamtown","Wollongong",
"Canberra","Tuggeranong", "MountGinini","Ballarat", "Bendigo","Sale",
"MelbourneAirport","Melbourne", "Mildura","Nhil", "Portland","Watsonia",
"Dartmoor","Brisbane", "Cairns","GoldCoast", "Townsville","Adelaide",
"MountGambier","Nuriootpa", "Woomera","Albany", "Witchcliffe","PearceRAAF",
```



```

"PerthAirport","Perth", "SalmonGums","Walpole", "Hobart","Launceston",
"AliceSprings","Darwin", "Katherine", "Uluru"])
MinTemp = col2.number_input("what is the minimum temperature ?")
MaxTemp = col3.number_input("what is the maximum temperature ?")
Rainfall = col1.number_input("enter rainfall")
Windspeed = col2.number_input("enter wind_speed")
Humidity = col3.number_input("enter humidity")
Pressure = col1.number_input("enter pressure")
RainToday = col2.selectbox("was there rain today ?",["Yes","No"])

st.button('Predict')

#df_pred =
pd.DataFrame([[Location,MinTemp,MaxTemp,Rainfall,Windspeed,Humidity,Pressure
,RainToday]]),

#columns=
['Location','MinTemp','MaxTemp','Rainfall','Windspeed','Humidity','Pressure'
,'RainToday'])
model = pickle.load(open('scaler.pkl', 'rb'))

df_scaled = model.fit_transform(X)

clf = svm.SVR(kernel='rbf', C=10000.0, gamma=0.01, max_iter=-1).fit(df_scaled,
y)

def predict_HMF(dataframe):
    query_df=pd.DataFrame(dataframe)
    query_df=query_df[['f1','f2','f3']]
    query_df['target'] = [float(x) for x in
clf.predict((scaler.transform(query_df))) ]
    return query_df,

print(model)
prediction =
model.predict(Location,MinTemp,MaxTemp,Rainfall,Windspeed,Humidity,Pressure
,RainToday)

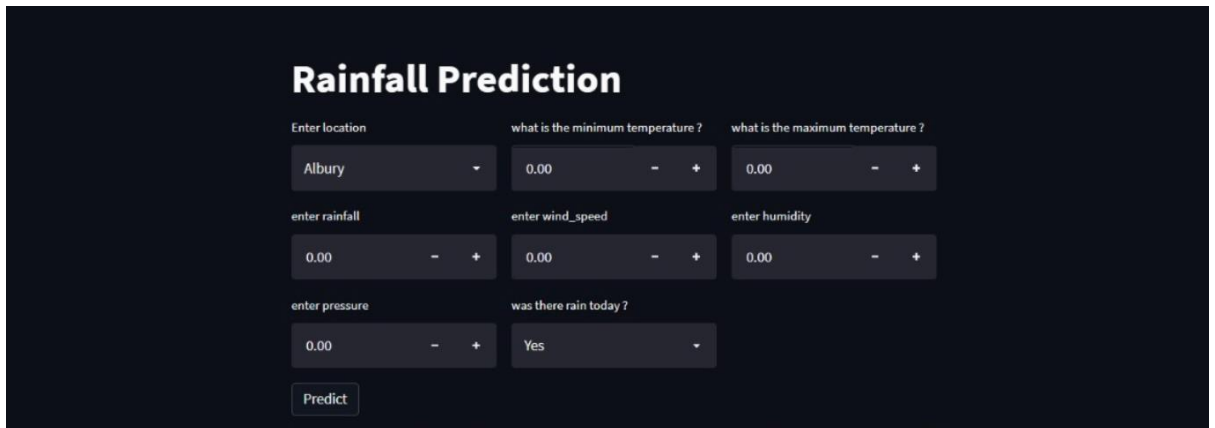
if st.button('Predict'):

    if(prediction[0]>1):
        st.write('<p class="big-font">there will be a rain
tomorrow.</p>',unsafe_allow_html=True)

    else:
        st.write('<p class="big-font">no rain
tomorrow.</p>',unsafe_allow_html=True)

```

➤ OUTPUT:



The image shows a web application titled "Rainfall Prediction" with a dark theme. It features several input fields for weather data: "Enter location" (a dropdown menu showing "Albury"), "what is the minimum temperature ?" (a numeric input with "0.00"), "what is the maximum temperature ?" (a numeric input with "0.00"), "enter rainfall" (a numeric input with "0.00"), "enter wind_speed" (a numeric input with "0.00"), "enter humidity" (a numeric input with "0.00"), "enter pressure" (a numeric input with "0.00"), and "was there rain today ?" (a dropdown menu showing "Yes"). A "Predict" button is located at the bottom left of the form area.

FUTURE ENHANCEMENT:

Need to improve the prediction in steamlit.