USA RAINFALL PREDICTION

```
#importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten,Dropout
from sklearn.preprocessing import StandardScaler
```

Reading data from source

```
df=pd.read_csv("weatherAUS.csv")
```

```
df
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustD
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WN
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WS
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	N
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	
...	...	...	...	...	...	...	...	
145455	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	
145456	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NN
145457	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	
145458	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	S
145459	2017-06-25	Uluru	14.9	NaN	0.0	NaN	NaN	Na

145460 rows × 23 columns

Understanding the Data

Descriptive Statistics

```
df.describe(include="all")
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGu
count	145460	145460	143975.000000	144199.000000	142199.000000	82670.000000	75625.000000	1
unique	3436	49	NaN	NaN	NaN	NaN	NaN	
top	2013-11-12	Canberra	NaN	NaN	NaN	NaN	NaN	
freq	49	3436	NaN	NaN	NaN	NaN	NaN	
mean	NaN	NaN	12.194034	23.221348	2.360918	5.468232	7.611178	
std	NaN	NaN	6.398495	7.119049	8.478060	4.193704	3.785483	

```
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
```

There are 145460 observations with 23 columns where the column named "RainTomorrow" is the target column and other columns are the explanatory/ independent columns.

There are many missing values which needs to be filled.

To fill missing values we will use imputer

```
from sklearn.impute import SimpleImputer
```

```
numsi=SimpleImputer(missing_values=np.nan,strategy="mean") #creating an object of class
catsi=SimpleImputer(missing_values=np.nan,strategy="most_frequent")
```

for numerical data we use mean of the column to impute in place of null values and for categorical data mode is to be imputed.

```
numcol,catcol=df.select_dtypes([float]).columns,df.select_dtypes([object]).columns
df[numcol]=numsi.fit_transform(df[numcol])
df[catcol]=catsi.fit_transform(df[catcol])
```

```
df.isnull().sum()
```

```
Date             0
Location         0
MinTemp          0
MaxTemp          0
Rainfall         0
Evaporation      0
Sunshine         0
WindGustDir      0
WindGustSpeed    0
WindDir9am       0
WindDir3pm       0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
```

```
Pressure9am      0
Pressure3pm      0
Cloud9am         0
Cloud3pm         0
Temp9am          0
Temp3pm          0
RainToday        0
RainTomorrow     0
dtype: int64
```

Encoding Categorical Data

```
from sklearn.preprocessing import OrdinalEncoder
oe=OrdinalEncoder()
df[catcol]=oe.fit_transform(df[catcol])
```

```
df
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpee
0	396.0	2.0	13.4	22.900000	0.6	5.468232	7.611178	13.0	44.0000
1	397.0	2.0	7.4	25.100000	0.0	5.468232	7.611178	14.0	44.0000
2	398.0	2.0	12.9	25.700000	0.0	5.468232	7.611178	15.0	46.0000
3	399.0	2.0	9.2	28.000000	0.0	5.468232	7.611178	4.0	24.0000
4	400.0	2.0	17.5	32.300000	1.0	5.468232	7.611178	13.0	41.0000
...	...	...	...	...	...	...	...	...	.
145455	3431.0	41.0	2.8	23.400000	0.0	5.468232	7.611178	0.0	31.0000
145456	3432.0	41.0	3.6	25.300000	0.0	5.468232	7.611178	6.0	22.0000
145457	3433.0	41.0	5.4	26.900000	0.0	5.468232	7.611178	3.0	37.0000
145458	3434.0	41.0	7.8	27.000000	0.0	5.468232	7.611178	9.0	28.0000
145459	3435.0	41.0	14.9	23.221348	0.0	5.468232	7.611178	13.0	40.0352

145460 rows × 23 columns

```
df["RainTomorrow"]=df["RainTomorrow"].astype(int)
```

```
df["RainTomorrow"].value_counts()
```

```
0    113583
1     31877
Name: RainTomorrow, dtype: int64
```

We can see that it is an IMBALANCED DATA ,so we need to perform SMOTE

SPLITTING DATA INTO TARGET AND FEATURES

```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

Training and testing data

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=3)
```

```
from imblearn.combine import SMOTETomek
```

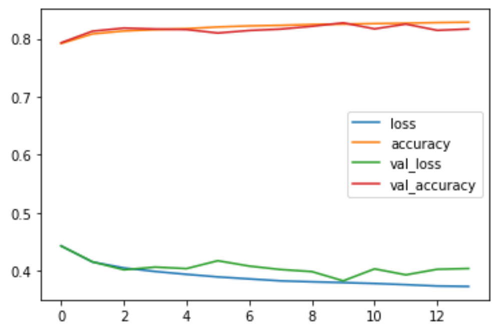
```
st=SMOTETomek(random_state=0)
xtrainn,ytrainn=st.fit_resample(xtrain,ytrain)
```

```
sc=StandardScaler()
xtrainn=sc.fit_transform(xtrainn)
xtest=sc.transform(xtest)
```

```
from tensorflow.keras.callbacks import EarlyStopping
early_stop=EarlyStopping(monitor="val_loss",mode="min",verbose=1,min_delta=1,patience=13)


#Step 1: init the model
ann=Sequential()
#Step 2: add layers into the model
# Hidden Layer 1
ann.add(Dense(units=64,activation="relu"))
#Dropout Layer for Hidden Layer 1
ann.add(Dropout(rate=0.2))

#Hidden Layer 2:
ann.add(Dense(units=64,activation="relu"))
#dropout layer for Hidden Layer 2
ann.add(Dropout(rate=0.1))

#Output Layer
ann.add(Dense(units=1,activation="sigmoid"))     #since its binary classification

#Step 3:Establish connection between layers
ann.compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy"])




#step 4: Train the model
ann.fit(xtrainn,ytrainn,epochs=800,validation_data=(xtest,ytest),verbose=1,callbacks=[early_stop])
```

```
Epoch 1/800
5655/5655 [==============================] - 19s 3ms/step - loss: 0.4425 - accuracy: 0.7917 - val_loss: 0.4424 - val_accuracy: 0.79
Epoch 2/800
5655/5655 [==============================] - 16s 3ms/step - loss: 0.4148 - accuracy: 0.8083 - val_loss: 0.4150 - val_accuracy: 0.81
Epoch 3/800
5655/5655 [==============================] - 15s 3ms/step - loss: 0.4045 - accuracy: 0.8134 - val_loss: 0.4013 - val_accuracy: 0.81
Epoch 4/800
5655/5655 [==============================] - 15s 3ms/step - loss: 0.3984 - accuracy: 0.8157 - val_loss: 0.4059 - val_accuracy: 0.81
Epoch 5/800
5655/5655 [==============================] - 15s 3ms/step - loss: 0.3935 - accuracy: 0.8173 - val_loss: 0.4033 - val_accuracy: 0.81
Epoch 6/800
5655/5655 [==============================] - 15s 3ms/step - loss: 0.3890 - accuracy: 0.8203 - val_loss: 0.4169 - val_accuracy: 0.80
Epoch 7/800
5655/5655 [==============================] - 17s 3ms/step - loss: 0.3855 - accuracy: 0.8221 - val_loss: 0.4077 - val_accuracy: 0.81
Epoch 8/800
5655/5655 [==============================] - 16s 3ms/step - loss: 0.3822 - accuracy: 0.8231 - val_loss: 0.4017 - val_accuracy: 0.81
Epoch 9/800
5655/5655 [==============================] - 15s 3ms/step - loss: 0.3805 - accuracy: 0.8246 - val_loss: 0.3982 - val_accuracy: 0.82
Epoch 10/800
5655/5655 [==============================] - 16s 3ms/step - loss: 0.3792 - accuracy: 0.8252 - val_loss: 0.3822 - val_accuracy: 0.82
Epoch 11/800
5655/5655 [==============================] - 16s 3ms/step - loss: 0.3776 - accuracy: 0.8261 - val_loss: 0.4029 - val_accuracy: 0.81
Epoch 12/800
5655/5655 [==============================] - 18s 3ms/step - loss: 0.3756 - accuracy: 0.8268 - val_loss: 0.3926 - val_accuracy: 0.82
Epoch 13/800
5655/5655 [==============================] - 17s 3ms/step - loss: 0.3733 - accuracy: 0.8278 - val_loss: 0.4022 - val_accuracy: 0.81
Epoch 14/800
5655/5655 [==============================] - 16s 3ms/step - loss: 0.3726 - accuracy: 0.8285 - val_loss: 0.4035 - val_accuracy: 0.81
Epoch 14: early stopping
<keras.callbacks.History at 0x7f06b7ad48b0>
```

```
df2=pd.DataFrame(ann.history.history)
df2.plot()
plt.show()
```




```
# Step 5: Predicting using model
ypred=ann.predict(xtest)
```

```
910/910 [==============================] - 1s 1ms/step
```

```
ypred
```

```
array([[0.42791754],
       [0.12508179],
       [0.24476512],
       ...,
       [0.04935293],
       [0.00862174],
       [0.31016973]], dtype=float32)
```

These are nothing but probablistic value that lies within the logit function , so we need to convert them to binary class by specifing the condition(threshold value)

```
ypred=np.where(ypred>0.5,1,0)
ypred
```

```
array([[0],
       [0],
       [0],
       ...,
       [0],
       [0],
       [0]])
```

EVALUATING THE MODEL

```
from sklearn.metrics import accuracy_score,classification_report
```

```
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.84      0.88     22714
           1       0.56      0.73      0.64      6378

    accuracy                           0.82     29092
   macro avg       0.74      0.79      0.76     29092
weighted avg       0.84      0.82      0.82     29092
```