In [1]:
```python
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
import pandas as pd
import numpy as np
np.random.seed(0)
```

```
WARNING:tensorflow:From C:\Users\chand\anaconda3\Lib\site-packages\keras\s
rc\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is depr
ecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instea
d.
```

In [2]:
```python
data = pd.read_csv("weatherAUS.csv")
data.head()
```

Out[2]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	Wind(
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 [3]:
```
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 [4]:
```
#Parsing datetime
#exploring the length of date objects
lengths = data["Date"].str.len()
lengths.value_counts()
```

Out[4]:
```
Date
10    145460
Name: count, dtype: int64
```

In [5]:

```
#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 (

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()
```

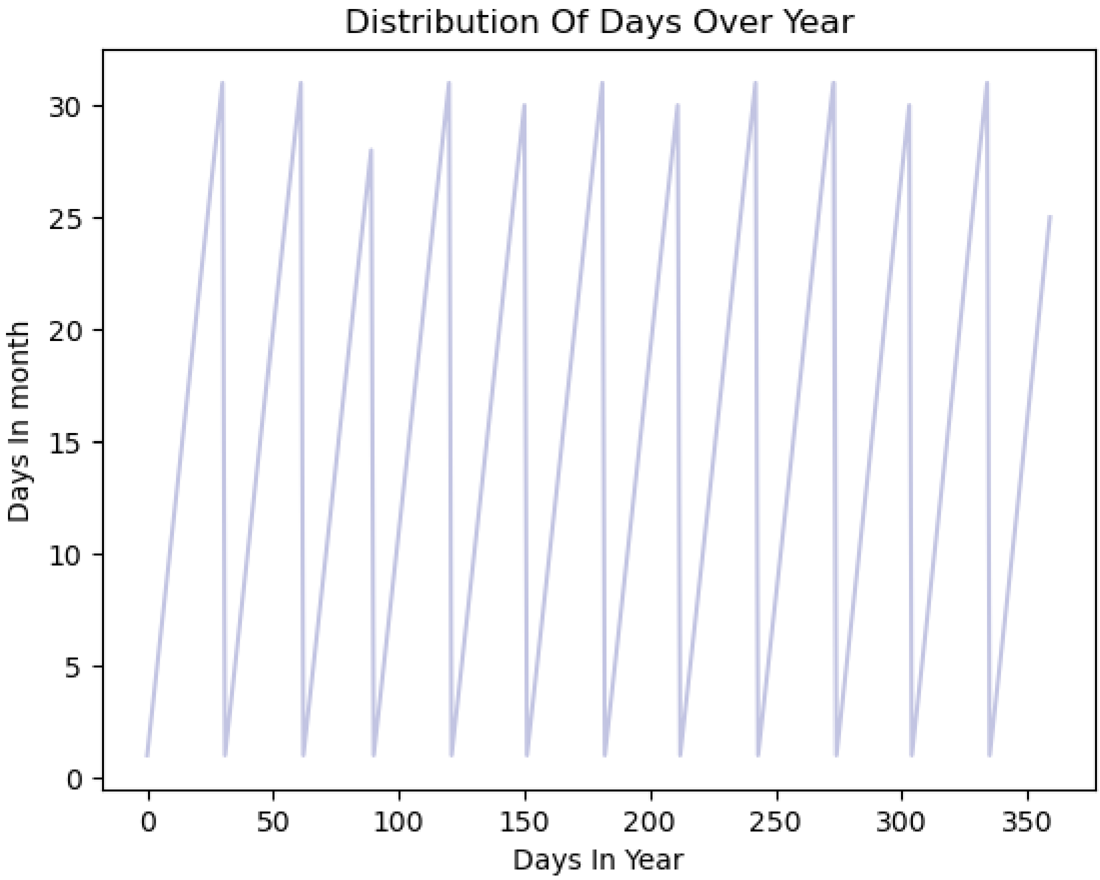
Out[5]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	Wind(
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

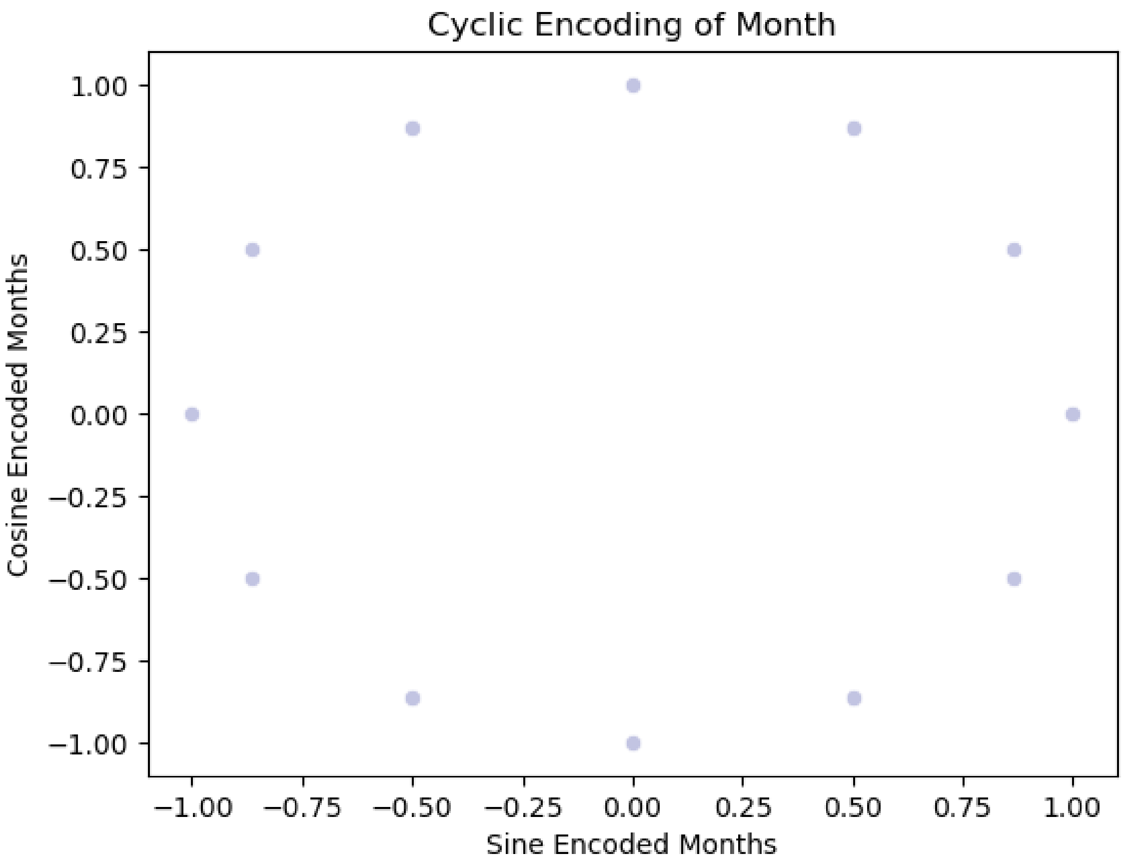
In [6]:
```
# roughly a year's span section
section = data[:360]
tm = section["day"].plot(color="#C2C4E2")
tm.set_title("Distribution Of Days Over Year")
tm.set_ylabel("Days In month")
tm.set_xlabel("Days In Year")
```

Out[6]:
```
Text(0.5, 0, 'Days In Year')
```



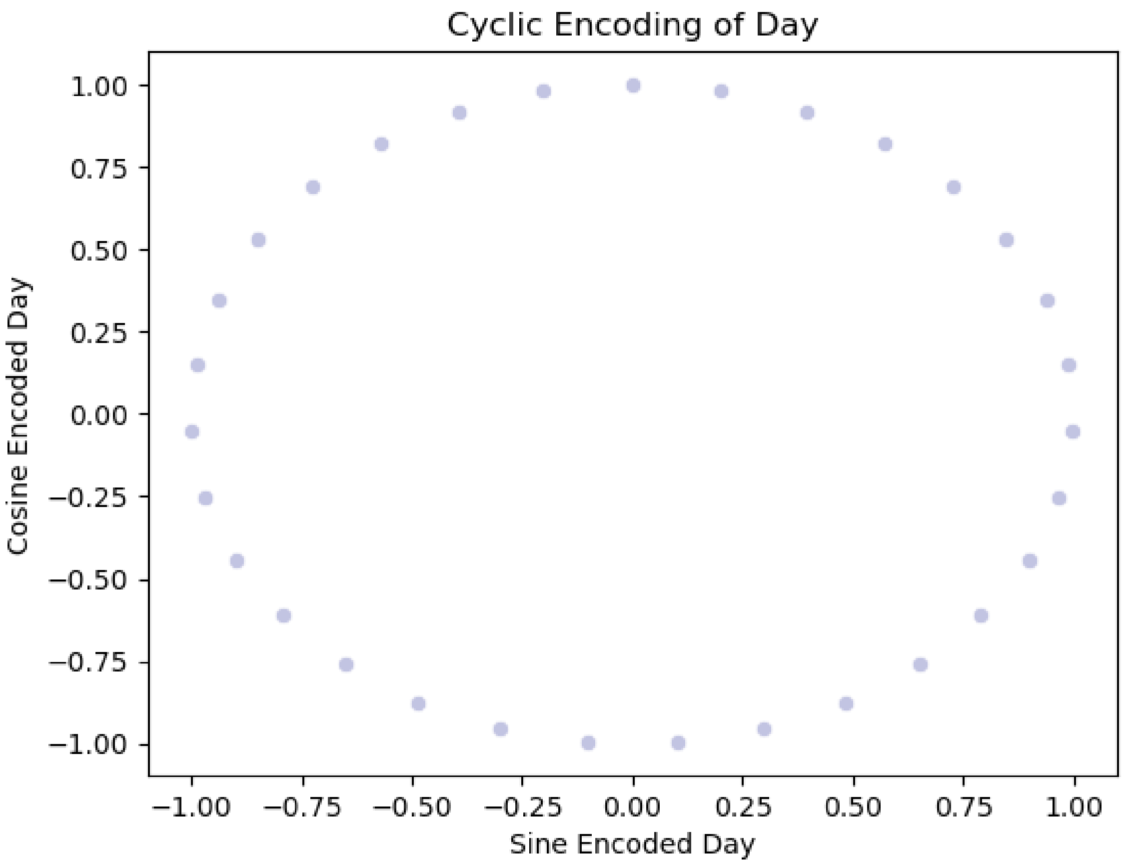
In [7]:
```
cyclic_month = sns.scatterplot(x="month_sin",y="month_cos",data=data, color
cyclic_month.set_title("Cyclic Encoding of Month")
cyclic_month.set_ylabel("Cosine Encoded Months")
cyclic_month.set_xlabel("Sine Encoded Months")
```

Out[7]:
```
Text(0.5, 0, 'Sine Encoded Months')
```



In [8]:
```
cyclic_day = sns.scatterplot(x='day_sin',y='day_cos',data=data, color="#C2C
cyclic_day.set_title("Cyclic Encoding of Day")
cyclic_day.set_ylabel("Cosine Encoded Day")
cyclic_day.set_xlabel("Sine Encoded Day")
```

Out[8]:
```
Text(0.5, 0, 'Sine Encoded Day')
```



In [9]:
```
# 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', 'Rain
Tomorrow']
```

In [10]:
```
# 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 [11]:
```
# Filling missing values with mode of the column in value

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

In [12]:
```
# 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', 'WindGustSpe
ed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressu
re9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm', 'mont
h_sin', 'month_cos', 'day_sin', 'day_cos']
```

In [13]:
```
# 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 [14]:

```
# Filling missing values with median of the column in value

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
```

In [15]:
```python
# 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 [16]:
```
# Prepairing attributes of scale data

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

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
```

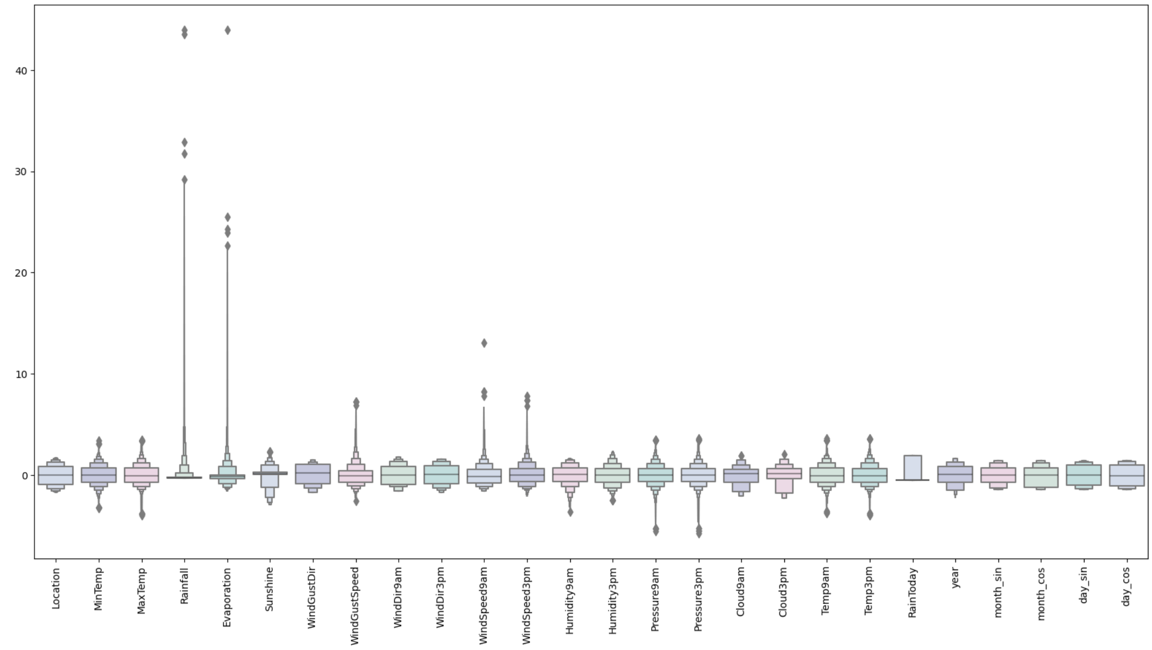
Out[16]:

	count	mean	std	min	25%	50%	75%	
Location	145460.0	7.815677e-18	1.000003	-1.672228	-0.899139	0.014511	0.857881	
MinTemp	145460.0	-4.501830e-16	1.000003	-3.250525	-0.705659	-0.030170	0.723865	
MaxTemp	145460.0	3.001220e-16	1.000003	-3.952405	-0.735852	-0.086898	0.703133	
Rainfall	145460.0	7.815677e-18	1.000003	-0.275097	-0.275097	-0.275097	-0.203581	4
Evaporation	145460.0	-3.282584e-17	1.000003	-1.629472	-0.371139	-0.119472	0.006361	4
Sunshine	145460.0	-5.424080e-16	1.000003	-2.897217	0.076188	0.148710	0.257494	
WindGustDir	145460.0	6.252542e-18	1.000003	-1.724209	-0.872075	0.193094	1.045228	
WindGustSpeed	145460.0	1.824961e-16	1.000003	-2.588407	-0.683048	-0.073333	0.460168	
WindDir9am	145460.0	7.190423e-17	1.000003	-1.550000	-0.885669	0.000105	0.885879	
WindDir3pm	145460.0	8.284618e-17	1.000003	-1.718521	-0.837098	0.044324	0.925747	
WindSpeed9am	145460.0	5.627287e-17	1.000003	-1.583291	-0.793380	-0.116314	0.560752	1
WindSpeed3pm	145460.0	6.565169e-17	1.000003	-2.141841	-0.650449	0.037886	0.611499	
Humidity9am	145460.0	2.250915e-16	1.000003	-3.654212	-0.631189	0.058273	0.747734	
Humidity3pm	145460.0	-8.440931e-17	1.000003	-2.518329	-0.710918	0.021816	0.656852	
Pressure9am	145460.0	-4.314254e-16	1.000003	-5.520544	-0.616005	-0.006653	0.617561	
Pressure3pm	145460.0	5.027043e-15	1.000003	-5.724832	-0.622769	-0.007520	0.622735	
Cloud9am	145460.0	-1.016038e-16	1.000003	-2.042425	-0.727490	0.149133	0.587445	
Cloud3pm	145460.0	7.346736e-17	1.000003	-2.235619	-0.336969	0.137693	0.612356	
Temp9am	145460.0	7.503050e-17	1.000003	-3.750358	-0.726764	-0.044517	0.699753	
Temp3pm	145460.0	-6.877796e-17	1.000003	-3.951301	-0.725322	-0.083046	0.661411	
RainToday	145460.0	-8.988029e-18	1.000003	-0.529795	-0.529795	-0.529795	-0.529795	
year	145460.0	2.080221e-14	1.000003	-2.273637	-0.697391	0.090732	0.878855	
month_sin	145460.0	5.861758e-19	1.000003	-1.434333	-0.725379	-0.016425	0.692529	
month_cos	145460.0	-2.745257e-17	1.000003	-1.388032	-1.198979	0.023080	0.728636	
day_sin	145460.0	1.075877e-17	1.000003	-1.403140	-1.019170	-0.003198	1.012774	

	count	mean	std	min	25%	50%	75%
day_cos	145460.0	-1.353700e-17	1.000003	-1.392587	-1.055520	-0.044639	1.011221

In [17]:

```
#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()
```



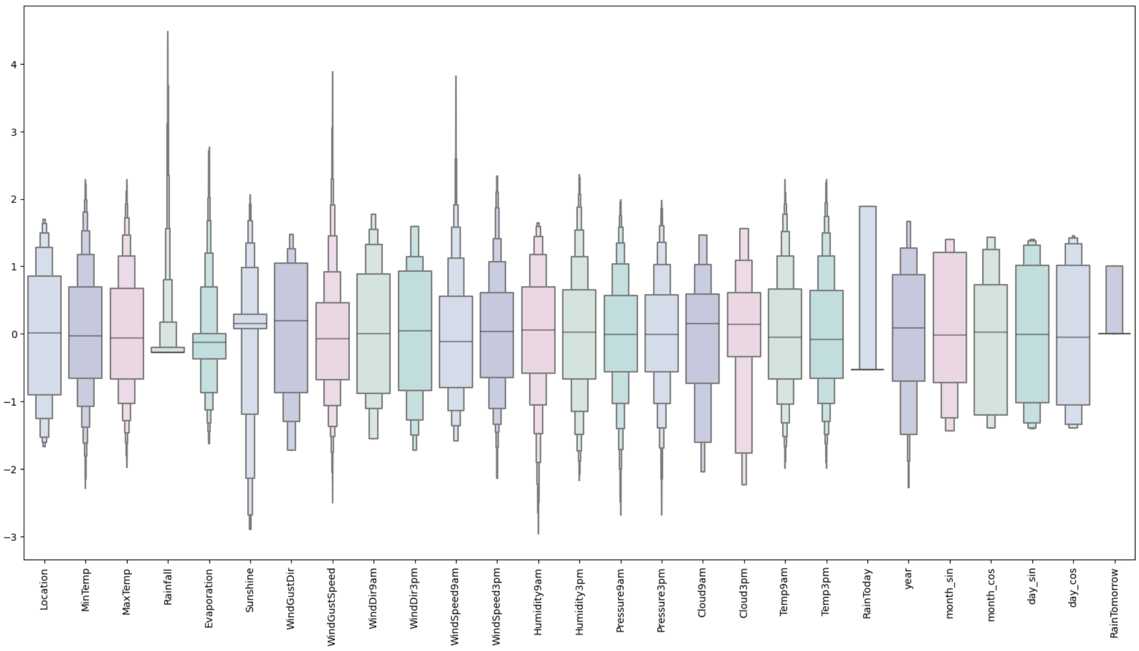
In [18]:

```
#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"]
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"]>-
features = features[(features["Pressure3pm"]< 2)&(features["Pressure3pm"]>-
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[18]: (127536, 27)

In [19]:

```
#looking at the scaled features without outliers

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



In [20]:

```
#Model building
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, 

X.shape
```

Out[20]:

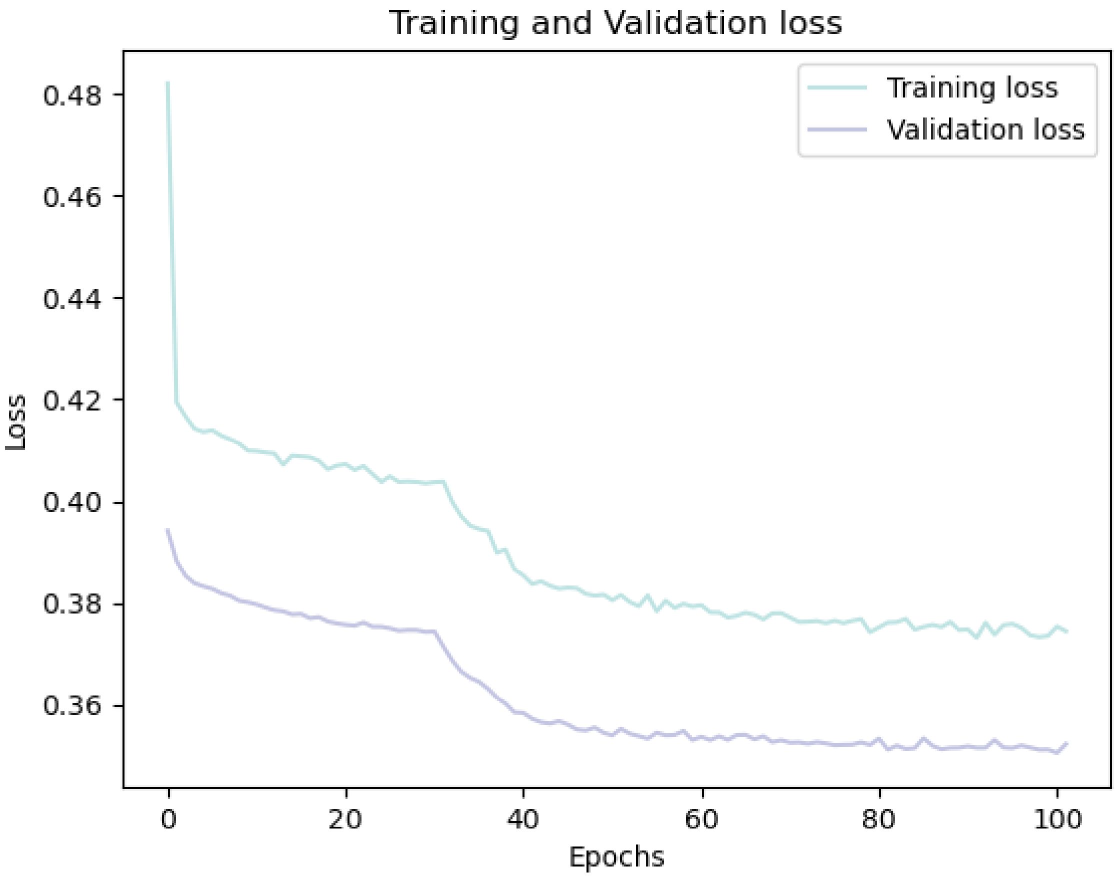
```
(127536, 26)
```

In [22]:

```
#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 = 16, kernel_initializer = 'uniform', activation = 'r
model.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'r
model.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 're
model.add(Dropout(0.25))
model.add(Dense(units = 4, kernel_initializer = 'uniform', activation = 're
model.add(Dropout(0.5))
model.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'si

# Compiling the ANN
opt = Adam(learning_rate=0.00009)
model.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['ac

# Train the ANN
history = model.fit(X_train, y_train, batch_size = 16, epochs = 50, callbac
```

```
Epoch 1/50
5102/5102 [==============================] - 18s 3ms/step - loss: 0.4931
- accuracy: 0.7842 - val_loss: 0.3982 - val_accuracy: 0.7860
Epoch 2/50
5102/5102 [==============================] - 15s 3ms/step - loss: 0.4451
- accuracy: 0.7842 - val_loss: 0.3951 - val_accuracy: 0.7860
Epoch 3/50
5102/5102 [==============================] - 15s 3ms/step - loss: 0.4394
- accuracy: 0.7842 - val_loss: 0.3945 - val_accuracy: 0.7860
Epoch 4/50
5102/5102 [==============================] - 16s 3ms/step - loss: 0.4376
- accuracy: 0.7842 - val_loss: 0.3942 - val_accuracy: 0.7860
Epoch 5/50
5102/5102 [==============================] - 16s 3ms/step - loss: 0.4378
- accuracy: 0.7842 - val_loss: 0.3948 - val_accuracy: 0.7860
Epoch 6/50
5102/5102 [==============================] - 17s 3ms/step - loss: 0.4380
- accuracy: 0.7842 - val_loss: 0.3929 - val_accuracy: 0.7860
Epoch 7/50
```

In [26]:
```
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 [27]:
```
history_df = pd.DataFrame(history.history)

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

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

In [ ]: