In [1]:
```
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sb
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import RandomOverSampler
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
import numpy as np
import pandas as pd
import os
import missingno as msno
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from scipy import stats
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, f1_score
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
for dirname, _, filenames in os.walk("C:/Users/palva/Desktop/DM project/weathe
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [3]:
```
rainaus= pd.read_csv("C:/Users/palva/Desktop/DM project/weather.csv")
rainaus.head(10)
```

Out[3]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGus
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	
5	2008-12-06	Albury	14.6	29.7	0.2	NaN	NaN	WNW	
6	2008-12-07	Albury	14.3	25.0	0.0	NaN	NaN	W	
7	2008-12-08	Albury	7.7	26.7	0.0	NaN	NaN	W	
8	2008-12-09	Albury	9.7	31.9	0.0	NaN	NaN	NNW	
9	2008-12-10	Albury	13.1	30.1	1.4	NaN	NaN	W	

10 rows × 23 columns

In [4]:
```
print(f'The number of rows are {rainaus.shape[0] } and the number of columns a
```

```
The number of rows are 145460 and the number of columns are 23
```

In [5]:
```
rainaus.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
```

In [6]:
```
categorical_col, contin_val=[],[]

for i in rainaus.columns:

    if rainaus[i].dtype == 'object':
        categorical_col.append(i)
    else:
        contin_val.append(i)

print(categorical_col)
print(contin_val)
```

```
['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday',
'RainTomorrow']
['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpee
d', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9a
m', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm']
```

In [7]:
```
rainaus.nunique()
```

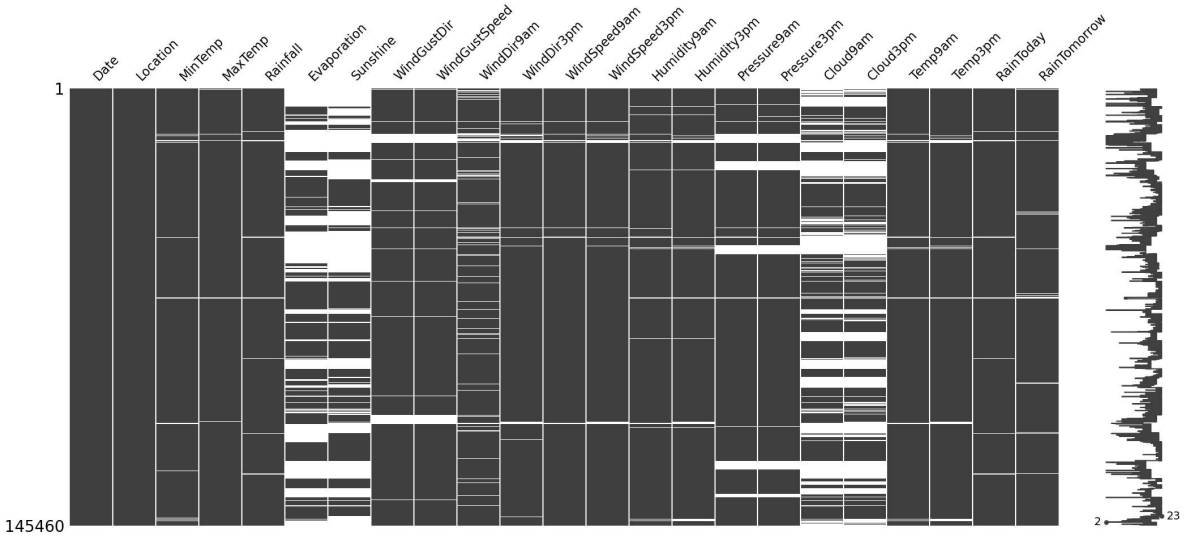
Out[7]:
```
Date             3436
Location           49
MinTemp           389
MaxTemp           505
Rainfall          681
Evaporation       358
Sunshine          145
WindGustDir        16
WindGustSpeed      67
WindDir9am         16
WindDir3pm         16
WindSpeed9am       43
WindSpeed3pm       44
Humidity9am       101
Humidity3pm       101
Pressure9am       546
Pressure3pm       549
Cloud9am           10
Cloud3pm           10
Temp9am           441
Temp3pm           502
RainToday           2
RainTomorrow        2
dtype: int64
```

In [8]:
```
rainaus.isnull().sum()
```

Out[8]:
```
Date                0
Location            0
MinTemp          1485
MaxTemp          1261
Rainfall         3261
Evaporation     62790
Sunshine        69835
WindGustDir     10326
WindGustSpeed   10263
WindDir9am      10566
WindDir3pm       4228
WindSpeed9am     1767
WindSpeed3pm     3062
Humidity9am      2654
Humidity3pm      4507
Pressure9am     15065
Pressure3pm     15028
Cloud9am        55888
Cloud3pm        59358
Temp9am          1767
Temp3pm          3609
RainToday        3261
RainTomorrow     3267
dtype: int64
```

In [9]:
```
msno.matrix(rainaus)
```

Out[9]: <AxesSubplot:>



In [10]:
```
plt.figure(figsize=(17,15))
ax = sns.heatmap(rainaus.corr(), square=True, annot=True, fmt='.2f')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.show()


#MinTemp and maxtemp highly correlated.
#pressure 9am and pressure 3pm
```

In [11]:
```
rainaus['RainTomorrow'] = rainaus['RainTomorrow'].map({'Yes': 1, 'No': 0})
rainaus['RainToday'] = rainaus['RainToday'].map({'Yes': 1, 'No': 0})

print(rainaus.RainToday)
print(rainaus.RainTomorrow)
```

```
0         0.0
1         0.0
2         0.0
3         0.0
4         0.0
          ...
145455    0.0
145456    0.0
145457    0.0
145458    0.0
145459    0.0
Name: RainToday, Length: 145460, dtype: float64
0         0.0
1         0.0
2         0.0
3         0.0
4         0.0
          ...
145455    0.0
145456    0.0
145457    0.0
145458    0.0
145459    NaN
Name: RainTomorrow, Length: 145460, dtype: float64
```

In [12]:
```
#Checking percentage of missing data in every column

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

Out[12]:
```
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
```

In [13]:
```
#Filling the missing values for continuous variables with mean
rainaus['MinTemp']=rainaus['MinTemp'].fillna(rainaus['MinTemp'].mean())
rainaus['MaxTemp']=rainaus['MinTemp'].fillna(rainaus['MaxTemp'].mean())
rainaus['Rainfall']=rainaus['Rainfall'].fillna(rainaus['Rainfall'].mean())
rainaus['Evaporation']=rainaus['Evaporation'].fillna(rainaus['Evaporation'].me
rainaus['Sunshine']=rainaus['Sunshine'].fillna(rainaus['Sunshine'].mean())
rainaus['WindGustSpeed']=rainaus['WindGustSpeed'].fillna(rainaus['WindGustSpee
rainaus['WindSpeed9am']=rainaus['WindSpeed9am'].fillna(rainaus['WindSpeed9am']
rainaus['WindSpeed3pm']=rainaus['WindSpeed3pm'].fillna(rainaus['WindSpeed3pm']
rainaus['Humidity9am']=rainaus['Humidity9am'].fillna(rainaus['Humidity9am'].me
rainaus['Humidity3pm']=rainaus['Humidity3pm'].fillna(rainaus['Humidity3pm'].me
rainaus['Pressure9am']=rainaus['Pressure9am'].fillna(rainaus['Pressure9am'].me
rainaus['Pressure3pm']=rainaus['Pressure3pm'].fillna(rainaus['Pressure3pm'].me
rainaus['Cloud9am']=rainaus['Cloud9am'].fillna(rainaus['Cloud9am'].mean())
rainaus['Cloud3pm']=rainaus['Cloud3pm'].fillna(rainaus['Cloud3pm'].mean())
rainaus['Temp9am']=rainaus['Temp9am'].fillna(rainaus['Temp9am'].mean())
rainaus['Temp3pm']=rainaus['Temp3pm'].fillna(rainaus['Temp3pm'].mean())
```

In [14]:
```
#Filling the missing values for continuous variables with mode

rainaus['RainToday']=rainaus['RainToday'].fillna(rainaus['RainToday'].mode()[0
rainaus['RainTomorrow']=rainaus['RainTomorrow'].fillna(rainaus['RainTomorrow']
```

In [15]:
```
#Filling the missing values for continuous variables with mode
rainaus['WindDir9am'] = rainaus['WindDir9am'].fillna(rainaus['WindDir9am'].moc
rainaus['WindGustDir'] = rainaus['WindGustDir'].fillna(rainaus['WindGustDir'].
rainaus['WindDir3pm'] = rainaus['WindDir3pm'].fillna(rainaus['WindDir3pm'].moc
```

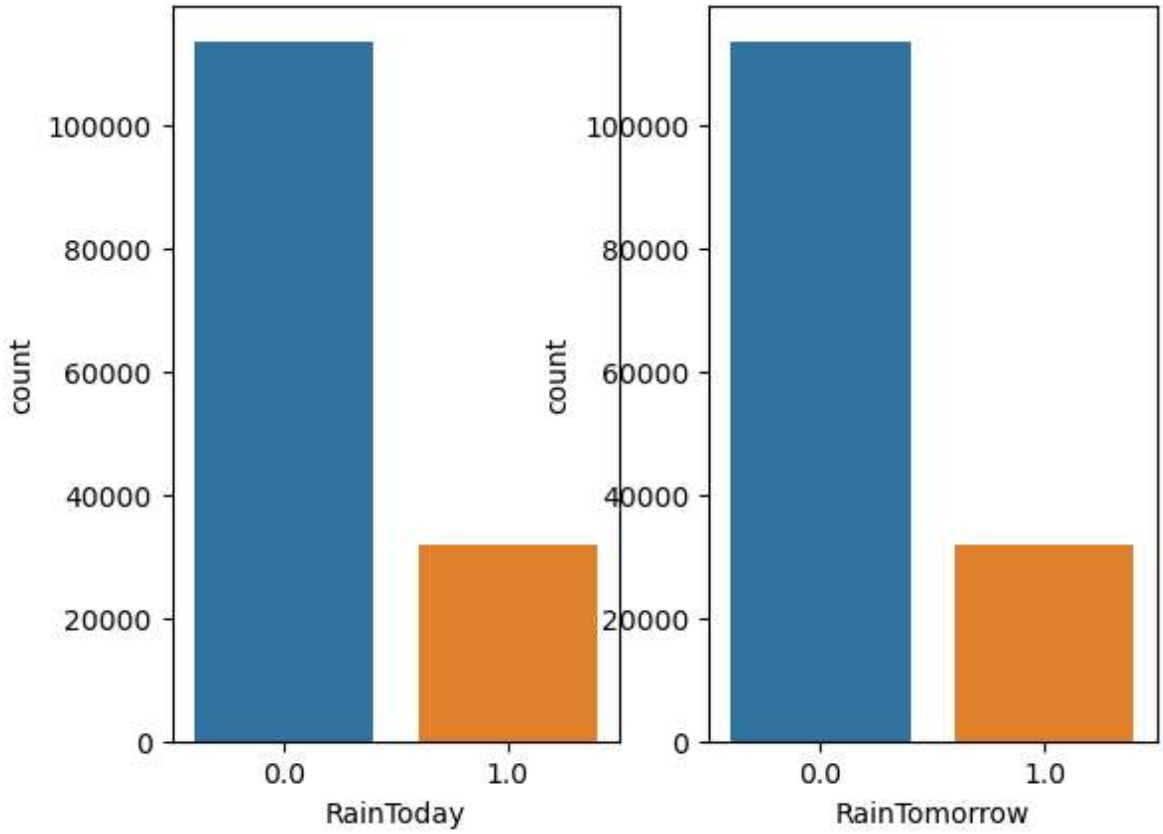
In [16]:
```
#Checking percentage of missing data in every column

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

Out[16]:
```
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
```

In [17]:
```
#count of rain today and rain tomorrow
fig, ax =plt.subplots(1,2)
print(rainaus.RainToday.value_counts())
print(rainaus.RainTomorrow.value_counts())

plt.figure(figsize=(20,20))
sns.countplot(data=rainaus,x='RainToday',ax=ax[0])
sns.countplot(data=rainaus,x='RainTomorrow',ax=ax[1])
```

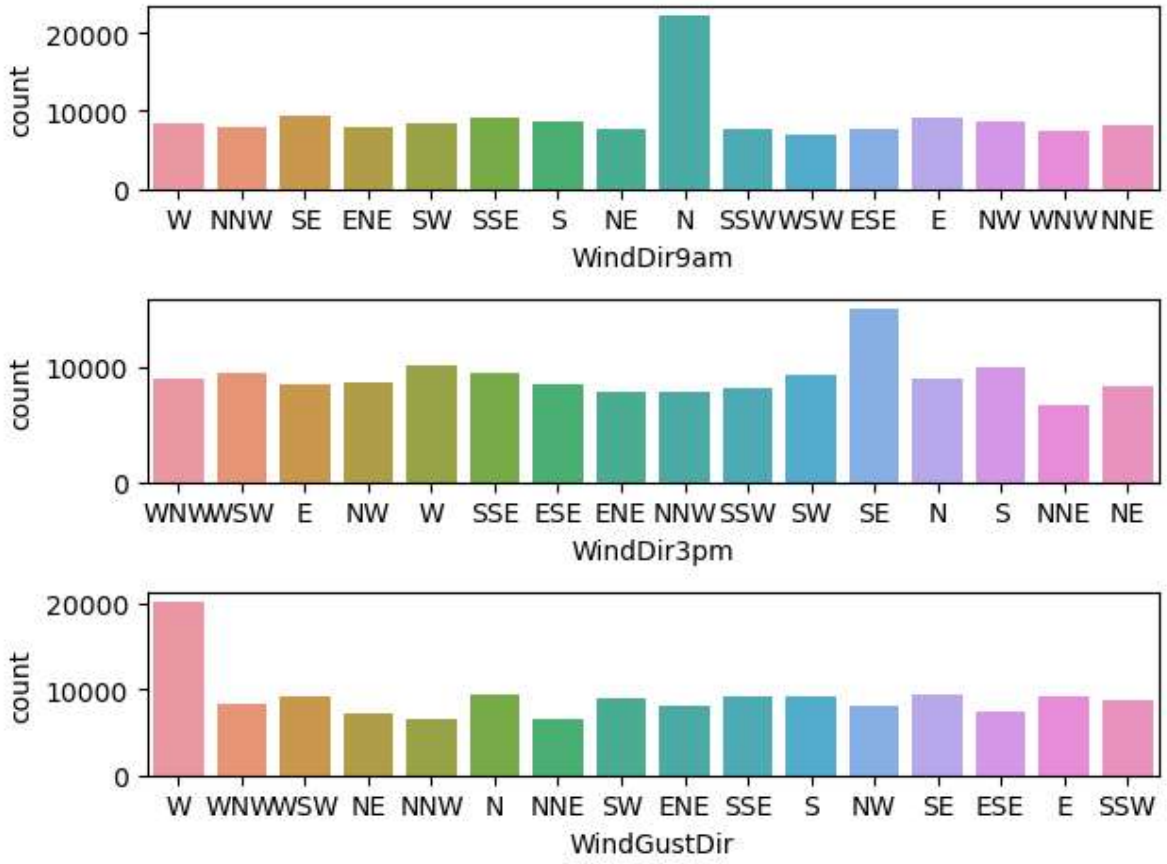
```
0.0    113580
1.0     31880
Name: RainToday, dtype: int64
0.0    113583
1.0     31877
Name: RainTomorrow, dtype: int64
```

Out[17]: <AxesSubplot:xlabel='RainTomorrow', ylabel='count'>



<Figure size 2000x2000 with 0 Axes>

In [18]:
```
# Direction of wind at 9 am, 3 pm.
fig, ax =plt.subplots(3,1)
plt.figure(figsize=(10,10))

sns.countplot(data=rainaus,x='WindDir9am',ax=ax[0])  # north side.
sns.countplot(data=rainaus,x='WindDir3pm',ax=ax[1])  # south east.
sns.countplot(data=rainaus,x='WindGustDir',ax=ax[2])  #west side
fig.tight_layout()
```




```
<Figure size 1000x1000 with 0 Axes>
```

In [19]:
```
#Dropping date column
rainaus=rainaus.iloc[:,1:]
rainaus
```

Out[19]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustS
0	Albury	13.4	13.4	0.6	5.468232	7.611178	W	44.(
1	Albury	7.4	7.4	0.0	5.468232	7.611178	WNW	44.(
2	Albury	12.9	12.9	0.0	5.468232	7.611178	WSW	46.(
3	Albury	9.2	9.2	0.0	5.468232	7.611178	NE	24.(
4	Albury	17.5	17.5	1.0	5.468232	7.611178	W	41.(
...	...	...	...	...	...	...	...	
145455	Uluru	2.8	2.8	0.0	5.468232	7.611178	E	31.(
145456	Uluru	3.6	3.6	0.0	5.468232	7.611178	NNW	22.(
145457	Uluru	5.4	5.4	0.0	5.468232	7.611178	N	37.(
145458	Uluru	7.8	7.8	0.0	5.468232	7.611178	SE	28.(
145459	Uluru	14.9	14.9	0.0	5.468232	7.611178	W	40.(

145460 rows × 22 columns

In [20]:
```
#Encoding the categorical variables
le = preprocessing.LabelEncoder()
rainaus['Location'] = le.fit_transform(rainaus['Location'])
rainaus['WindDir9am'] = le.fit_transform(rainaus['WindDir9am'])
rainaus['WindDir3pm'] = le.fit_transform(rainaus['WindDir3pm'])
rainaus['WindGustDir'] = le.fit_transform(rainaus['WindGustDir'])
```

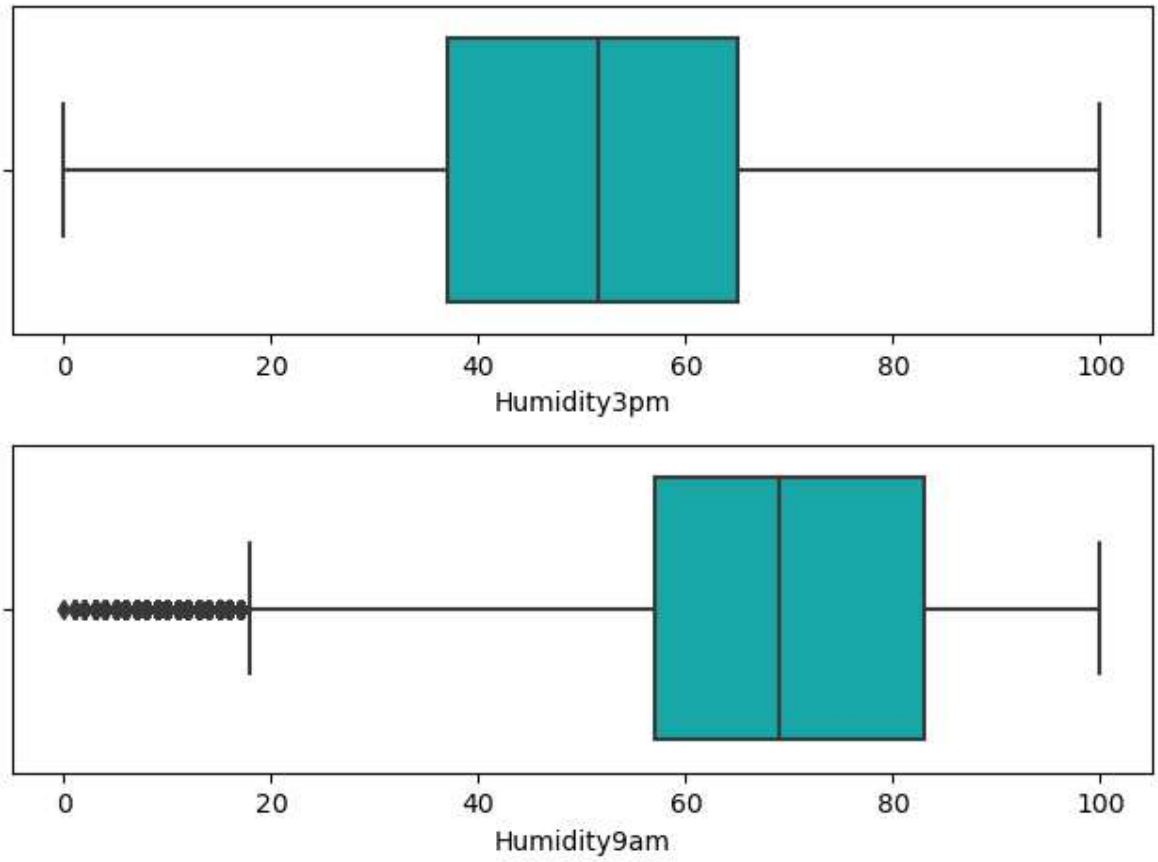
In [21]:
```
rainaus.head()
```

Out[21]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
0	2	13.4	13.4	0.6	5.468232	7.611178	13	44.0
1	2	7.4	7.4	0.0	5.468232	7.611178	14	44.0
2	2	12.9	12.9	0.0	5.468232	7.611178	15	46.0
3	2	9.2	9.2	0.0	5.468232	7.611178	4	24.0
4	2	17.5	17.5	1.0	5.468232	7.611178	13	41.0

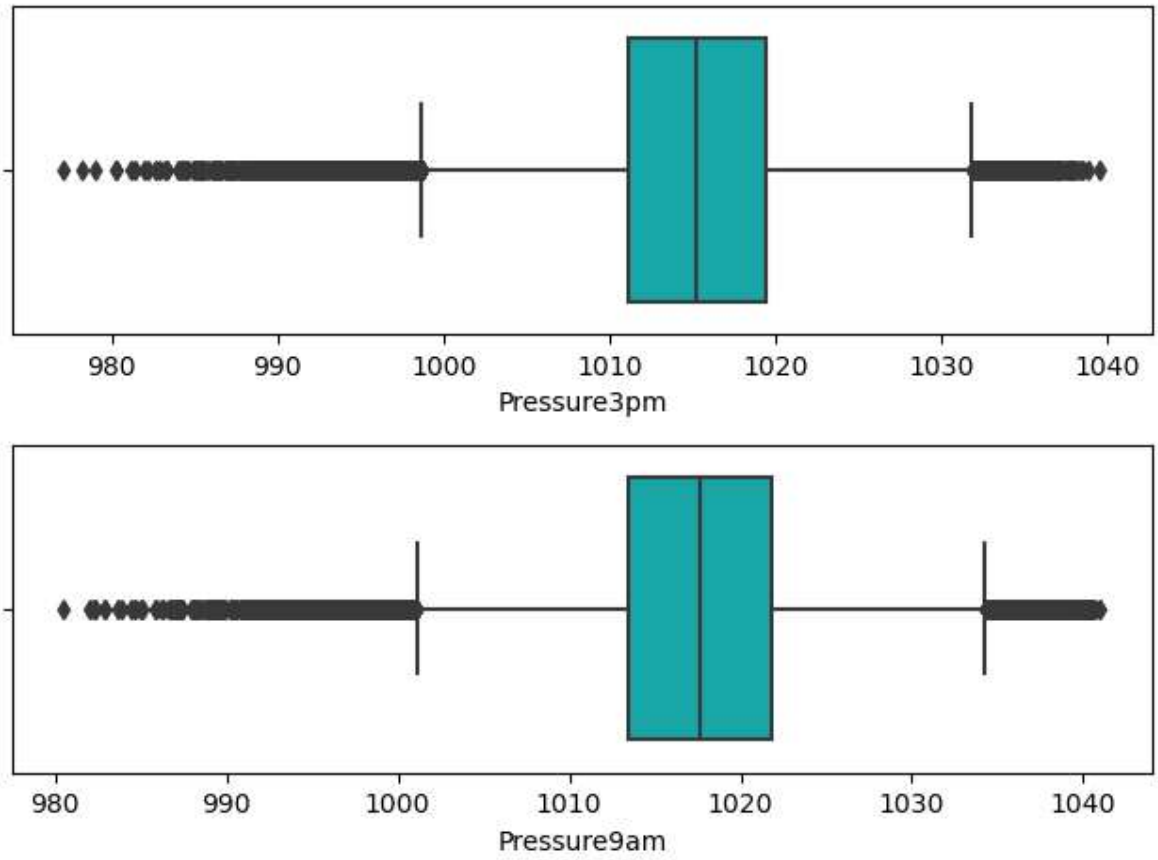
5 rows × 22 columns

In [22]:
```
#using box plots see for outliers
fig, ax =plt.subplots(2,1)
plt.figure(figsize=(10,10))
sns.boxplot(rainaus['Humidity3pm'],orient='v',color='c',ax=ax[0])
sns.boxplot(rainaus['Humidity9am'],orient='v',color='c',ax=ax[1])
fig.tight_layout()
```




```
<Figure size 1000x1000 with 0 Axes>
```

In [23]:
```
fig, ax =plt.subplots(2,1)
plt.figure(figsize=(10,10))
sns.boxplot(rainaus['Pressure3pm'],orient='v',color='c',ax=ax[0])
sns.boxplot(rainaus['Pressure9am'],orient='v',color='c',ax=ax[1])
fig.tight_layout()
```




```
<Figure size 1000x1000 with 0 Axes>
```

In [24]:
```
# removing the outliers from the dataset
print('Shape of DataFrame Before Removing Outliers', rainaus.shape )
rainaus=rainaus[(np.abs(stats.zscore(rainaus)) < 3).all(axis=1)]
print('Shape of DataFrame After Removing Outliers', rainaus.shape )
```

```
Shape of DataFrame Before Removing Outliers (145460, 22)
Shape of DataFrame After Removing Outliers (136653, 22)
```

In [25]:
```
# removing the highly corelated colums
rainaus=rainaus.drop(['Temp3pm','Temp9am','Humidity9am'],axis=1)
rainaus.columns
```

Out[25]:
```
Index(['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshin
e',
       'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity3pm', 'Pressure9am',
       'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'RainToday', 'RainTomorrow'],
      dtype='object')
```

In [26]:
```
#SLICING
x_train, x_test, y_train, y_test = train_test_split(rainaus.iloc[:,:-1], raina
```

In [27]:
```
#Balancing the data using SMOTE
os = SMOTE()
x_train, y_train = os.fit_resample(x_train, y_train)
count = Counter(y_train)
print(count) #over sampling - add the additional rows
```

```
Counter({0.0: 86876, 1.0: 86876})
```

In [28]:
```
#Separating the target variable
    #decsion tree

X = rainaus.values[:, :-1] #traindataset
Y = rainaus.values[:, -1] # test

scaler = StandardScaler()
X = scaler.fit_transform(X)

cols = rainaus.columns[:-1]

# Splitting the dataset into train and test
X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.3, ra

clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100,max_d

clf_gini.fit(X_train, y_train)
```

Out[28]:
```
DecisionTreeClassifier(max_depth=4, min_samples_leaf=5, random_state=100)
```

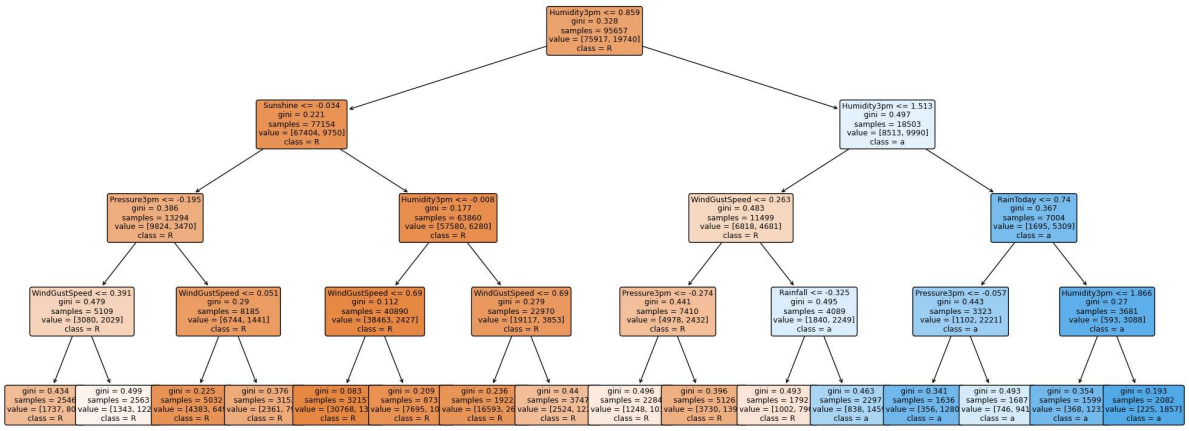
In [29]:
```
# Predicton on test with giniIndex
y_pred = clf_gini.predict(X_test)
print("Predicted values:")
print(y_pred)
```

```
Predicted values:
[0. 0. 0. ... 0. 0. 0.]
```

In [30]:
```
print("Confusion Matrix:\n ",confusion_matrix(y_test, y_pred))

print ("Accuracy : ",accuracy_score(y_test,y_pred)*100)

print("Report : ",classification_report(y_test, y_pred))
```

```
Confusion Matrix:
  [[31467  1119]
 [ 5570  2840]]
Accuracy :  83.68377402673431
Report :                precision    recall  f1-score   support

         0.0       0.85      0.97      0.90     32586
         1.0       0.72      0.34      0.46      8410

    accuracy                           0.84     40996
   macro avg       0.78      0.65      0.68     40996
weighted avg       0.82      0.84      0.81     40996
```

In [31]:
```
from sklearn import tree
plt.figure(figsize = (25,10))
tree.plot_tree(clf_gini, feature_names = cols, class_names ='RainTomorrow', fi
plt.show()
```



In [32]:
```
#Separating the target variable
    #knn
X = rainaus.values[:, :-1]
Y =rainaus.values[:, -1]

scaler = StandardScaler()
X = scaler.fit_transform(X)

# Splitting the dataset into train and test
X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.3, ra

knn = KNeighborsClassifier(n_neighbors=500, metric="euclidean")
```

In [33]:
```
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(y_pred)
```

```
[0. 0. 0. ... 0. 0. 0.]
```

In [34]:
```
print("Confusion Matrix:\n ",confusion_matrix(y_test, y_pred))

print ("Accuracy : ",accuracy_score(y_test,y_pred)*100)

print("Report : ",classification_report(y_test, y_pred))
```

```
Confusion Matrix:
  [[31956   630]
 [ 6221  2189]]
Accuracy :  83.28861352327056
Report :                precision    recall  f1-score   support

         0.0       0.84      0.98      0.90     32586
         1.0       0.78      0.26      0.39      8410

    accuracy                           0.83     40996
   macro avg       0.81      0.62      0.65     40996
weighted avg       0.82      0.83      0.80     40996
```

In [35]:
```
# Gaussian Naive Bayes
from sklearn import datasets
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
GNB = GaussianNB()
GNB.fit(X_train, y_train)
GNB_predicted_labels = GNB.predict(X_train)
y_pred  =  GNB.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print("confusion matrix")
print(cm)
print(classification_report(y_test,y_pred))

GNB_predicted_labels = GNB.predict(X_test)

print("Model Accuracy with Testing data: {0:.4f}".format(metrics.accuracy_scor
print()
```

```
confusion matrix
[[28279  4307]
 [ 3672  4738]]
              precision    recall  f1-score   support

         0.0       0.89      0.87      0.88     32586
         1.0       0.52      0.56      0.54      8410

    accuracy                           0.81     40996
   macro avg       0.70      0.72      0.71     40996
weighted avg       0.81      0.81      0.81     40996

Model Accuracy with Testing data: 80.5371
```

In [36]:
```
rainaus.to_csv("C:/Users/palva/Desktop/DM project/weather.csv", index = False)
rainaus.head(20)
```

Out[36]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSp
0	2	13.4	13.4	0.600000	5.468232	7.611178	13	44.00
1	2	7.4	7.4	0.000000	5.468232	7.611178	14	44.00
2	2	12.9	12.9	0.000000	5.468232	7.611178	15	46.00
3	2	9.2	9.2	0.000000	5.468232	7.611178	4	24.00
4	2	17.5	17.5	1.000000	5.468232	7.611178	13	41.00
5	2	14.6	14.6	0.200000	5.468232	7.611178	14	56.00
6	2	14.3	14.3	0.000000	5.468232	7.611178	13	50.00
7	2	7.7	7.7	0.000000	5.468232	7.611178	13	35.00
9	2	13.1	13.1	1.400000	5.468232	7.611178	13	28.00
10	2	13.4	13.4	0.000000	5.468232	7.611178	3	30.00
11	2	15.9	15.9	2.200000	5.468232	7.611178	5	31.00
13	2	12.6	12.6	3.600000	5.468232	7.611178	12	44.00
14	2	8.4	8.4	0.000000	5.468232	7.611178	13	40.03
15	2	9.8	9.8	2.360918	5.468232	7.611178	14	50.00
16	2	14.1	14.1	0.000000	5.468232	7.611178	1	22.00
17	2	13.5	13.5	16.800000	5.468232	7.611178	13	63.00
18	2	11.2	11.2	10.600000	5.468232	7.611178	10	43.00
19	2	9.8	9.8	0.000000	5.468232	7.611178	10	26.00
20	2	11.5	11.5	0.000000	5.468232	7.611178	8	24.00
21	2	17.1	17.1	0.000000	5.468232	7.611178	4	43.00

In [37]:
```
rainaus.shape
```

Out[37]: (136653, 19)

weather dataset use chesinam and save kudda chesinam in future we find this this heading is a help to find

In [ ]: