Importing Libraries

In [1]:
```
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, cla
from keras import callbacks
import pandas as pd
import numpy as np
```

Loading Data

In [2]:
```
data = pd.read_csv(r"C:\Users\chitt\Downloads\weatherAUS.csv\weatherAUS.csv")
```

In [3]:
```
data
```

Out[3]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindG
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	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	
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	
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	
145459	2017-06-25	Uluru	14.9	NaN	0.0	NaN	NaN	

145460 rows × 23 columns

In [4]:
```
data.head()
```

Out[4]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir
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 rows × 23 columns

In [5]:
```
data.info()
```

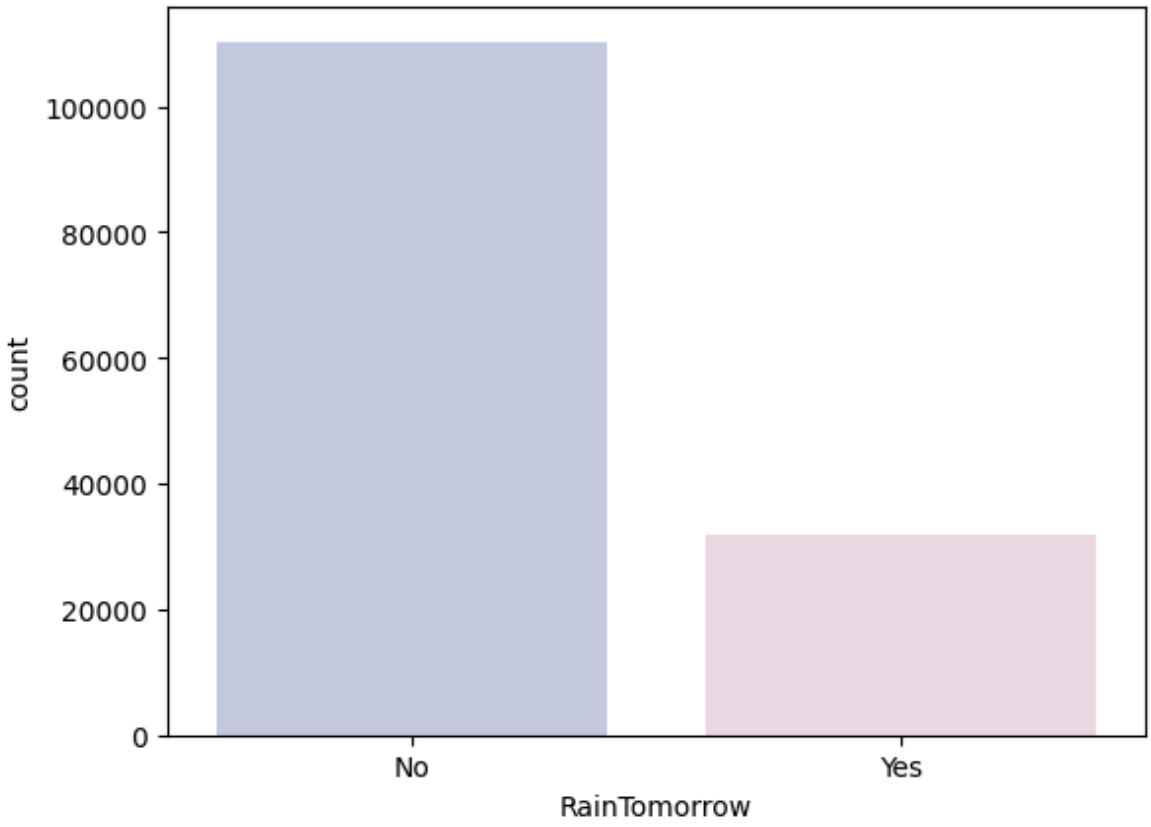
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   Date           145460 non-null  object 
 1   Location       145460 non-null  object 
 2   MinTemp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

Data Visualization And Cleaning

In [6]:
```
#first of all let us evaluate the target and find out if our data is imbalanced
cols= ["#C2C4E2","#EED4E5"]
sns.countplot(x= data["RainTomorrow"], palette= cols)
```

```
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\3211194584.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.countplot(x= data["RainTomorrow"], palette= cols)
```

Out[6]:
```
<Axes: xlabel='RainTomorrow', ylabel='count'>
```



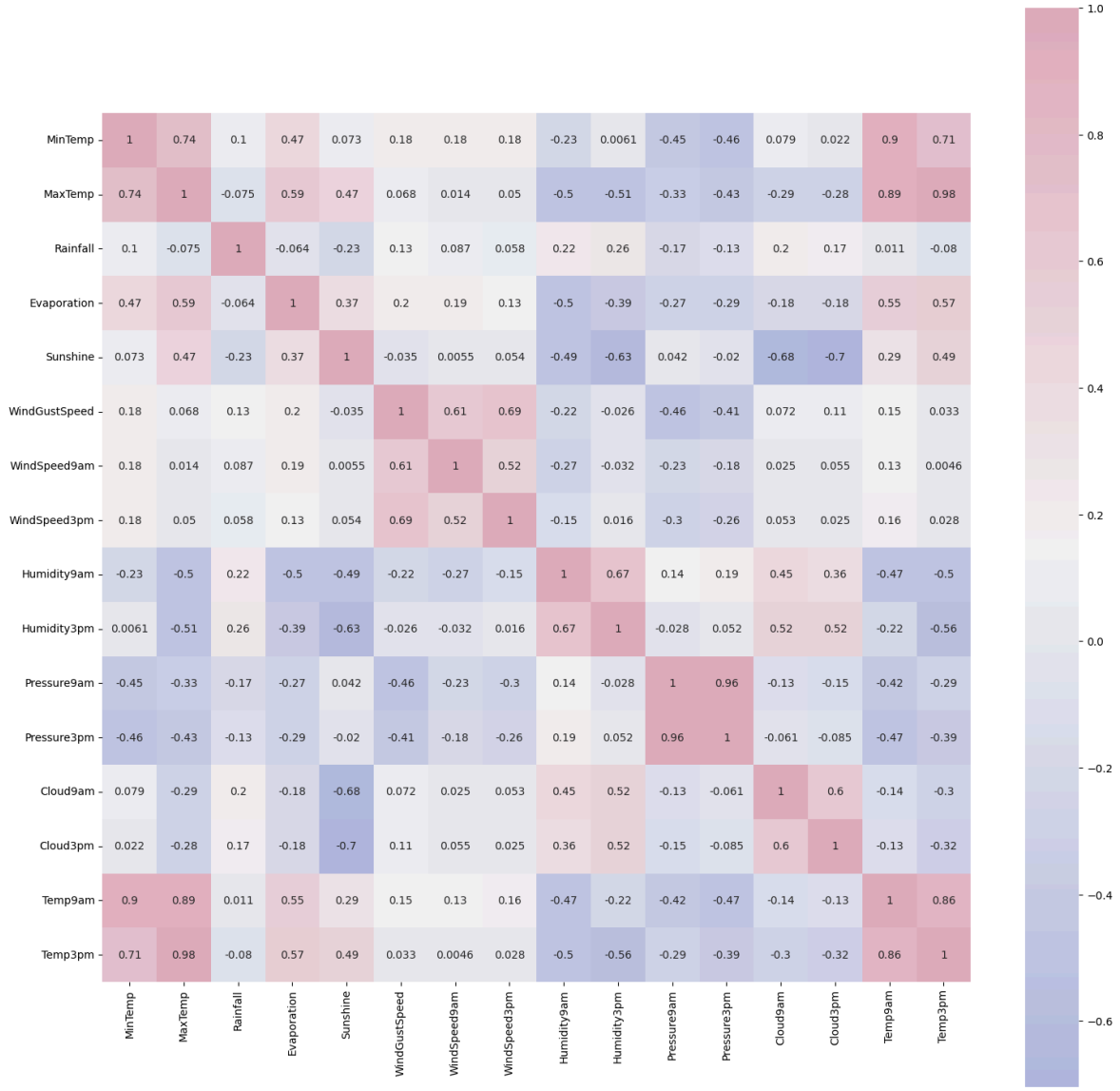
In [8]:
```
# Get list of neumeric variables
t = (data.dtypes == "float64")
num_cols = list(t[t].index)

print("Neumeric variables:")
print(num_cols)
```

```
Neumeric variables:
['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'W
indSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Press
ure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm']
```

In [9]:
```
# Missing values in numeric variables

for i in num_cols:
    print(i, data[i].isnull().sum())
```

```
MinTemp 1485
MaxTemp 1261
Rainfall 3261
Evaporation 62790
Sunshine 69835
WindGustSpeed 10263
WindSpeed9am 1767
WindSpeed3pm 3062
Humidity9am 2654
Humidity3pm 4507
Pressure9am 15065
Pressure3pm 15028
Cloud9am 55888
Cloud3pm 59358
Temp9am 1767
Temp3pm 3609
```

In [10]:
```
# Filter numeric columns only
numeric_data = data.select_dtypes(include='number')

# Calculate the correlation matrix
corrmat = numeric_data.corr()

# Define the colormap
cmap = sns.diverging_palette(260, -10, s=50, l=75, n=6, as_cmap=True)

# Plot the heatmap
plt.figure(figsize=(18, 18))
sns.heatmap(corrmat, cmap=cmap, annot=True, square=True)
plt.show()
```



In [11]:
```
#Parsing datetime
#exploring the length of date objects
lengths = data["Date"].str.len()
lengths.value_counts()
```

Out[11]:
```
Date
10    145460
Name: count, dtype: int64
```

In [12]:
```
#There don't seem to be any error in dates so parsing values into datetime
data['Date']= pd.to_datetime(data["Date"])
```

```
#Creating a collumn of year
data['year'] = data.Date.dt.year

# function to encode datetime into cyclic parameters.
#As I am planning to use this data in a neural network I prefer the months and d

def encode(data, col, max_val):
    data[col + '_sin'] = np.sin(2 * np.pi * data[col]/max_val)
    data[col + '_cos'] = np.cos(2 * np.pi * data[col]/max_val)
    return data

data['month'] = data.Date.dt.month
data = encode(data, 'month', 12)
data['day'] = data.Date.dt.day
data = encode(data, 'day', 31)

data.head()
```

Out[12]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir
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

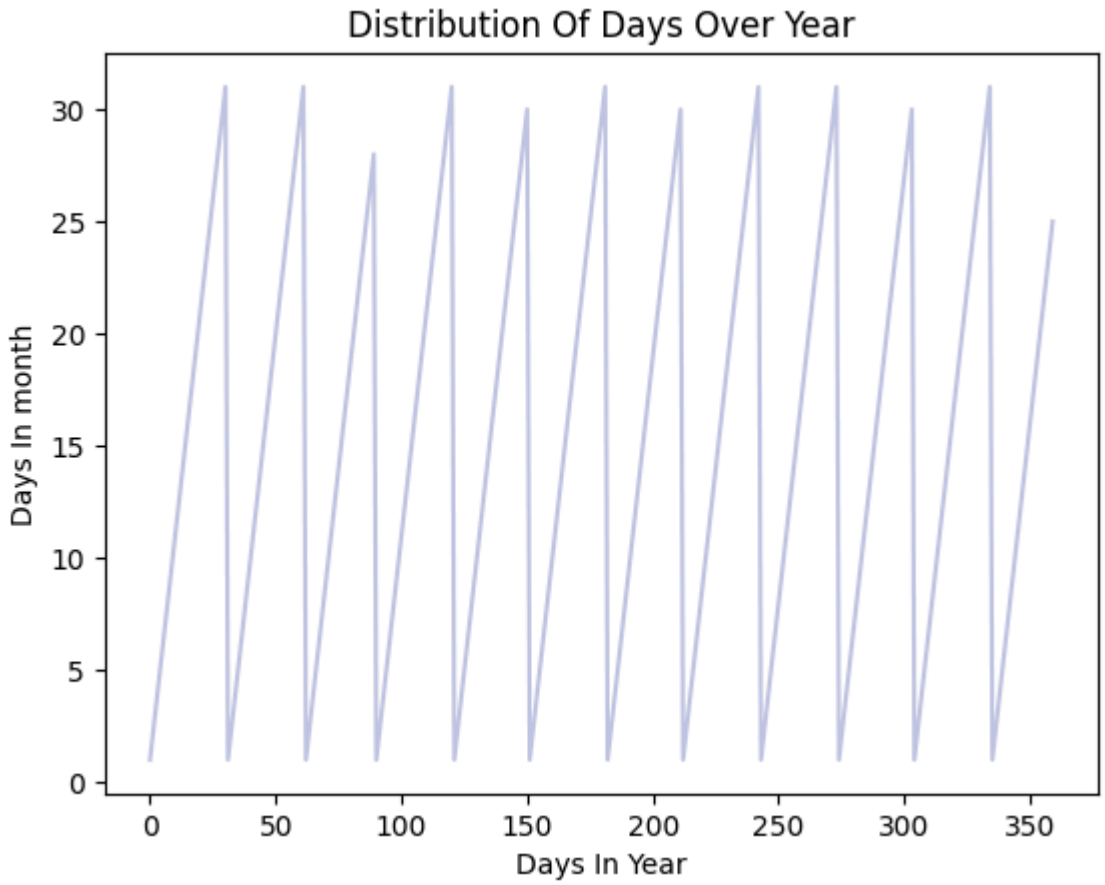
5 rows × 30 columns

In [13]:

```
# roughly a year's span section
section = data[:360]
tm = section["day"].plot(color="#C2C4E2")
tm.set_title("Distribution Of Days Over Year")
tm.set_ylabel("Days In month")
tm.set_xlabel("Days In Year")
```

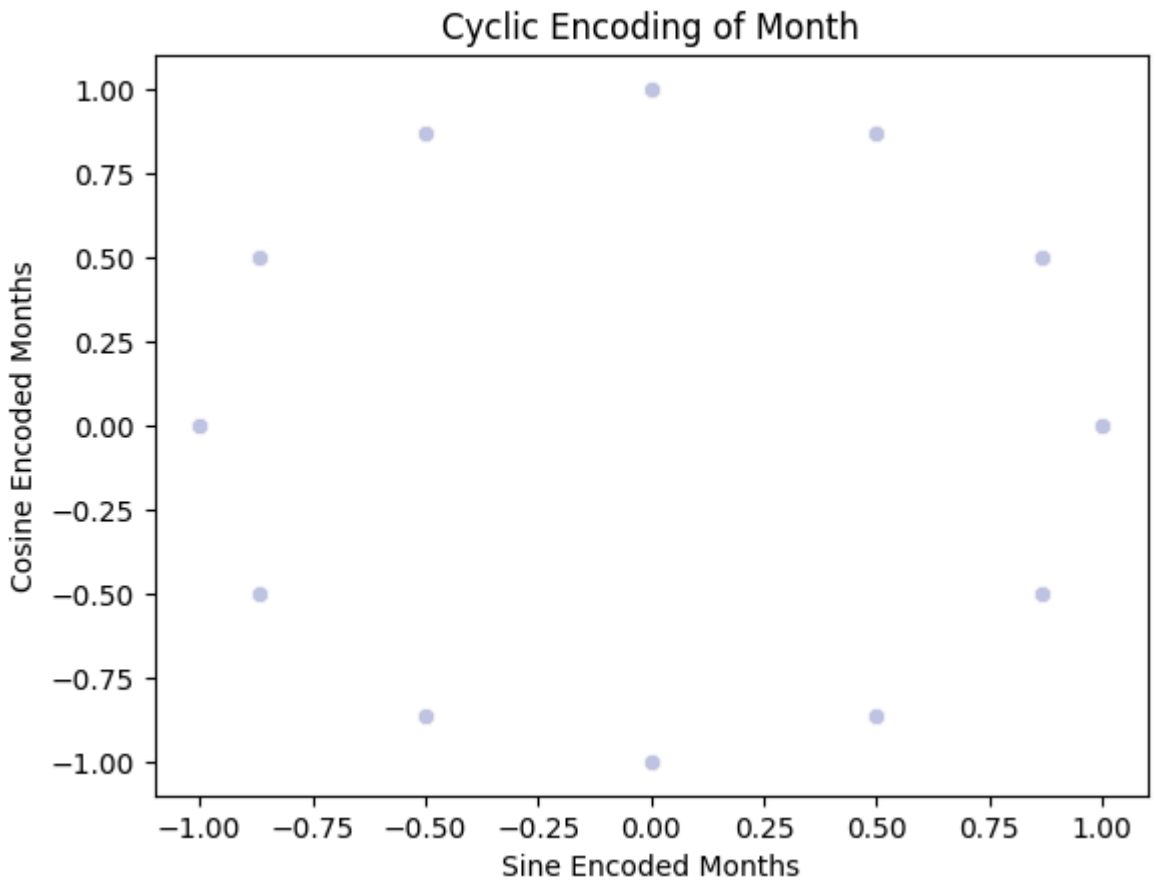
Out[13]:

```
Text(0.5, 0, 'Days In Year')
```



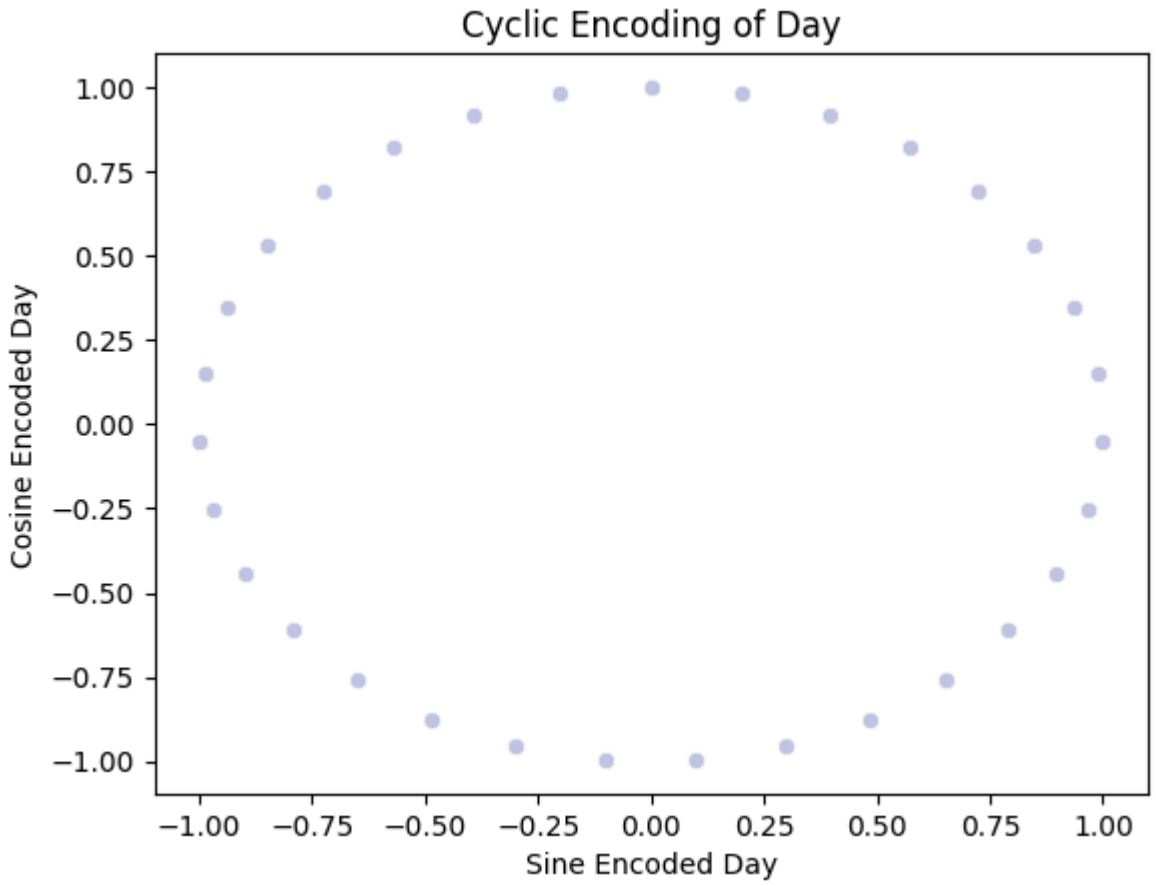
In [14]:
```
cyclic_month = sns.scatterplot(x="month_sin",y="month_cos",data=data, color="#C2
cyclic_month.set_title("Cyclic Encoding of Month")
cyclic_month.set_ylabel("Cosine Encoded Months")
cyclic_month.set_xlabel("Sine Encoded Months")
```

Out[14]:
```
Text(0.5, 0, 'Sine Encoded Months')
```



In [15]:
```
cyclic_day = sns.scatterplot(x='day_sin',y='day_cos',data=data, color="#C2C4E2")
cyclic_day.set_title("Cyclic Encoding of Day")
cyclic_day.set_ylabel("Cosine Encoded Day")
cyclic_day.set_xlabel("Sine Encoded Day")
```

Out[15]:
```
Text(0.5, 0, 'Sine Encoded Day')
```



In [16]:
```
# Get list of categorical variables
s = (data.dtypes == "object")
object_cols = list(s[s].index)

print("Categorical variables:")
print(object_cols)
```

```
Categorical variables:
['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday', 'RainTomorro
w']
```

In [17]:
```
# Missing values in categorical variables

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

```
Location 0
WindGustDir 10326
WindDir9am 10566
WindDir3pm 4228
RainToday 3261
RainTomorrow 3267
```

In [18]:
```
for i in object_cols:
    mode_value = data[i].mode()[0]  # Calculate the mode
    data[i] = data[i].fillna(mode_value)  # Fill missing values explicitly
```

In [19]:
```
# Get list of neumeric variables
t = (data.dtypes == "float64")
num_cols = list(t[t].index)
```

```
print("Neumeric variables:")
print(num_cols)
```

```
Neumeric variables:
['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'W
indSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Press
ure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm', 'month_sin', 'month_cos',
'day_sin', 'day_cos']
```

In [20]:
```
# Missing values in numeric variables

for i in num_cols:
    print(i, data[i].isnull().sum())
```

```
MinTemp 1485
MaxTemp 1261
Rainfall 3261
Evaporation 62790
Sunshine 69835
WindGustSpeed 10263
WindSpeed9am 1767
WindSpeed3pm 3062
Humidity9am 2654
Humidity3pm 4507
Pressure9am 15065
Pressure3pm 15028
Cloud9am 55888
Cloud3pm 59358
Temp9am 1767
Temp3pm 3609
month_sin 0
month_cos 0
day_sin 0
day_cos 0
```

In [21]:
```
# Filling missing values with median of the column in value

for i in num_cols:
    median_value = data[i].median()  # Calculate the median
    data[i] = data[i].fillna(median_value)  # Fill missing values explicitly
```

In [22]:
```
data[num_cols] = data[num_cols].apply(lambda col: col.fillna(col.median()))
```

In [23]:
```
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 30 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  int32         
 24  month          145460 non-null  int32         
 25  month_sin      145460 non-null  float64       
 26  month_cos      145460 non-null  float64       
 27  day            145460 non-null  int32         
 28  day_sin        145460 non-null  float64       
 29  day_cos        145460 non-null  float64       
dtypes: datetime64[ns](1), float64(20), int32(3), object(6)
memory usage: 31.6+ MB
```

```
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning: 
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning: 
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning: 
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning: 
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning: 
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
```

```
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
```

```
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.
```

```
  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\761895308.py:2: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  data[i].fillna(data[i].median(), inplace=True)
```

Data Preprocessing

In [24]:
```
# Apply label encoder to each column with categorical data
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 30 columns):
 #   Column         Non-Null Count   Dtype         
---  ------         --------------   -----         
 0   Date           145460 non-null  datetime64[ns]
 1   Location       145460 non-null  int32         
 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  int32         
 8   WindGustSpeed  145460 non-null  float64       
 9   WindDir9am     145460 non-null  int32         
 10  WindDir3pm     145460 non-null  int32         
 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  int32         
 22  RainTomorrow   145460 non-null  int32         
 23  year           145460 non-null  int32         
 24  month          145460 non-null  int32         
 25  month_sin      145460 non-null  float64       
 26  month_cos      145460 non-null  float64       
 27  day            145460 non-null  int32         
 28  day_sin        145460 non-null  float64       
 29  day_cos        145460 non-null  float64       
dtypes: datetime64[ns](1), float64(20), int32(9)
memory usage: 28.3 MB
```

In [25]:

```
# Prepairing attributes of scale data

features = data.drop(['RainTomorrow', 'Date','day', 'month'], axis=1) # dropping

target = data['RainTomorrow']

#Set up a standard scaler for the features
col_names = list(features.columns)
s_scaler = preprocessing.StandardScaler()
features = s_scaler.fit_transform(features)
features = pd.DataFrame(features, columns=col_names)

features.describe().T
```

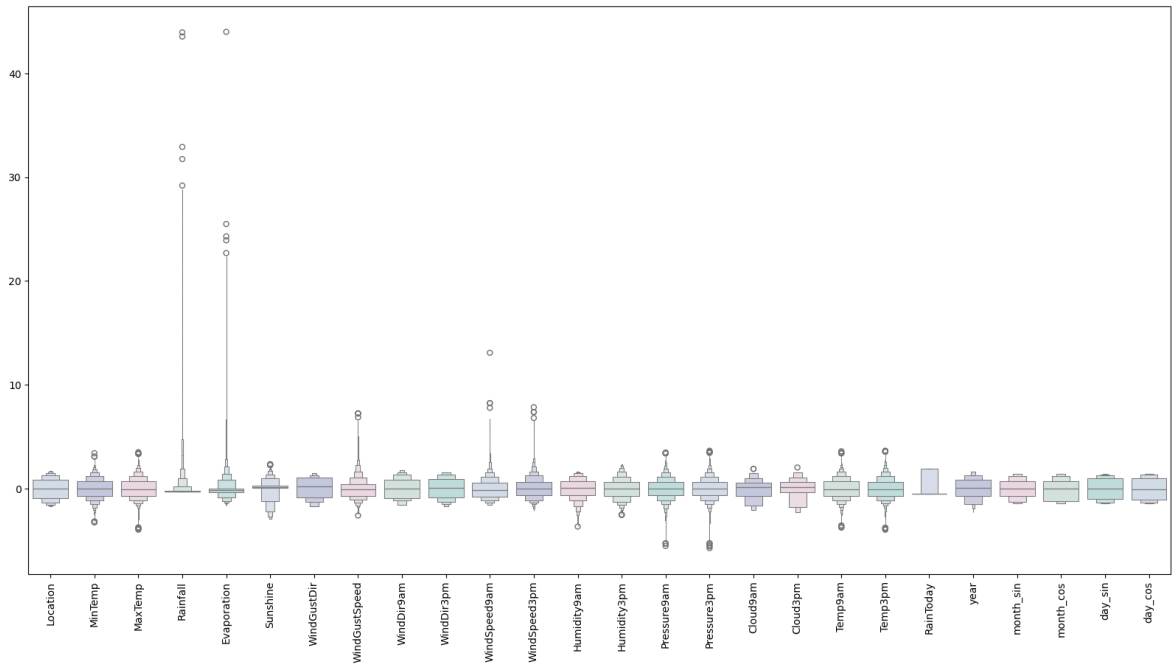
Out[25]:

	count	mean	std	min	25%	50%	7
Location	145460.0	7.815677e-18	1.000003	-1.672228	-0.899139	0.014511	0.857
MinTemp	145460.0	-4.501830e-16	1.000003	-3.250525	-0.705659	-0.030170	0.723
MaxTemp	145460.0	3.001220e-16	1.000003	-3.952405	-0.735852	-0.086898	0.703
Rainfall	145460.0	7.815677e-18	1.000003	-0.275097	-0.275097	-0.275097	-0.203
Evaporation	145460.0	-3.282584e-17	1.000003	-1.629472	-0.371139	-0.119472	0.006
Sunshine	145460.0	-5.424080e-16	1.000003	-2.897217	0.076188	0.148710	0.257
WindGustDir	145460.0	6.252542e-18	1.000003	-1.724209	-0.872075	0.193094	1.045
WindGustSpeed	145460.0	1.824961e-16	1.000003	-2.588407	-0.683048	-0.073333	0.460
WindDir9am	145460.0	7.190423e-17	1.000003	-1.550000	-0.885669	0.000105	0.885
WindDir3pm	145460.0	8.284618e-17	1.000003	-1.718521	-0.837098	0.044324	0.925
WindSpeed9am	145460.0	5.627287e-17	1.000003	-1.583291	-0.793380	-0.116314	0.560
WindSpeed3pm	145460.0	6.565169e-17	1.000003	-2.141841	-0.650449	0.037886	0.611
Humidity9am	145460.0	2.250915e-16	1.000003	-3.654212	-0.631189	0.058273	0.747
Humidity3pm	145460.0	-8.440931e-17	1.000003	-2.518329	-0.710918	0.021816	0.656
Pressure9am	145460.0	-4.314254e-16	1.000003	-5.520544	-0.616005	-0.006653	0.617
Pressure3pm	145460.0	5.027043e-15	1.000003	-5.724832	-0.622769	-0.007520	0.622
Cloud9am	145460.0	-1.016038e-16	1.000003	-2.042425	-0.727490	0.149133	0.587
Cloud3pm	145460.0	7.346736e-17	1.000003	-2.235619	-0.336969	0.137693	0.612
Temp9am	145460.0	7.503050e-17	1.000003	-3.750358	-0.726764	-0.044517	0.699
Temp3pm	145460.0	-6.877796e-17	1.000003	-3.951301	-0.725322	-0.083046	0.661
RainToday	145460.0	-8.988029e-18	1.000003	-0.529795	-0.529795	-0.529795	-0.529

	count	mean	std	min	25%	50%	7
year	145460.0	2.080221e-14	1.000003	-2.273637	-0.697391	0.090732	0.878
month_sin	145460.0	5.861758e-19	1.000003	-1.434333	-0.725379	-0.016425	0.692
month_cos	145460.0	-2.745257e-17	1.000003	-1.388032	-1.198979	0.023080	0.728
day_sin	145460.0	1.075877e-17	1.000003	-1.403140	-1.019170	-0.003198	1.012
day_cos	145460.0	-1.353700e-17	1.000003	-1.392587	-1.055520	-0.044639	1.011

In [26]:
```
#Detecting outliers
#looking at the scaled features
colours = ["#D0DBEE", "#C2C4E2", "#EED4E5", "#D1E6DC", "#BDE2E2"]
plt.figure(figsize=(20,10))
sns.boxenplot(data = features,palette = colours)
plt.xticks(rotation=90)
plt.show()
```

```
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\2961525613.py:5: UserWarning:
The palette list has fewer values (5) than needed (26) and will cycle, which may
produce an uninterpretable plot.
  sns.boxenplot(data = features,palette = colours)
```



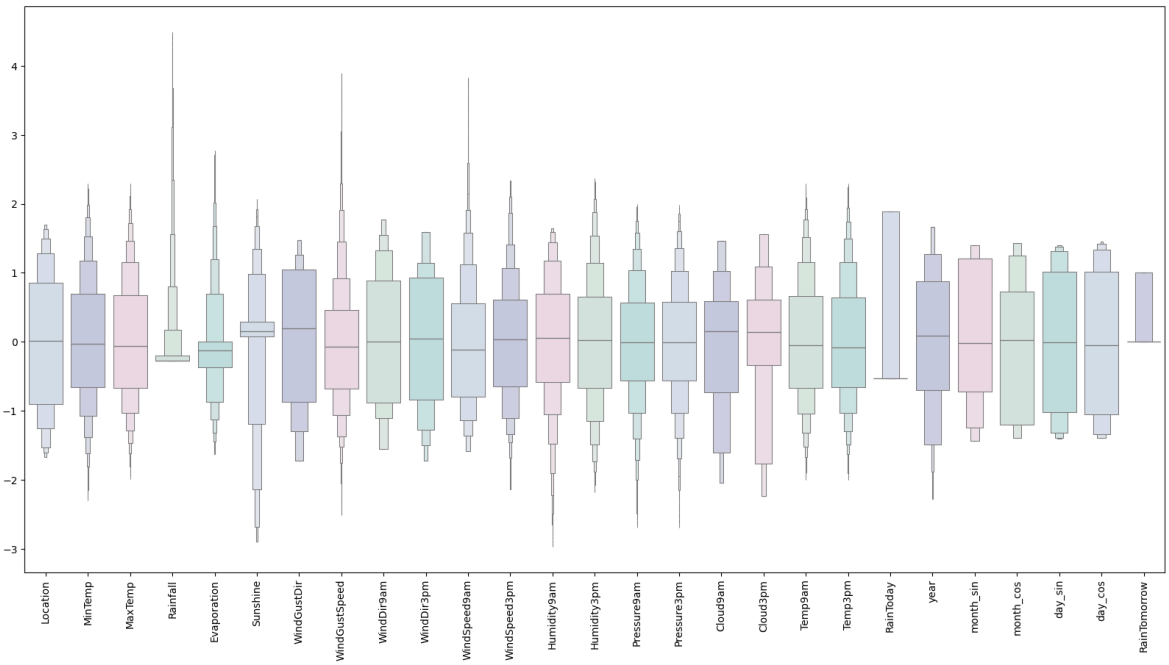
In [27]:
```
#full data for
features["RainTomorrow"] = target

#Dropping with outlier

features = features[(features["MinTemp"]<2.3)&(features["MinTemp"]>-2.3)]
features = features[(features["MaxTemp"]<2.3)&(features["MaxTemp"]>-2)]
features = features[(features["Rainfall"]<4.5)]
features = features[(features["Evaporation"]<2.8)]
```

```
features = features[(features["Sunshine"]<2.1)]
features = features[(features["WindGustSpeed"]<4)&(features["WindGustSpeed"]>-4)
features = features[(features["WindSpeed9am"]<4)]
features = features[(features["WindSpeed3pm"]<2.5)]
features = features[(features["Humidity9am"]>-3)]
features = features[(features["Humidity3pm"]>-2.2)]
features = features[(features["Pressure9am"]< 2)&(features["Pressure9am"]>-2.7)]
features = features[(features["Pressure3pm"]< 2)&(features["Pressure3pm"]>-2.7)]
features = features[(features["Cloud9am"]<1.8)]
features = features[(features["Cloud3pm"]<2)]
features = features[(features["Temp9am"]<2.3)&(features["Temp9am"]>-2)]
features = features[(features["Temp3pm"]<2.3)&(features["Temp3pm"]>-2)]


features.shape
```

Out[27]: (127536, 27)

In [28]:
```
#looking at the scaled features without outliers

plt.figure(figsize=(20,10))
sns.boxenplot(data = features,palette = colours)
plt.xticks(rotation=90)
plt.show()
```

```
C:\Users\chitt\AppData\Local\Temp\ipykernel_23064\1688296607.py:4: UserWarning:
The palette list has fewer values (5) than needed (27) and will cycle, which may
produce an uninterpretable plot.
  sns.boxenplot(data = features,palette = colours)
```



MODEL BUILDING

In [29]:
```
X = features.drop(["RainTomorrow"], axis=1)
y = features["RainTomorrow"]

# Splitting test and training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, randc
```

```
X.shape
```

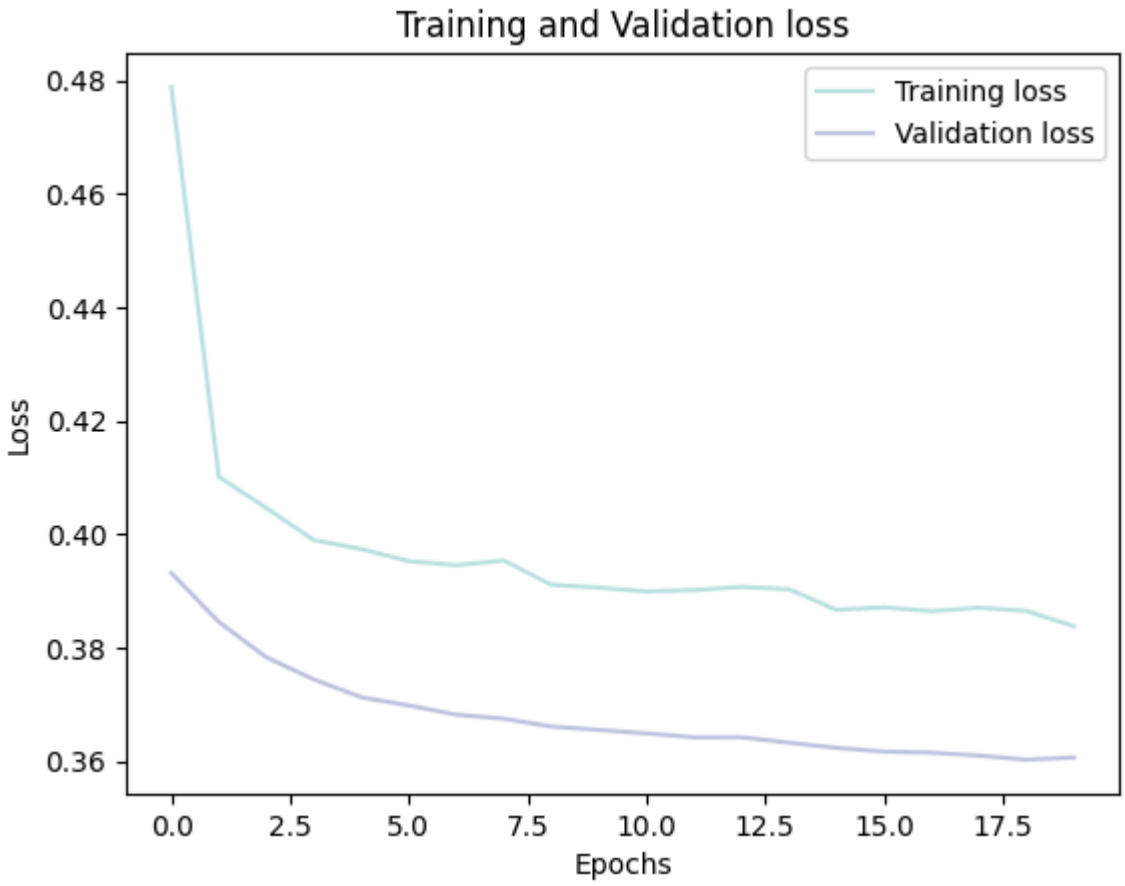
Out[29]: (127536, 26)

In [30]:
```
#Early stopping
early_stopping = callbacks.EarlyStopping(
    min_delta=0.001, # minimium amount of change to count as an improvement
    patience=20, # how many epochs to wait before stopping
    restore_best_weights=True,
)

# Initialising the NN
model = Sequential()

# layers

model.add(Dense(units = 32, kernel_initializer = 'uniform', activation = 'relu',
model.add(Dense(units = 32, kernel_initializer = 'uniform', activation = 'relu')
model.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu')
model.add(Dropout(0.25))
model.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid
# Compiling the ANN
opt = Adam(learning_rate=0.00009)
model.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['accurac

# Train the ANN
history = model.fit(X_train, y_train, batch_size = 32, epochs = 20, callbacks=[e
```

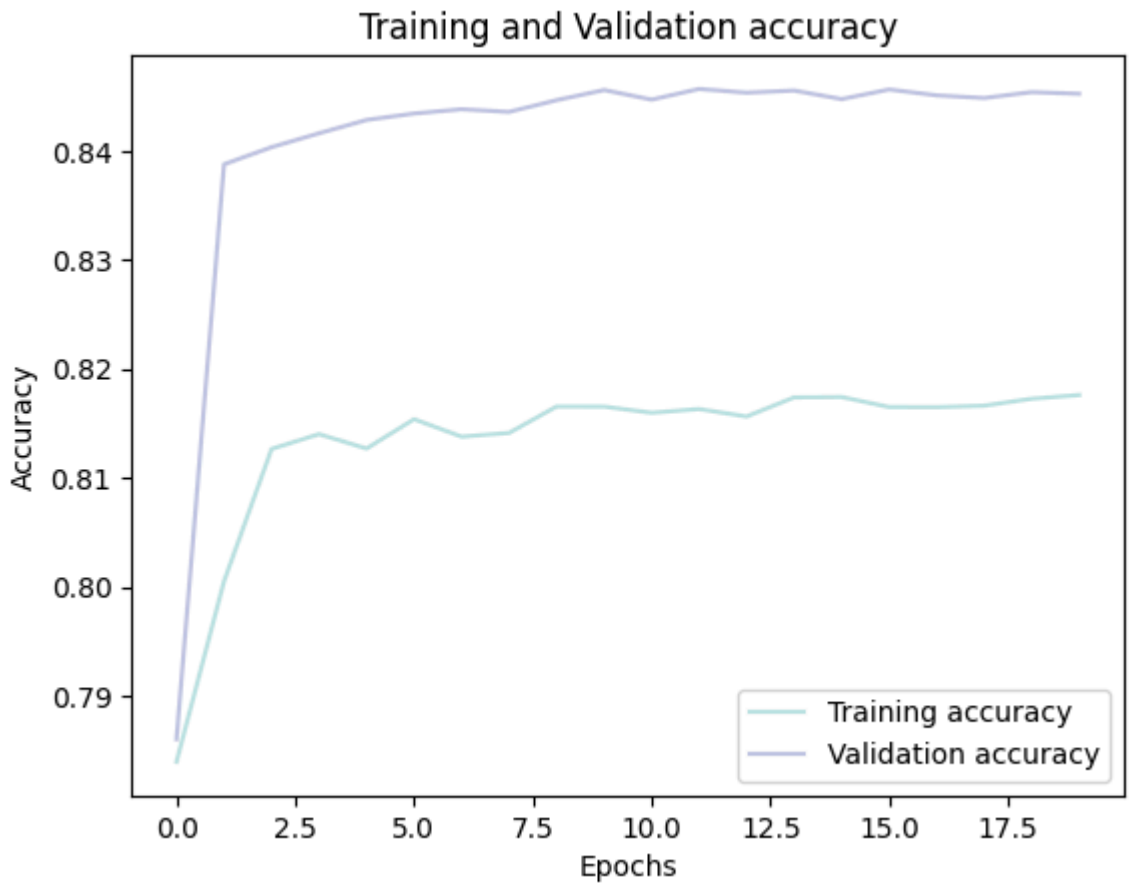
```
Epoch 1/20
C:\Users\chitt\anaconda3\envs\tensorflow_env\lib\site-packages\keras\src\layers\c
ore\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument t
o a layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.7823 - loss: 0.5629 - va
l_accuracy: 0.7860 - val_loss: 0.3932
Epoch 2/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.7911 - loss: 0.4134 - va
l_accuracy: 0.8388 - val_loss: 0.3845
Epoch 3/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8122 - loss: 0.4065 - va
l_accuracy: 0.8403 - val_loss: 0.3783
Epoch 4/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8146 - loss: 0.3989 - va
l_accuracy: 0.8416 - val_loss: 0.3744
Epoch 5/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8129 - loss: 0.3979 - va
l_accuracy: 0.8428 - val_loss: 0.3712
Epoch 6/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8155 - loss: 0.3956 - va
l_accuracy: 0.8434 - val_loss: 0.3698
Epoch 7/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8139 - loss: 0.3941 - va
l_accuracy: 0.8438 - val_loss: 0.3682
Epoch 8/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - accuracy: 0.8151 - loss: 0.3946 - va
l_accuracy: 0.8436 - val_loss: 0.3675
Epoch 9/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8156 - loss: 0.3927 - va
l_accuracy: 0.8447 - val_loss: 0.3661
Epoch 10/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8180 - loss: 0.3884 - va
l_accuracy: 0.8456 - val_loss: 0.3655
Epoch 11/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8176 - loss: 0.3851 - va
l_accuracy: 0.8447 - val_loss: 0.3649
Epoch 12/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.8196 - loss: 0.3883 - va
l_accuracy: 0.8457 - val_loss: 0.3642
Epoch 13/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8158 - loss: 0.3905 - va
l_accuracy: 0.8453 - val_loss: 0.3642
Epoch 14/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8183 - loss: 0.3890 - va
l_accuracy: 0.8455 - val_loss: 0.3632
Epoch 15/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8168 - loss: 0.3862 - va
l_accuracy: 0.8448 - val_loss: 0.3623
Epoch 16/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8160 - loss: 0.3862 - va
l_accuracy: 0.8456 - val_loss: 0.3617
Epoch 17/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8173 - loss: 0.3840 - va
l_accuracy: 0.8451 - val_loss: 0.3615
Epoch 18/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8149 - loss: 0.3870 - va
l_accuracy: 0.8448 - val_loss: 0.3610
Epoch 19/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8180 - loss: 0.3841 - va
l_accuracy: 0.8454 - val_loss: 0.3602
Epoch 20/20
2551/2551 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8154 - loss: 0.3868 - va
l_accuracy: 0.8452 - val_loss: 0.3606
```

In [31]:
```
history_df = pd.DataFrame(history.history)

plt.plot(history_df.loc[:, ['loss']], "#BDE2E2", label='Training loss')
plt.plot(history_df.loc[:, ['val_loss']],"#C2C4E2", label='Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc="best")

plt.show()
```



In [32]:
```
history_df = pd.DataFrame(history.history)

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

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



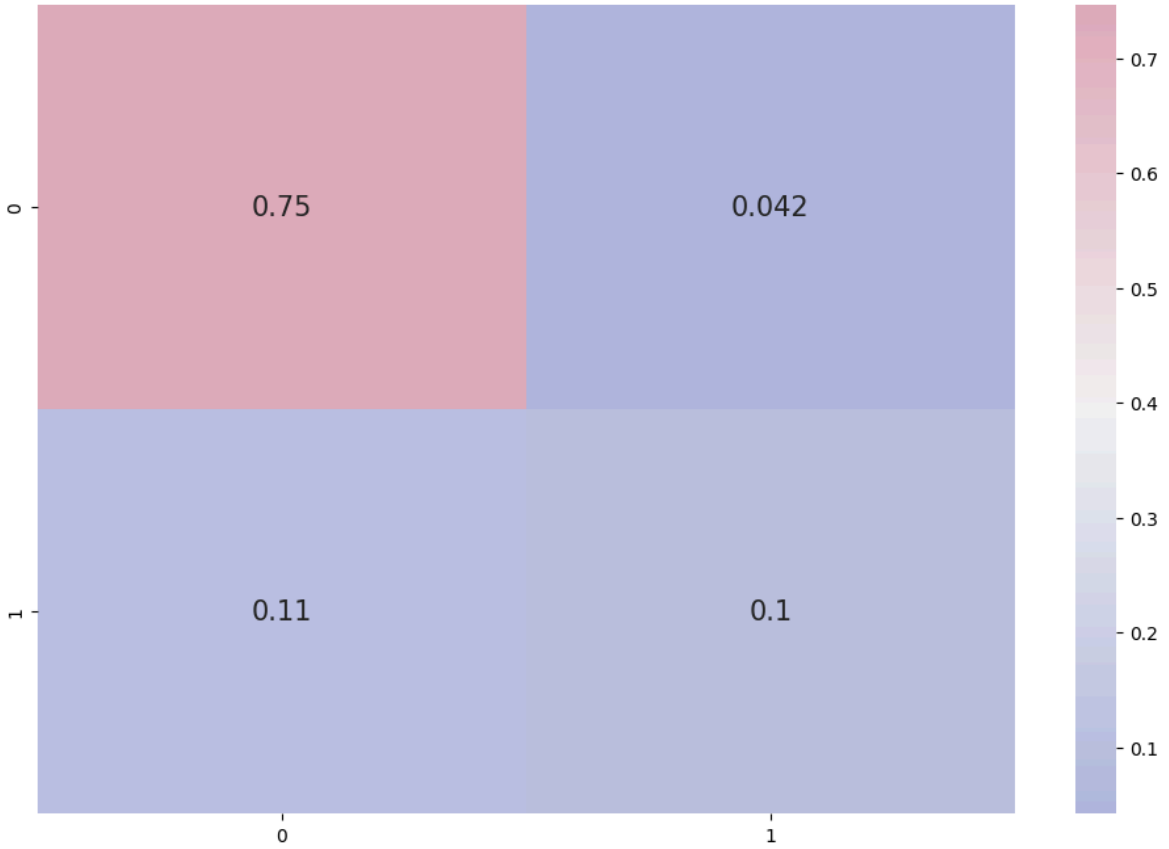
Conclusion

In [33]:
```
# Predicting the test set results
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)
```

```
798/798 ━━━━━━━━━━━━━━━━━━━━ 1s 707us/step
```

In [34]:
```
# confusion matrix
cmap1 = sns.diverging_palette(260,-10,s=50, l=75, n=5, as_cmap=True)
plt.subplots(figsize=(12,8))
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix/np.sum(cf_matrix), cmap = cmap1, annot = True, annot_kws =
```

Out[34]: <Axes: >



In [35]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.95      0.91     20110
           1       0.71      0.49      0.58      5398

    accuracy                           0.85     25508
   macro avg       0.79      0.72      0.74     25508
weighted avg       0.84      0.85      0.84     25508
```

In [ ]: