In [1]:
```
# rainfall prediction in australia for the next day using 10 years of data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import seaborn as sns
from keras.layers import Dense, BatchNormalization, Dropout, LSTM
from keras.models import Sequential
from keras.utils import to_categorical
from keras.optimizers import Adam
from tensorflow.keras import regularizers
from sklearn.metrics import precision_score, recall_score, confusion_matrix
from keras import callbacks
from sklearn.metrics import confusion_matrix
```

In [2]:
```
data = pd.read_csv("/content/drive/MyDrive/weatherAUS.csv")
```

In [ ]:
```
data.head(n=10)
```

Out[3]:

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

10 rows × 23 columns

Basic Data cleaning and data analysis

In [3]:
```
data.shape
```

Out[3]:
```
(145460, 23)
```

In [42]:
```

```

In [5]:
```
data.dtypes
```

Out[5]:
```
Date              object
Location          object
MinTemp          float64
MaxTemp          float64
Rainfall         float64
Evaporation      float64
Sunshine         float64
WindGustDir       object
WindGustSpeed    float64
WindDir9am        object
WindDir3pm        object
WindSpeed9am     float64
WindSpeed3pm     float64
Humidity9am      float64
Humidity3pm      float64
Pressure9am      float64
Pressure3pm      float64
Cloud9am         float64
Cloud3pm         float64
Temp9am          float64
Temp3pm          float64
RainToday         object
RainTomorrow      object
dtype: object
```

In [6]:
```
data.info()
```

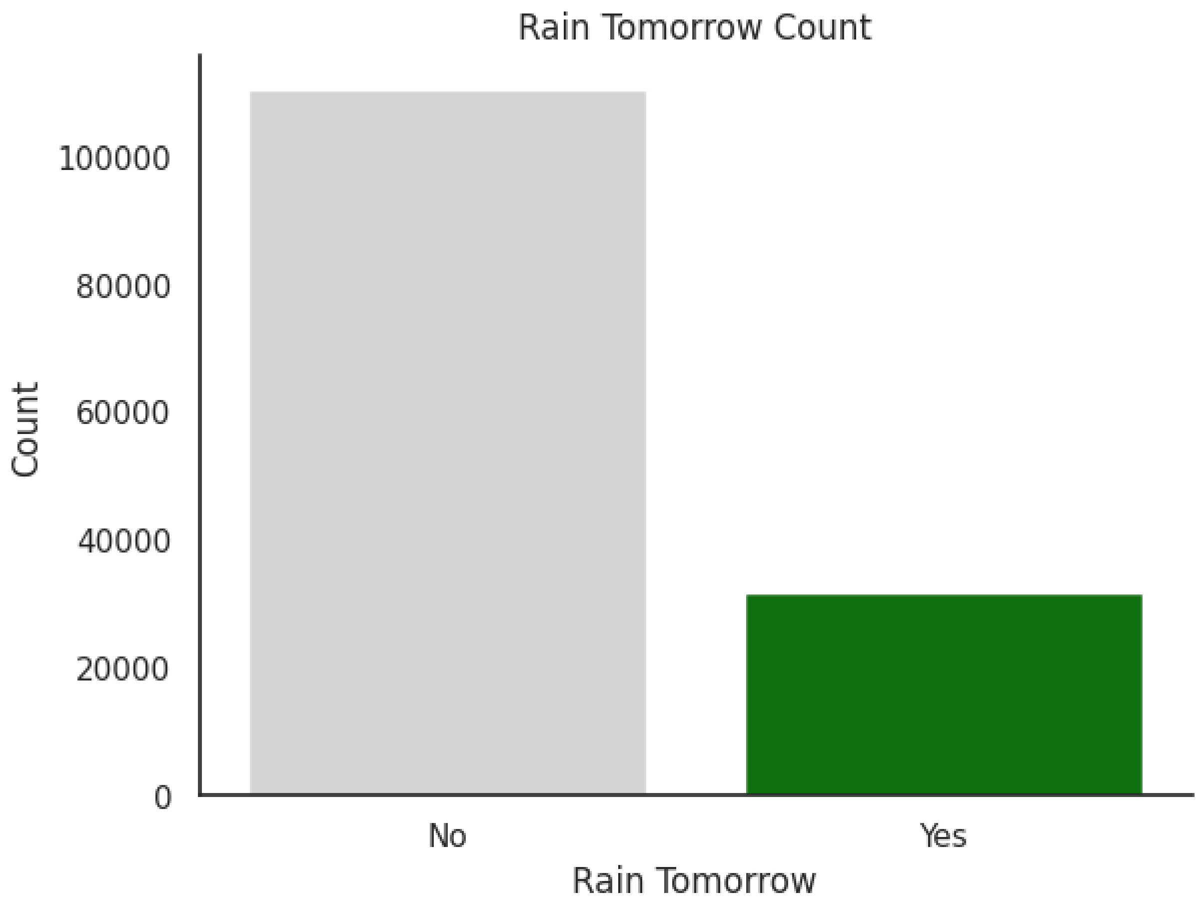
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   Date           145460 non-null  object 
 1   Location       145460 non-null  object 
 2   MinTemp        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 [ ]:

we can see that we have null values in our dataset. Nevertheless first try to get some information from the dataset then will come back to the missing values.

In [ ]:
```
# checking the class balance using simple graph.
sns.set(style="white")
sns.countplot(x= data["RainTomorrow"], palette=["lightgray", "green"] )
sns.despine(top=True, right=True)
plt.title("Rain Tomorrow Count")
plt.xlabel("Rain Tomorrow")
plt.ylabel("Count")
```

Out[9]:
```
Text(0, 0.5, 'Count')
```



In [ ]:

In [ ]:

In [ ]:
```
class_counts = data['RainTomorrow'].value_counts()

# Identify the majority and minority classes
majority_class = class_counts.idxmax()
minority_class = class_counts.idxmin()

# Calculate imbalance ratio
imbalance_ratio = class_counts[majority_class] / class_counts[minority_clas

print(f"Class Counts:\n{class_counts}")
print(f"Imbalance Ratio: {imbalance_ratio:.2f}")
```

```
Class Counts:
No      110316
Yes      31877
Name: RainTomorrow, dtype: int64
Imbalance Ratio: 3.46
```

In [ ]:

Such unbalanced class ratios can have an effect on machine learning methods and model performance.

strategies to deal with unbalanced classes:

Resampling: To balance the class distribution, you might oversample the minority class or undersample the dominant class.

Synthetic Data Generation: Methods such as SMOTE can be used to create synthetic instances of the minority class.

Cost-Sensitive Learning: Change the algorithm's learning process to penalise minority misclassification more severely.

Algorithm Selection: Select algorithms that are built to deal with skewed data, such as ensemble methods or algorithms with built-in class weights.

In [ ]:

We will use ANN as our machine learning method because it works well with unbalanced data as it contains built-in class weights to deal with unbalanced data.

In [ ]:

Coverting date to date-time format

In [7]:
```
data['Date']= pd.to_datetime(data["Date"])
data['year'] = data.Date.dt.year
data['month'] = data.Date.dt.month
data['day']= data.Date.dt.day
```

In [8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 26 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           145460 non-null  datetime64[ns]
 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
 23  year           145460 non-null  int64
 24  month          145460 non-null  int64
 25  day            145460 non-null  int64
dtypes: datetime64[ns](1), float64(16), int64(3), object(6)
memory usage: 28.9+ MB
```

In [ ]:
```
data.head()
```

Out[13]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpe
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	4
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	4
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	4
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	2
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	4

5 rows × 26 columns

In [9]:

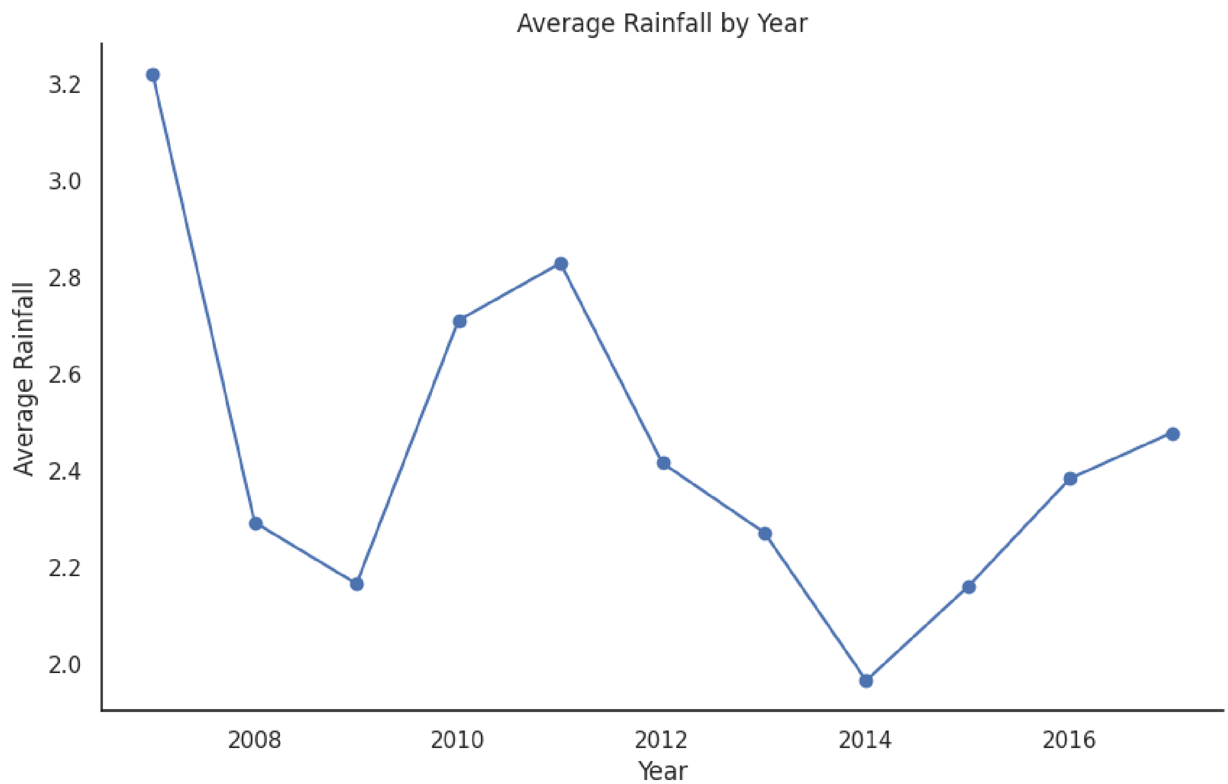
Out[9]:
```
year
2007    25.086885
2008    22.874359
2009    23.251019
2010    22.571247
2011    22.540180
2012    22.311424
2013    23.290936
2014    23.792739
2015    23.515652
2016    23.419984
2017    25.031702
Name: MaxTemp, dtype: float64
```

In [9]:

In [ ]:
```
yearly_avg_rain = data.groupby('year')['Rainfall'].mean()
yearly_avg_rain
```

Out[16]:
```
year
2007    3.219672
2008    2.293541
2009    2.166385
2010    2.710924
2011    2.829197
2012    2.416200
2013    2.272402
2014    1.966341
2015    2.160753
2016    2.384054
2017    2.478834
Name: Rainfall, dtype: float64
```

In [ ]:
```
yearly_avg_rain_df = yearly_avg_rain.reset_index()

plt.figure(figsize=(10, 6))
plt.plot(yearly_avg_rain_df['year'], yearly_avg_rain_df['Rainfall'], marker
sns.despine(top=True, right=True)
plt.title('Average Rainfall by Year')
plt.xlabel('Year')
plt.ylabel('Average Rainfall')

plt.show()
```



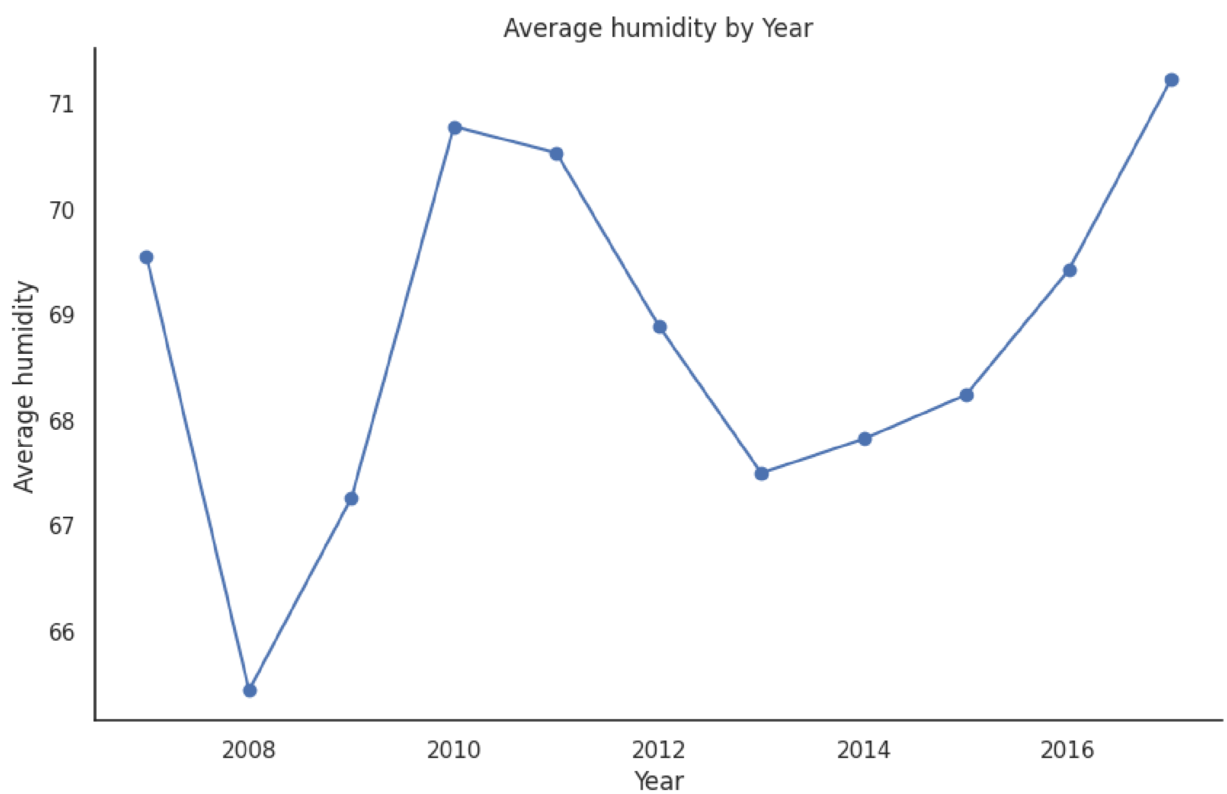
In [9]:
```
```

In [9]:
```
```

In [ ]:
```
yearly_avg_humidity = data.groupby('year')['Humidity9am'].mean()
yearly_avg_humidity
```

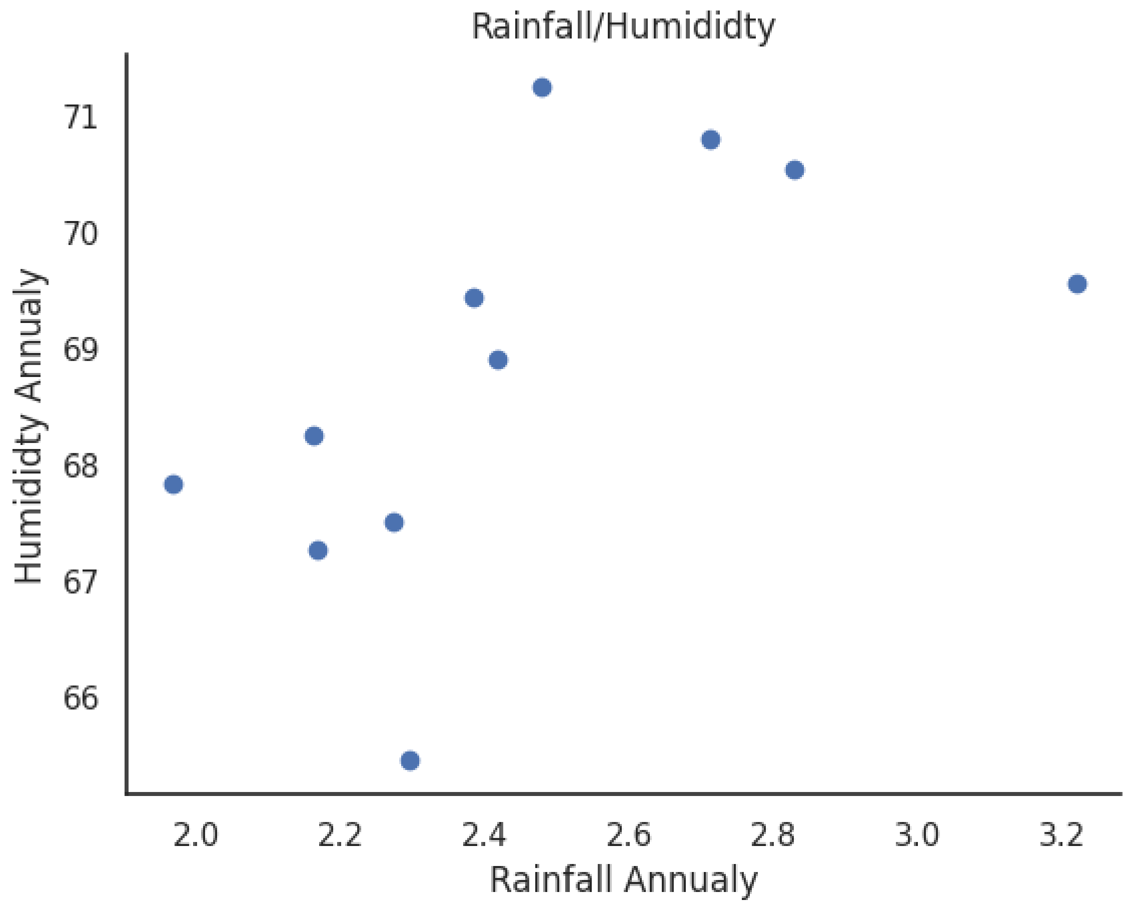
Out[20]:
```
year
2007    69.557377
2008    65.442762
2009    67.266126
2010    70.788379
2011    70.539129
2012    68.896674
2013    67.501632
2014    67.825234
2015    68.244000
2016    69.424885
2017    71.234636
Name: Humidity9am, dtype: float64
```

In [ ]:
```
yearly_avg_humidity_df = yearly_avg_humidity.reset_index()

plt.figure(figsize=(10, 6))
plt.plot(yearly_avg_humidity_df['year'], yearly_avg_humidity_df['Humidity9a
sns.despine(top=True, right=True)
plt.title('Average humidity by Year')
plt.xlabel('Year')
plt.ylabel('Average humidity')

plt.show()
```



In [ ]:
```
plt.scatter(yearly_avg_rain,yearly_avg_humidity )
sns.despine(top=True, right=True)
plt.title('Rainfall/Humididty')
plt.xlabel('Rainfall Annualy')
plt.ylabel('Humididty Annualy')
```

Out[22]:
```
Text(0, 0.5, 'Humididty Annualy')
```



In [11]:
```

```

Type *Markdown* and LaTeX: α^2

The month and year columns must be converted to cyclic format. This allows the neural network to capture cyclical patterns in the data without introducing discontinuities.

In [12]:

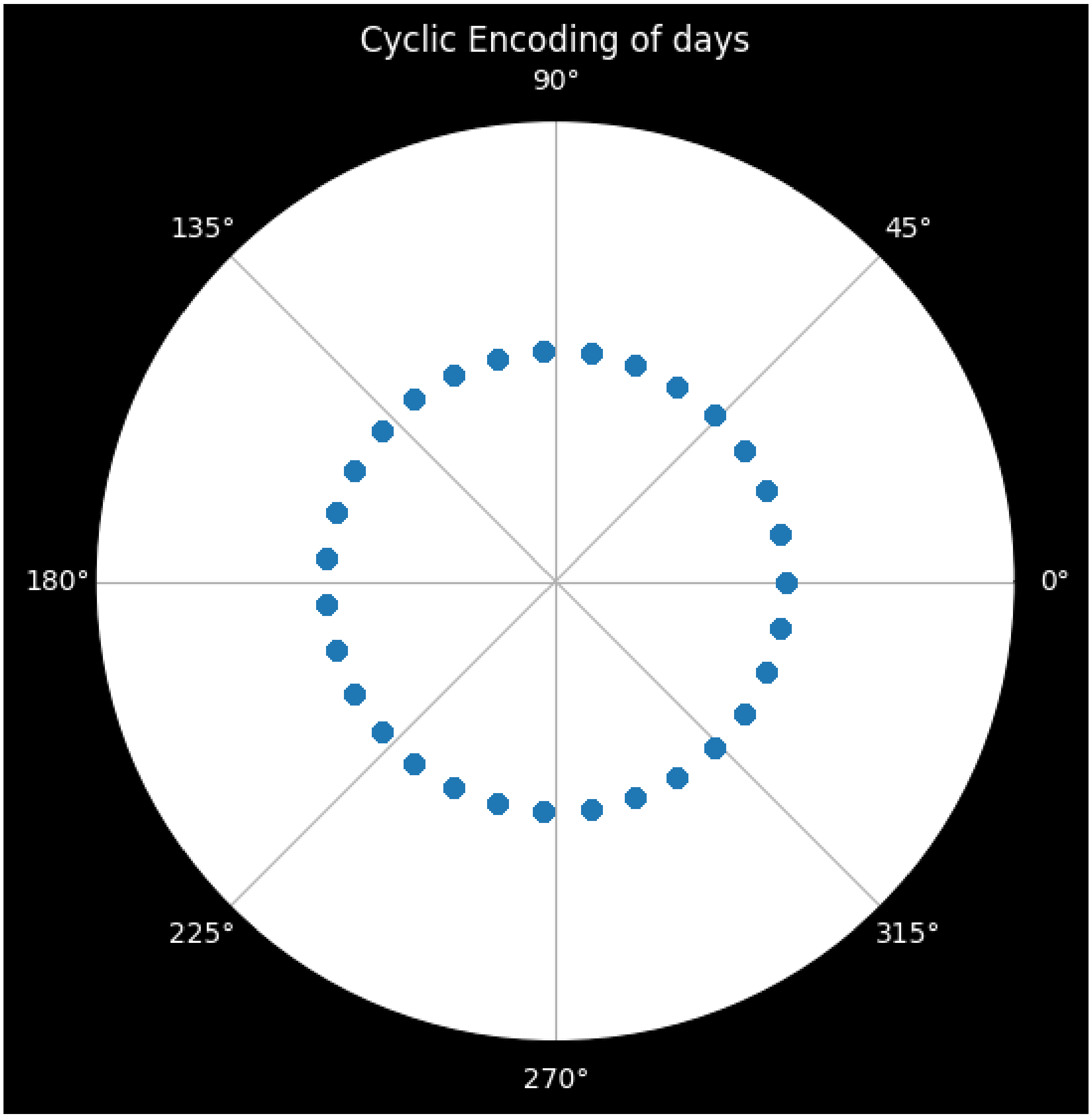
```
# Convert month and year to cyclic representations
def encode_cyclic(value, period):
    return np.sin(2 * np.pi * value / period), np.cos(2 * np.pi * value / p

data[['year_sin', 'year_cos']] = data['year'].apply(lambda x: pd.Series(enc
data[['month_sin', 'month_cos']] = data['month'].apply(lambda x: pd.Series(
data[['day_sin', 'day_cos']] = data['day'].apply(lambda x: pd.Series(encode
```

In [ ]:

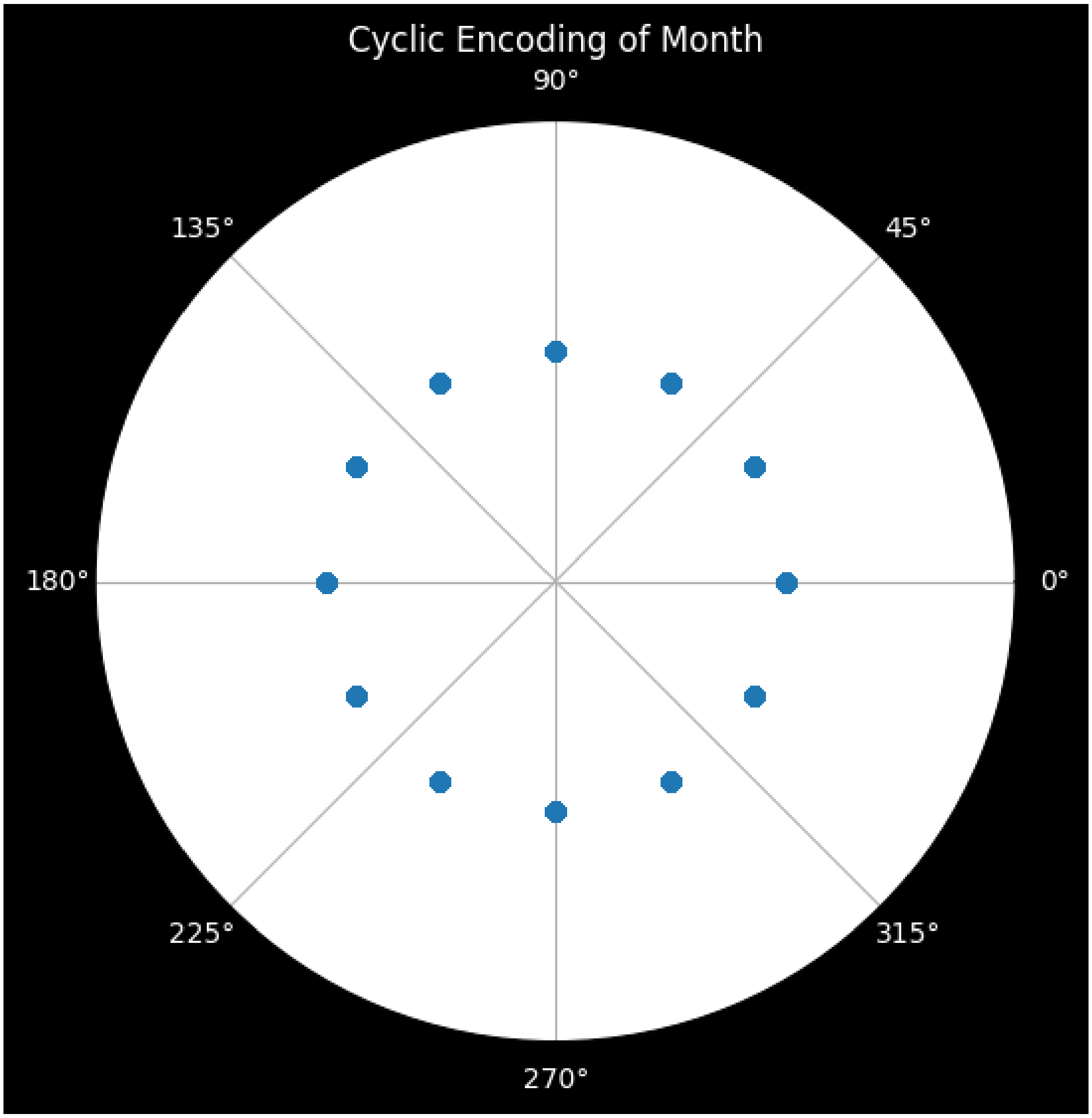
In [ ]:

In [ ]:

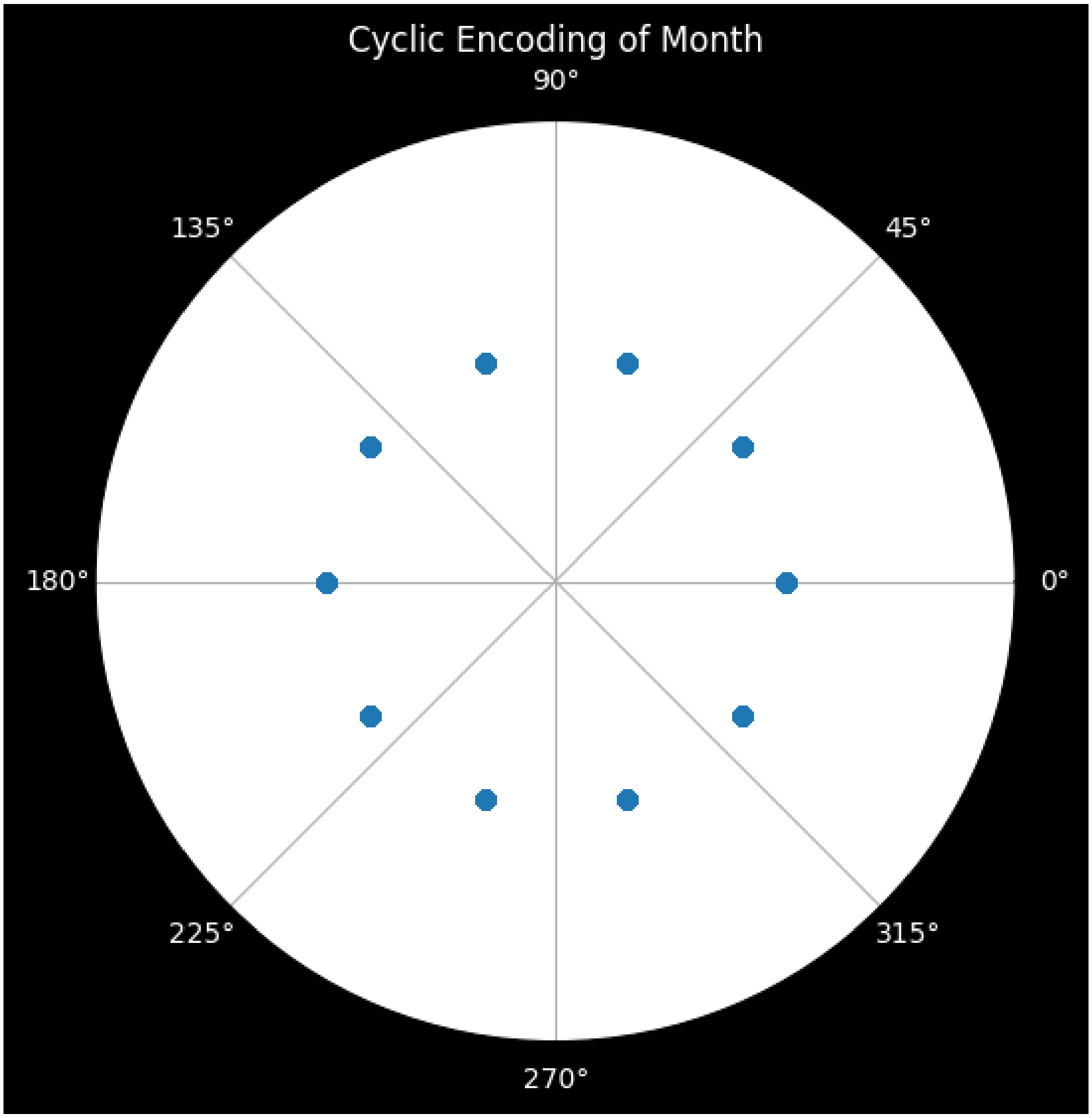
In [13]:
```
fig = plt.figure(figsize=(6, 6))
fig.patch.set_facecolor('black')
ax = fig.add_subplot(111, projection='polar')
ax.plot(np.arctan2(data['day_sin'], data['day_cos']), np.sqrt(np.square(dat

ax.set_title("Cyclic Encoding of days", color='white')  # Set plot title co
ax.tick_params(axis='both', colors='white')
ax.set_rticks([])  # Hide radial tick labels

plt.show()
```



In [14]:
```
fig = plt.figure(figsize=(6, 6))
fig.patch.set_facecolor('black')
ax = fig.add_subplot(111, projection='polar')
ax.plot(np.arctan2(data['month_sin'], data['month_cos']), np.sqrt(np.square
ax.set_title("Cyclic Encoding of Month", color='white')  # Set plot title c
ax.tick_params(axis='both', colors='white')
ax.set_rticks([])  # Hide radial tick labels

plt.show()
```



In [15]:
```
fig = plt.figure(figsize=(6, 6))
fig.patch.set_facecolor('black')
ax = fig.add_subplot(111, projection='polar')
ax.plot(np.arctan2(data['year_sin'], data['year_cos']), np.sqrt(np.square(d
ax.set_title("Cyclic Encoding of Month", color='white')  # Set plot title c
ax.tick_params(axis='both', colors='white')
ax.set_rticks([])  # Hide radial tick labels

plt.show()
```



In [16]:
```
data.isnull().sum()
```

Out[16]:
```
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
year                 0
month                0
day                  0
year_sin             0
year_cos             0
month_sin            0
month_cos            0
day_sin              0
day_cos              0
dtype: int64
```

In [ ]:

As we see missing values in our dataset, we will replace them with mode for categorical variables and meadian for numeric values because mean is heavily influenced by outliers.

In [17]:
```
categorical_missing = (data.dtypes == "object")
object_cols = list(categorical_missing[categorical_missing].index)
```

In [18]:
```
for i in object_cols:
    data[i].fillna(data[i].mode()[0], inplace=True)
```

In [19]:
```
for i in object_cols:
    print(i, data[i].isnull().sum())
```

```
Location 0
WindGustDir 0
WindDir9am 0
WindDir3pm 0
RainToday 0
RainTomorrow 0
```

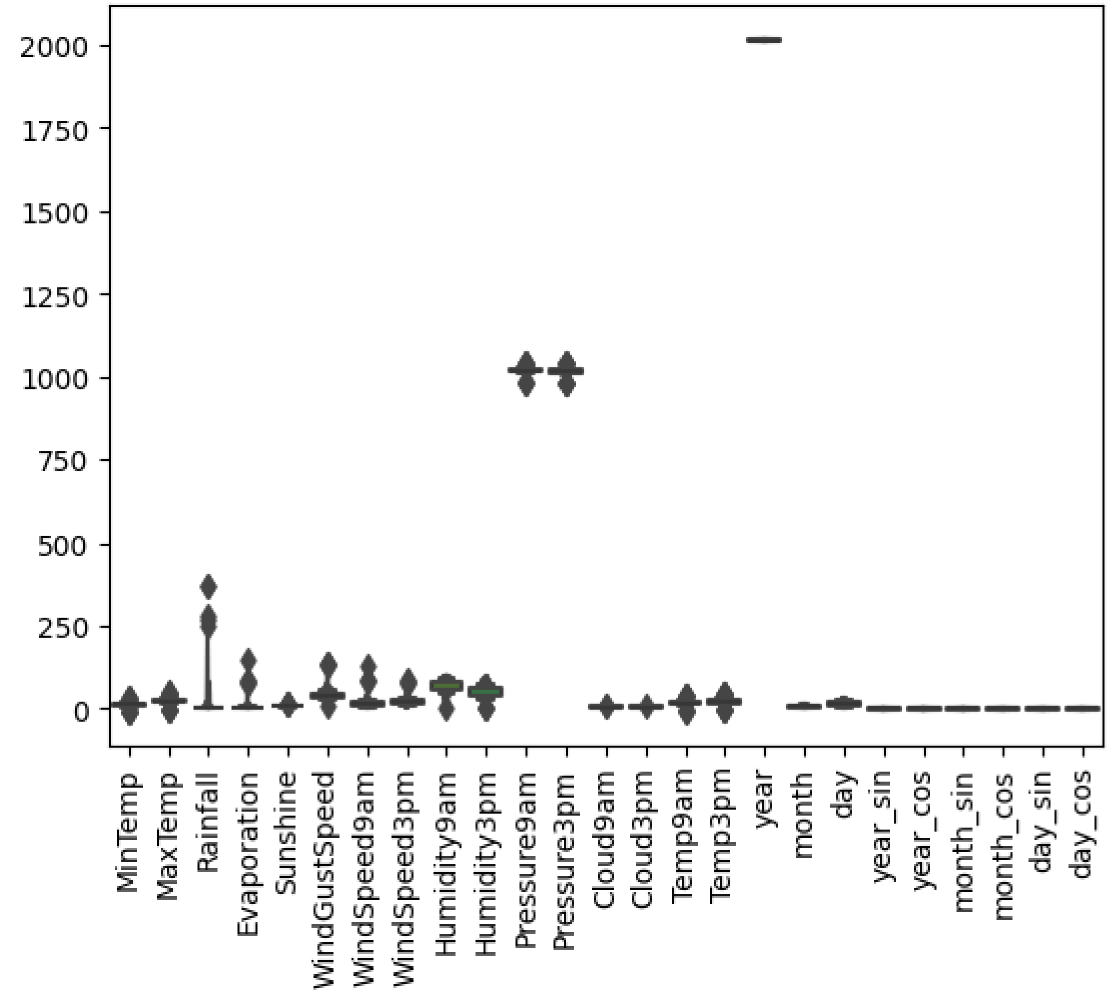
In [20]:
```
numerical_missing = (data.dtypes == "float64")
num_cols = list(numerical_missing[numerical_missing].index)
```

In [ ]:

In [21]:
```
for i in num_cols:
    data[i].fillna(data[i].median(), inplace=True)

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 32 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           145460 non-null  datetime64[ns]
 1   Location       145460 non-null  object
 2   MinTemp        145460 non-null  float64
 3   MaxTemp        145460 non-null  float64
 4   Rainfall       145460 non-null  float64
 5   Evaporation    145460 non-null  float64
 6   Sunshine       145460 non-null  float64
 7   WindGustDir    145460 non-null  object
 8   WindGustSpeed  145460 non-null  float64
 9   WindDir9am     145460 non-null  object
 10  WindDir3pm     145460 non-null  object
 11  WindSpeed9am   145460 non-null  float64
 12  WindSpeed3pm   145460 non-null  float64
 13  Humidity9am    145460 non-null  float64
 14  Humidity3pm    145460 non-null  float64
 15  Pressure9am    145460 non-null  float64
 16  Pressure3pm    145460 non-null  float64
 17  Cloud9am       145460 non-null  float64
 18  Cloud3pm       145460 non-null  float64
 19  Temp9am        145460 non-null  float64
 20  Temp3pm        145460 non-null  float64
 21  RainToday      145460 non-null  object
 22  RainTomorrow   145460 non-null  object
 23  year           145460 non-null  int64
 24  month          145460 non-null  int64
 25  day            145460 non-null  int64
 26  year_sin       145460 non-null  float64
 27  year_cos       145460 non-null  float64
 28  month_sin      145460 non-null  float64
 29  month_cos      145460 non-null  float64
 30  day_sin        145460 non-null  float64
 31  day_cos        145460 non-null  float64
dtypes: datetime64[ns](1), float64(22), int64(3), object(6)
memory usage: 35.5+ MB
```

In [22]:
```
sns.boxenplot(data = data)
plt.xticks(rotation=90)
plt.show()
```



In [ ]:

As we can see, our variables have a wide range, making it difficult to spot outliers before scaling the data. As we have string values (city names), we will utilise label encoding.

In [ ]:

In [23]:

```
label_encoder = LabelEncoder()
for i in object_cols:
    data[i] = label_encoder.fit_transform(data[i])

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 32 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           145460 non-null  datetime64[ns]
 1   Location       145460 non-null  int64
 2   MinTemp        145460 non-null  float64
 3   MaxTemp        145460 non-null  float64
 4   Rainfall       145460 non-null  float64
 5   Evaporation    145460 non-null  float64
 6   Sunshine       145460 non-null  float64
 7   WindGustDir    145460 non-null  int64
 8   WindGustSpeed  145460 non-null  float64
 9   WindDir9am     145460 non-null  int64
 10  WindDir3pm     145460 non-null  int64
 11  WindSpeed9am   145460 non-null  float64
 12  WindSpeed3pm   145460 non-null  float64
 13  Humidity9am    145460 non-null  float64
 14  Humidity3pm    145460 non-null  float64
 15  Pressure9am    145460 non-null  float64
 16  Pressure3pm    145460 non-null  float64
 17  Cloud9am       145460 non-null  float64
 18  Cloud3pm       145460 non-null  float64
 19  Temp9am        145460 non-null  float64
 20  Temp3pm        145460 non-null  float64
 21  RainToday      145460 non-null  int64
 22  RainTomorrow   145460 non-null  int64
 23  year           145460 non-null  int64
 24  month          145460 non-null  int64
 25  day            145460 non-null  int64
 26  year_sin       145460 non-null  float64
 27  year_cos       145460 non-null  float64
 28  month_sin      145460 non-null  float64
 29  month_cos      145460 non-null  float64
 30  day_sin        145460 non-null  float64
 31  day_cos        145460 non-null  float64
dtypes: datetime64[ns](1), float64(22), int64(9)
memory usage: 35.5 MB
```

In [23]:

```

```

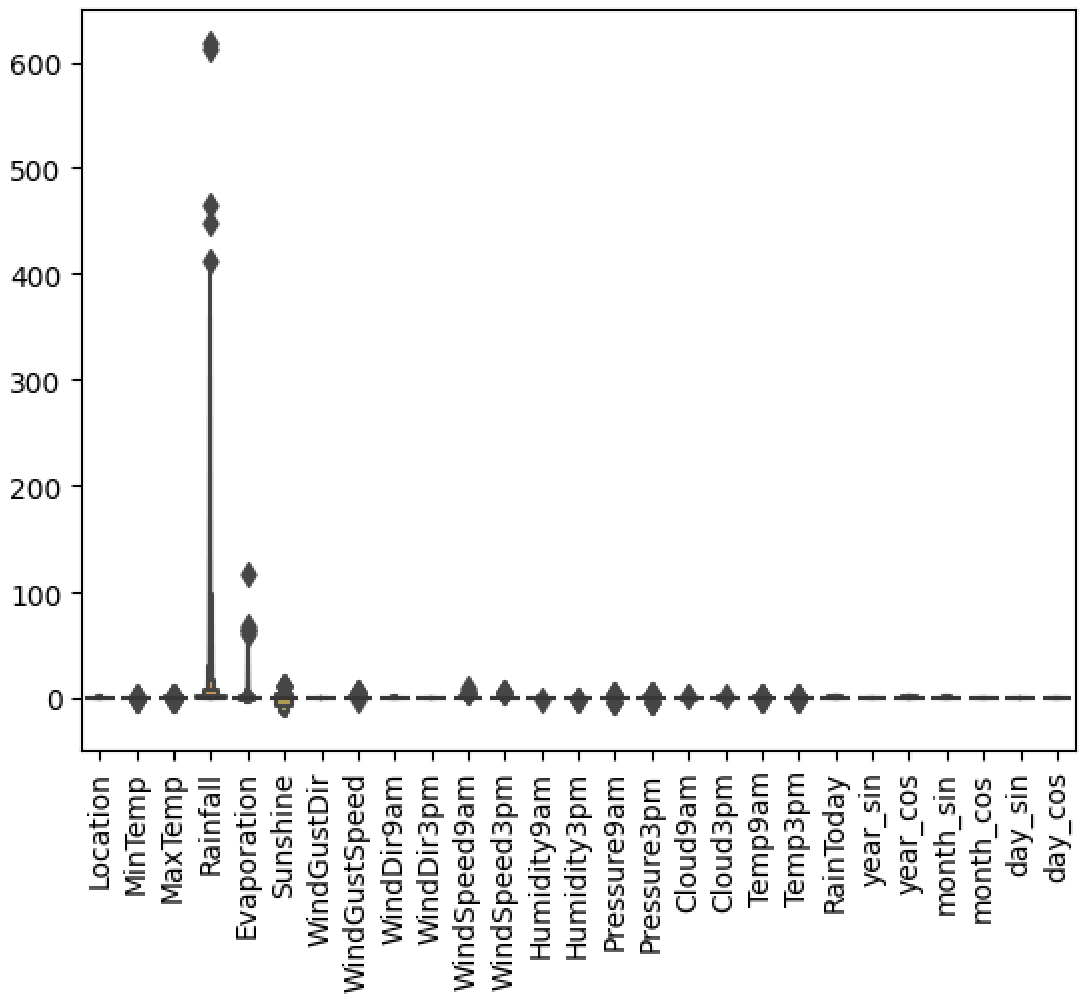
In [24]:

```
features = data.drop(['RainTomorrow', 'day', 'month', 'year','Date'], axis
target = data['RainTomorrow']
```

In [25]:
```
col_names = list(features.columns)
robust = RobustScaler()
features = robust.fit_transform(features)
features = pd.DataFrame(features, columns=col_names)
```

Weather variables can have outliers due to extreme weather events in the context of weather data prediction. In such circumstances, robust scaling may be a better option to avoid outliers skewing the training process.

In [27]:
```
sns.boxenplot(data = features)
plt.xticks(rotation=90)
plt.show()
```

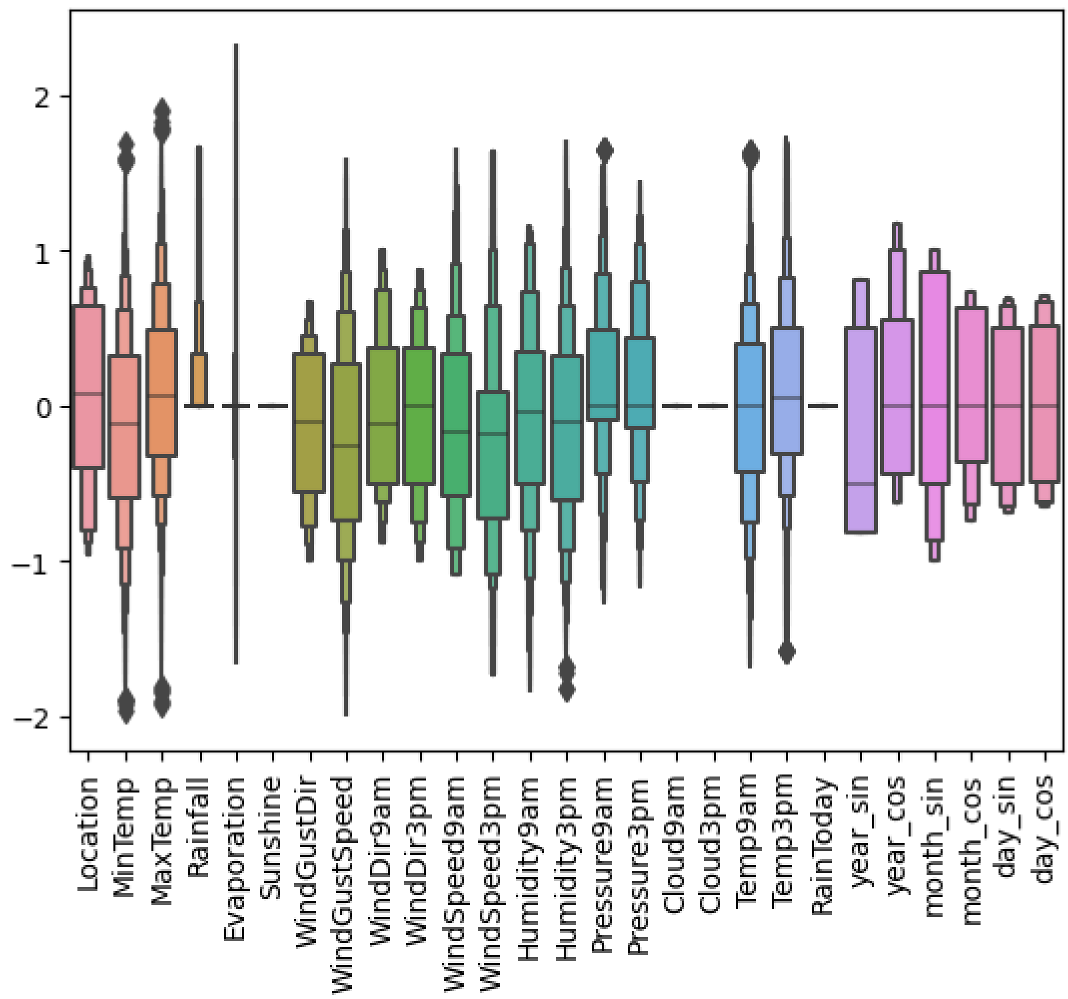


In [28]:

```
# Function to remove outliers using IQR method
def remove_outliers_iqr(features, threshold=1.5):
    for col in features.columns:
        Q1 = features[col].quantile(0.25)
        Q3 = features[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - threshold * IQR
        upper_bound = Q3 + threshold * IQR
        features = features[(features[col] >= lower_bound) & (features[col]
    return features

# Remove outliers using IQR method
df_no_outliers = remove_outliers_iqr(features)
```

In [29]:

```
sns.boxenplot(data = df_no_outliers)
plt.xticks(rotation=90)
plt.show()
```



In [ ]:

```
len(X_train.columns)
```

Out[103]: 27

In [31]:

In [34]:

```python
X_train, X_test, y_train, y_test = train_test_split(features, target, test_
```

Early stopping is a technique used to prevent overfitting by monitoring the validation loss during training. If the validation loss starts increasing or stagnates after a certain number of epochs, early stopping halts the training process. This prevents the model from continuing to learn on the training data past the point where it starts to overfit.

In [35]:

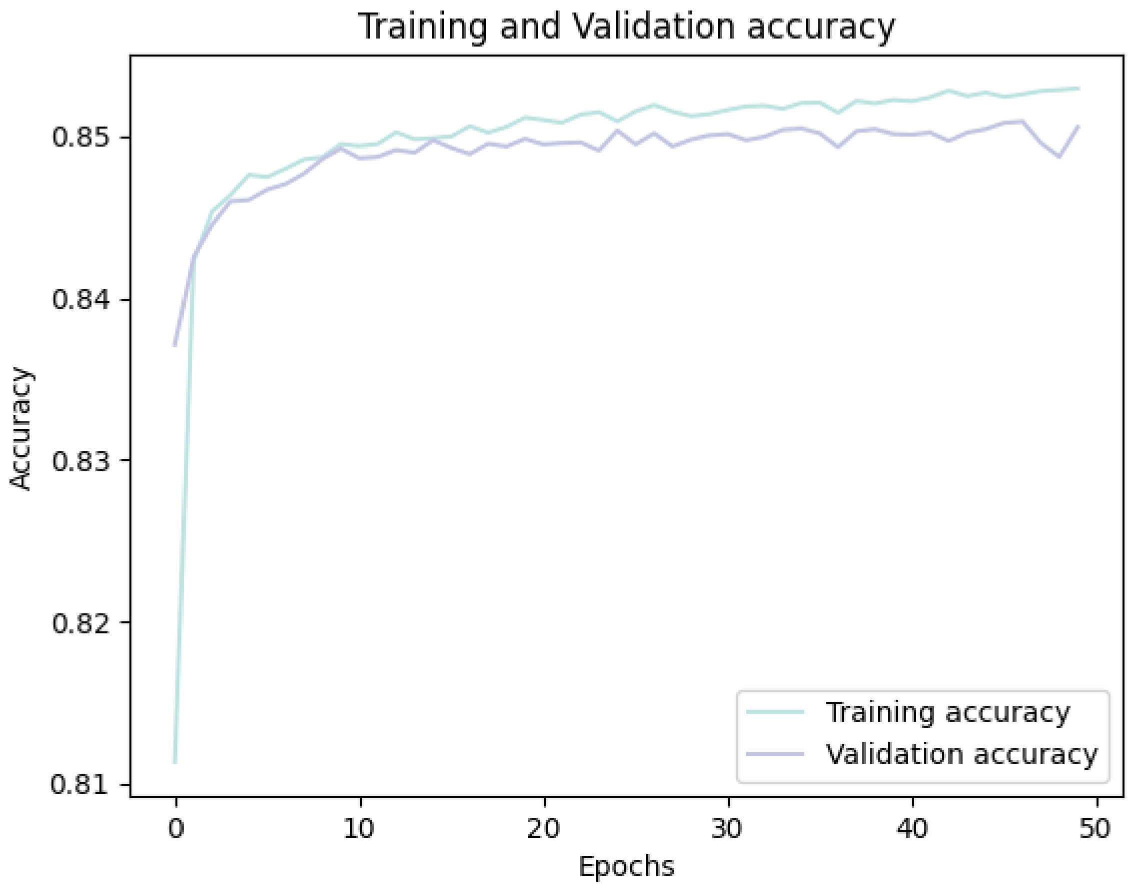
```python
early_stopping = callbacks.EarlyStopping(
    min_delta=0.001,
    patience=20,
    restore_best_weights=True,
)
```

In [36]:

```python
model2 = Sequential()
model2.add(Dense(units = 27 , kernel_initializer = 'uniform', activation = 
model2.add(Dense(units = 27, kernel_initializer = 'uniform', activation = '
model2.add(Dense(units = 13, kernel_initializer= 'uniform', activation = 'r
model2.add(Dense(units = 7, kernel_initializer = 'uniform', activation = 'r
model2.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 's
opt = Adam(learning_rate=0.00009)
model2.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['a
history2=model2.fit(X_train, y_train, epochs = 50 ,callbacks=[early_stoppin
```

```
Epoch 1/50
2910/2910 [==============================] - 10s 3ms/step - loss: 0.4553
- accuracy: 0.8113 - val_loss: 0.3841 - val_accuracy: 0.8371
Epoch 2/50
2910/2910 [==============================] - 6s 2ms/step - loss: 0.3719 -
accuracy: 0.8423 - val_loss: 0.3683 - val_accuracy: 0.8426
Epoch 3/50
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3620 -
accuracy: 0.8453 - val_loss: 0.3631 - val_accuracy: 0.8445
Epoch 4/50
2910/2910 [==============================] - 7s 2ms/step - loss: 0.3580 -
accuracy: 0.8463 - val_loss: 0.3606 - val_accuracy: 0.8460
Epoch 5/50
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3556 -
accuracy: 0.8476 - val_loss: 0.3591 - val_accuracy: 0.8461
Epoch 6/50
2910/2910 [==============================] - 7s 2ms/step - loss: 0.3541 -
accuracy: 0.8475 - val_loss: 0.3581 - val_accuracy: 0.8467
Epoch 7/50
```

In [37]:
```
history_df2 = pd.DataFrame(history2.history)

plt.plot(history_df2.loc[:, ['accuracy']], "#BDE2E2", label='Training accur
plt.plot(history_df2.loc[:, ['val_accuracy']], "#C2C4E2", label='Validation

plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Overfitting and underfitting are checked using training and validation accuracy, as well as other metrics.

As the accuracy of training and validation varied slightly, our model is not overfitting. The fact that our model's training and validation accuracy are both quite good shows that it is learning patterns in the data.

In [39]:
```
model2.evaluate(X_test, y_test)
```

```
910/910 [==============================] - 3s 3ms/step - loss: 0.3487 - a
ccuracy: 0.8467
```

Out[39]:
```
[0.348657488822937, 0.8466588854789734]
```

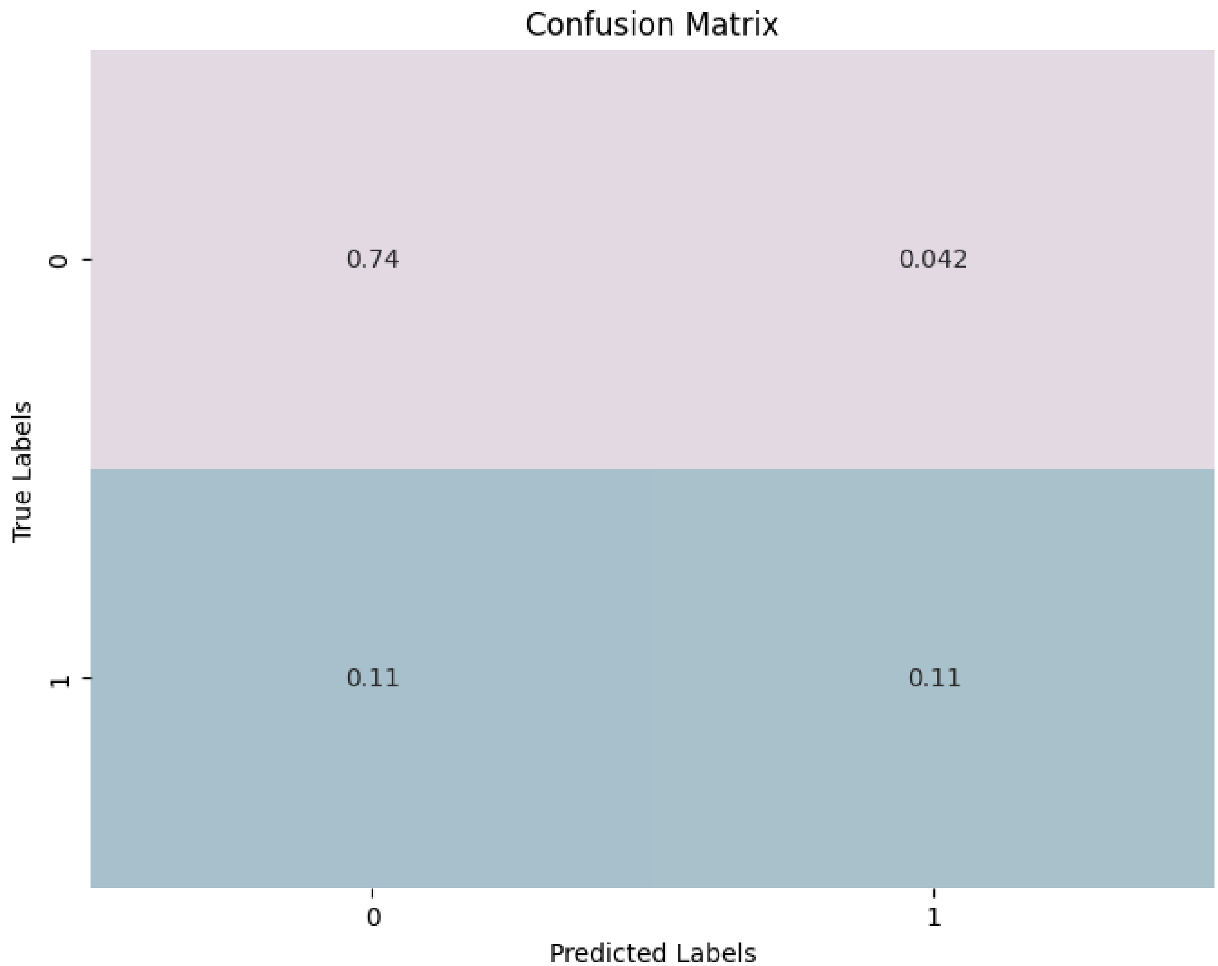
In [40]:

```
y_predict = model2.predict(X_test)
y_pred_classes = (y_predict > 0.5)
```

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

In [41]:

```
cmap = 'twilight'
# Plot the confusion matrix using Seaborn
cm = confusion_matrix(y_test, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(cm/np.sum(cm), annot=True, cmap=cmap, cbar=False)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```



In [42]:
```
report = classification_report(y_test, y_pred_classes)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.87      0.95      0.91     22672
           1       0.72      0.50      0.59      6420

    accuracy                           0.85     29092
   macro avg       0.80      0.72      0.75     29092
weighted avg       0.84      0.85      0.84     29092
```

In [ ]:

A confusion matrix is a tabular representation that summarizes the performance of a classification model. It provides insights into how well the model's predictions match the actual class labels. The matrix displays true positive, true negative, false positive, and false negative counts, enabling assessment of accuracy, precision, recall, and F1-score. It's a valuable tool for understanding a model's strengths and weaknesses in predicting different classes.

In [ ]:

Future Works

Data Quality Enhancement: Focus on data cleaning, validation, and augmentation to ensure the quality of input data, which directly impacts the accuracy of predictions.

Performance Evaluation: Conduct thorough validation and comparison with existing rainfall prediction models to assess the effectiveness and superiority of our approach.

Longer Time Horizons: Extend the prediction horizon to forecast rainfall on longer time scales, such as monthly or seasonal predictions, to cater to different planning and decision-making needs.

Advanced Machine Learning Techniques: Explore more advanced techniques, or hybrid model to futhure improve prediction accuracy.

In [ ]:

Thank you

In [ ]: