Rain Prediction

Import Libraries

In [2]:
```
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, classification_report, accuracy_score, f1_
from keras import callbacks

np.random.seed(23)
```

Load Data

In [3]:
```
data = pd.read_csv("../input/weather-dataset-rattle-package/weatherAUS.csv")
data.head()
```

Out[3]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity9am	Hu
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	71.0	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	44.0	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	38.0	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	45.0	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	82.0	

5 rows × 23 columns

About the data:

The dataset contains approximately 10 years of daily weather observations from various locations across Australia. The observations were drawn from numerous weather stations.

In this project, I will use this data to predict whether or not it will rain the next day. There are 23 attributes, including the target variable "RainTomorrow," indicating whether it will rain the following day or not.

In [4]:
```
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
```

The dataset contains missing values.

The dataset comprises both numeric and categorical values

Data Cleaning and visualization

Generate a count plot of the target column.

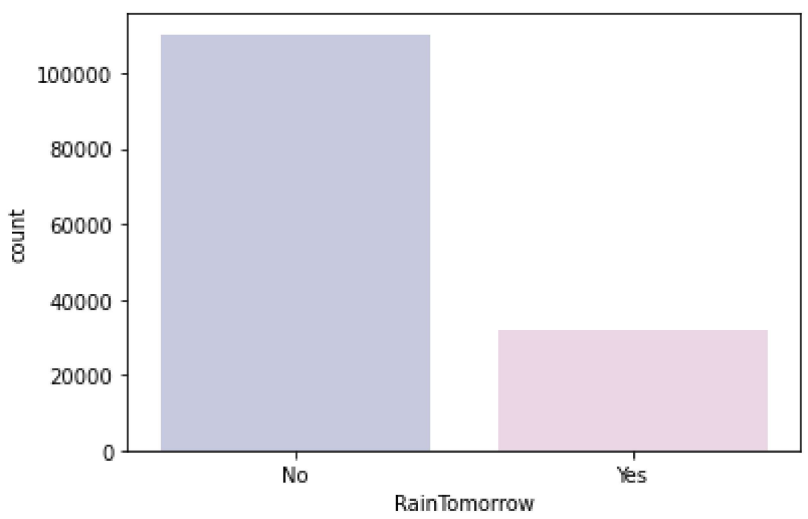
Calculate the correlation among numeric attributes.

Parse dates into the datetime format.

Encode days and months as continuous cyclic features.

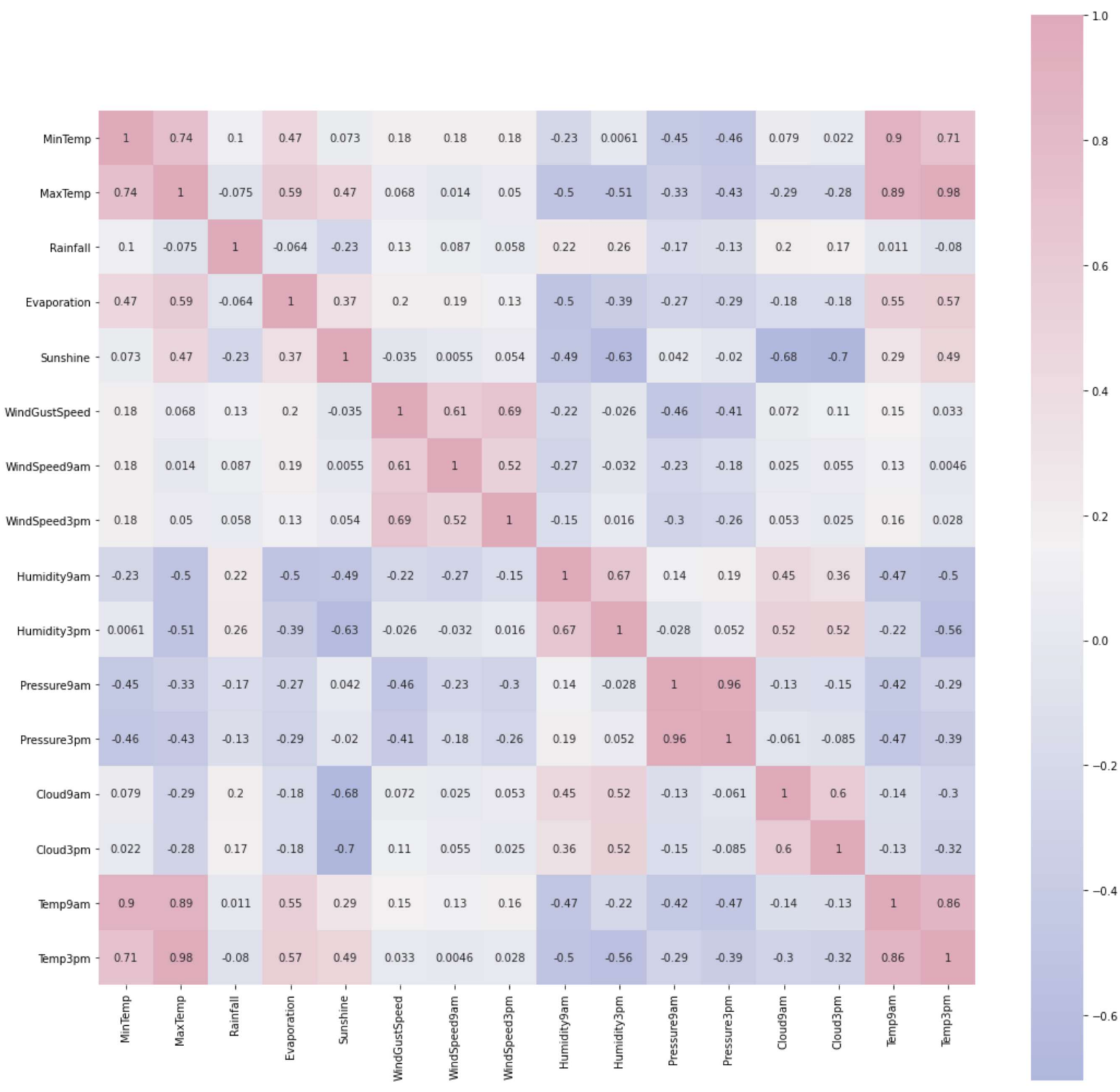
In [5]:
```
# Evaluate the target and find out if  data is imbalanced or not
cols= ["#C2C4E2","#EED4E5"]
sns.countplot(x= data["RainTomorrow"], palette= cols)
```

Out[5]:
```
<AxesSubplot:xlabel='RainTomorrow', ylabel='count'>
```



In [6]:
```
# Correlation amongst numeric attributes
corrmat = data.corr()
cmap = sns.diverging_palette(260,-10,s=50, l=75, n=6, as_cmap=True)
plt.subplots(figsize=(18,18))
sns.heatmap(corrmat,cmap= cmap,annot=True, square=True)
```

Out[6]:
```
<AxesSubplot:>
```



Parse Dates into datetime.

My goal is to construct an artificial neural network (ANN). I will appropriately encode dates by representing months and days as cyclic continuous features. Since date and time exhibit inherent cyclic patterns, I divide them into periodic subsections, specifically years, months, and days. For each of these subsections, I generate two new features by calculating the sine and cosine transforms of the corresponding time units.

In [7]:
```
#Parsing datetime
#exploring the length of date objects
lengths = data["Date"].str.len()
lengths.value_counts()
```

Out[7]:
```
10    145460
Name: Date, dtype: int64
```

In [8]:
```
#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 for encoding datetime into cyclic parameters.
#As I intend to utilize this data in a neural network, I favor representing months and days as cyclic continuous feature
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()
```

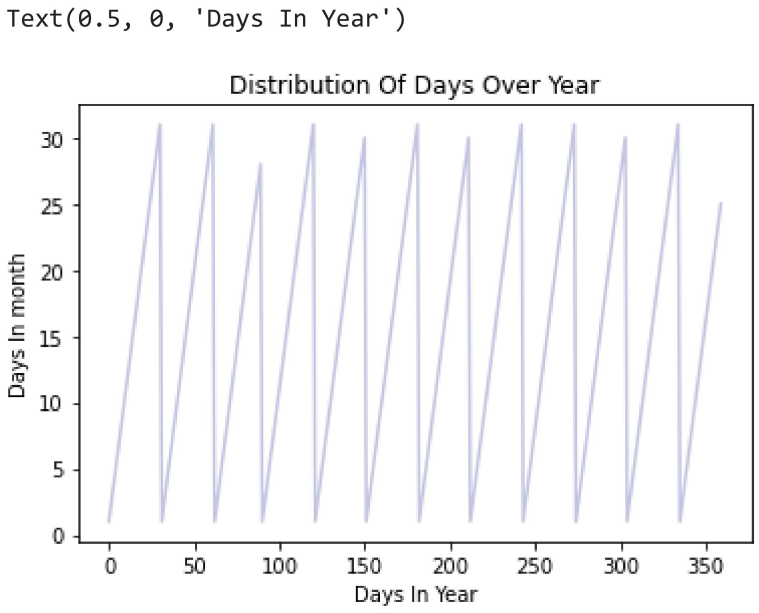
Out[8]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Temp3pm	RainTo
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	21.8	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	24.3	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	23.2	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	26.5	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	29.7	

5 rows × 30 columns

In [9]:
```
# A time period of approximately one year.
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")
```

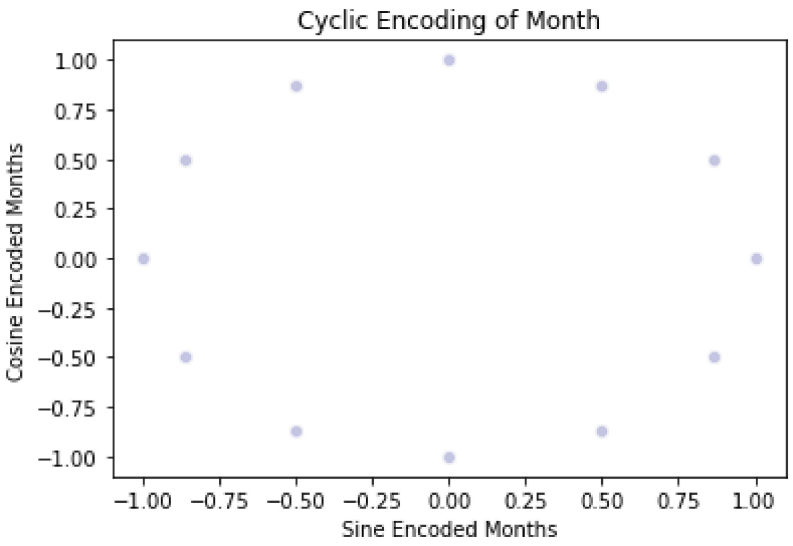
Out[9]:
```
Text(0.5, 0, 'Days In Year')
```



As expected, the 'year' attribute in the data repeats, but it doesn't exhibit a continuous cyclical nature. By breaking down months and days into sine and cosine combinations, a continuous cyclic feature is established. These features can be utilized as inputs for an Artificial Neural Network (ANN).

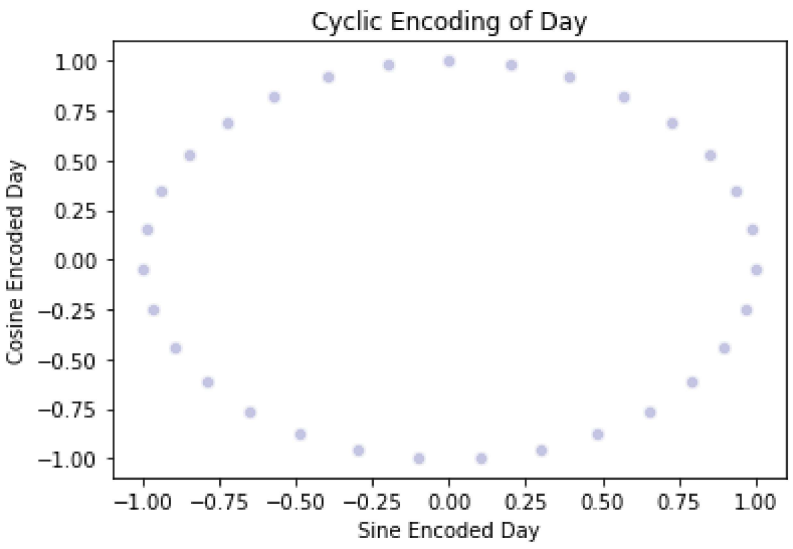
In [10]:
```
cyclic_month = sns.scatterplot(x="month_sin",y="month_cos",data=data, color="#C2C4E2")
cyclic_month.set_title("Cyclic Encoding of Month")
cyclic_month.set_ylabel("Cosine Encoded Months")
cyclic_month.set_xlabel("Sine Encoded Months")
```

Out[10]:
```
Text(0.5, 0, 'Sine Encoded Months')
```



In [11]:
```
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[11]:
```
Text(0.5, 0, 'Sine Encoded Day')
```



Subsequently, I'll address missing values in categorical and numeric attributes separately.

Categorical variables

Imputing missing values using the mode of the column.

In [12]:
```
# 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', 'RainTomorrow']
```

In [13]:
```
# 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 [ ]:
```
# Imputing missing values with the mode of the column.

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

Numerical variables

- Filling missing values with median of the column value

In [15]:
```
# 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', 'WindSpeed9am', 'WindSpeed3pm', 'Humidit
y9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm', 'month_sin', 'month_c
os', 'day_sin', 'day_cos']
```

In [16]:
```
# Missing values in numeric variables
```

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

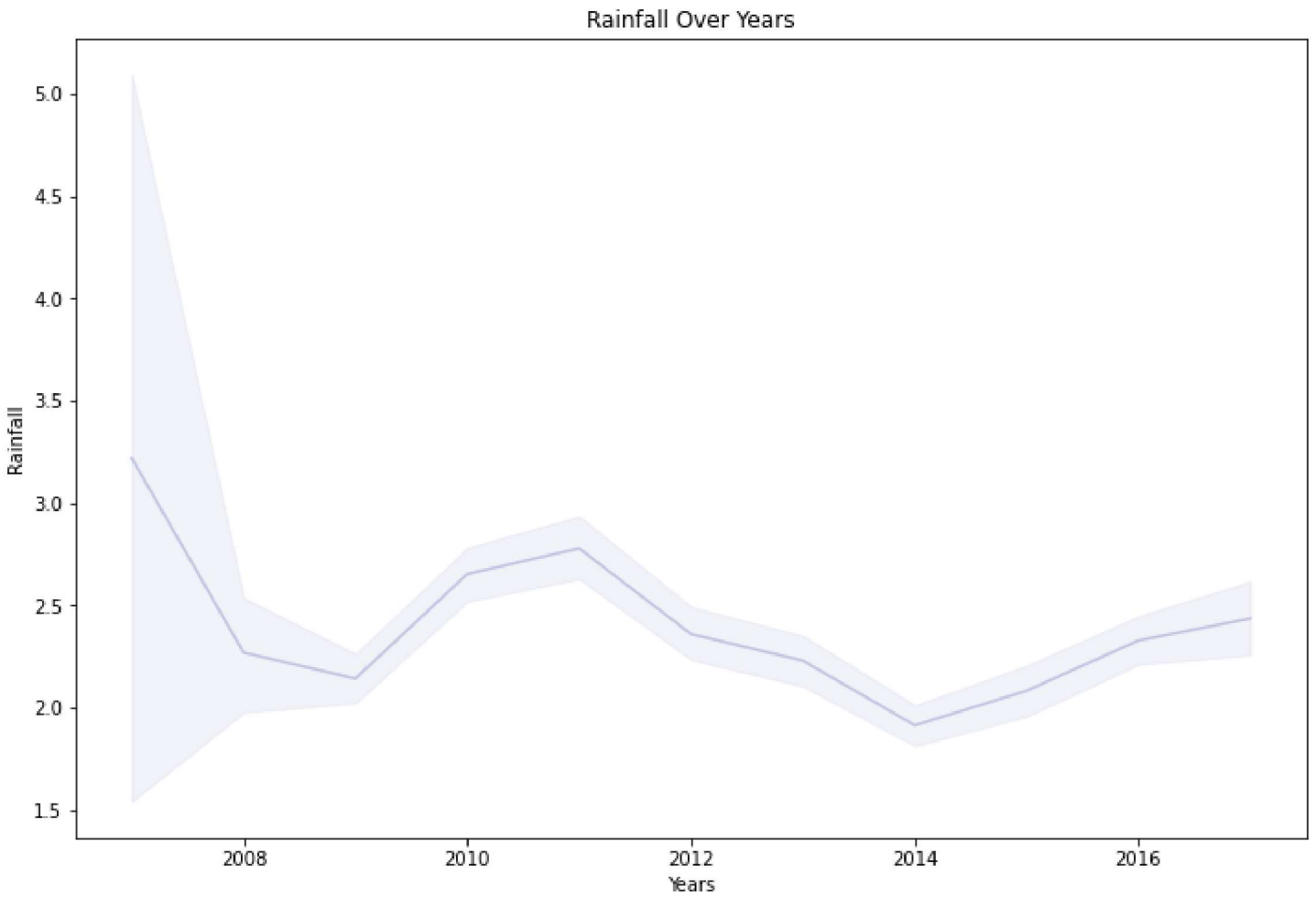
```
MinTemp 1485
MaxTemp 1261
Rainfall 3261
Evaporation 62790
Sunshine 69835
WindGustSpeed 10263
WindSpeed9am 1767
WindSpeed3pm 3062
Humidity9am 2654
Humidity3pm 4507
Pressure9am 15065
Pressure3pm 15028
Cloud9am 55888
Cloud3pm 59358
Temp9am 1767
Temp3pm 3609
month_sin 0
month_cos 0
day_sin 0
day_cos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

In [17]:
```
# Replacing missing values with the median of the column.

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  int64
 24  month          145460 non-null  int64
 25  month_sin      145460 non-null  float64
 26  month_cos      145460 non-null  float64
 27  day            145460 non-null  int64
 28  day_sin        145460 non-null  float64
 29  day_cos        145460 non-null  float64
dtypes: datetime64[ns](1), float64(20), int64(3), object(6)
memory usage: 33.3+ MB
```

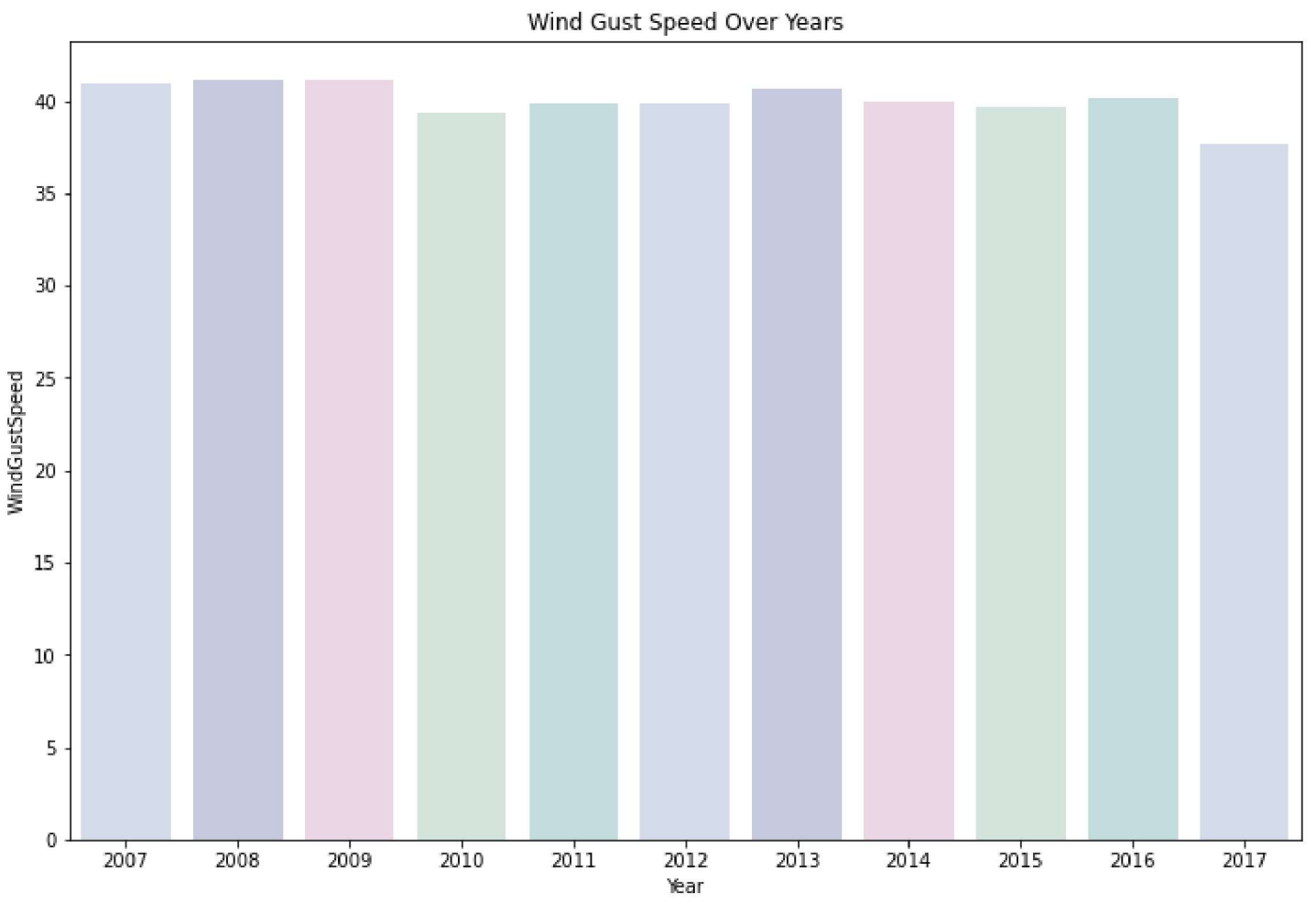
In [18]:
```
#plotting a lineplot rainfall over years
plt.figure(figsize=(12,8))
Time_series=sns.lineplot(x=data['Date'].dt.year,y="Rainfall",data=data,color="#C2C4E2")
Time_series.set_title("Rainfall Over Years")
Time_series.set_ylabel("Rainfall")
Time_series.set_xlabel("Years")
```

Out[18]:
```
Text(0.5, 0, 'Years')
```



In [19]:
```
#Evauating Wind gust speed over years
colours = ["#D0DBEE", "#C2C4E2", "#EED4E5", "#D1E6DC", "#BDE2E2"]
plt.figure(figsize=(12,8))
Days_of_week=sns.barplot(x=data['Date'].dt.year,y="WindGustSpeed",data=data, ci =None,palette = colours)
Days_of_week.set_title("Wind Gust Speed Over Years")
Days_of_week.set_ylabel("WindGustSpeed")
Days_of_week.set_xlabel("Year")
```

Out[19]:
```
Text(0.5, 0, 'Year')
```



Data Preprocessing

Data Preprocessing Steps:

Encode columns with categorical data using label encoding

Scale the features

Detect outliers

Remove outliers based on data analysis

Label encoding the catagorical varable

In [20]:
```
# Apply a label encoder to each column containing 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  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  month_sin      145460 non-null  float64
 26  month_cos      145460 non-null  float64
 27  day            145460 non-null  int64
 28  day_sin        145460 non-null  float64
 29  day_cos        145460 non-null  float64
dtypes: datetime64[ns](1), float64(20), int64(9)
memory usage: 33.3 MB
```

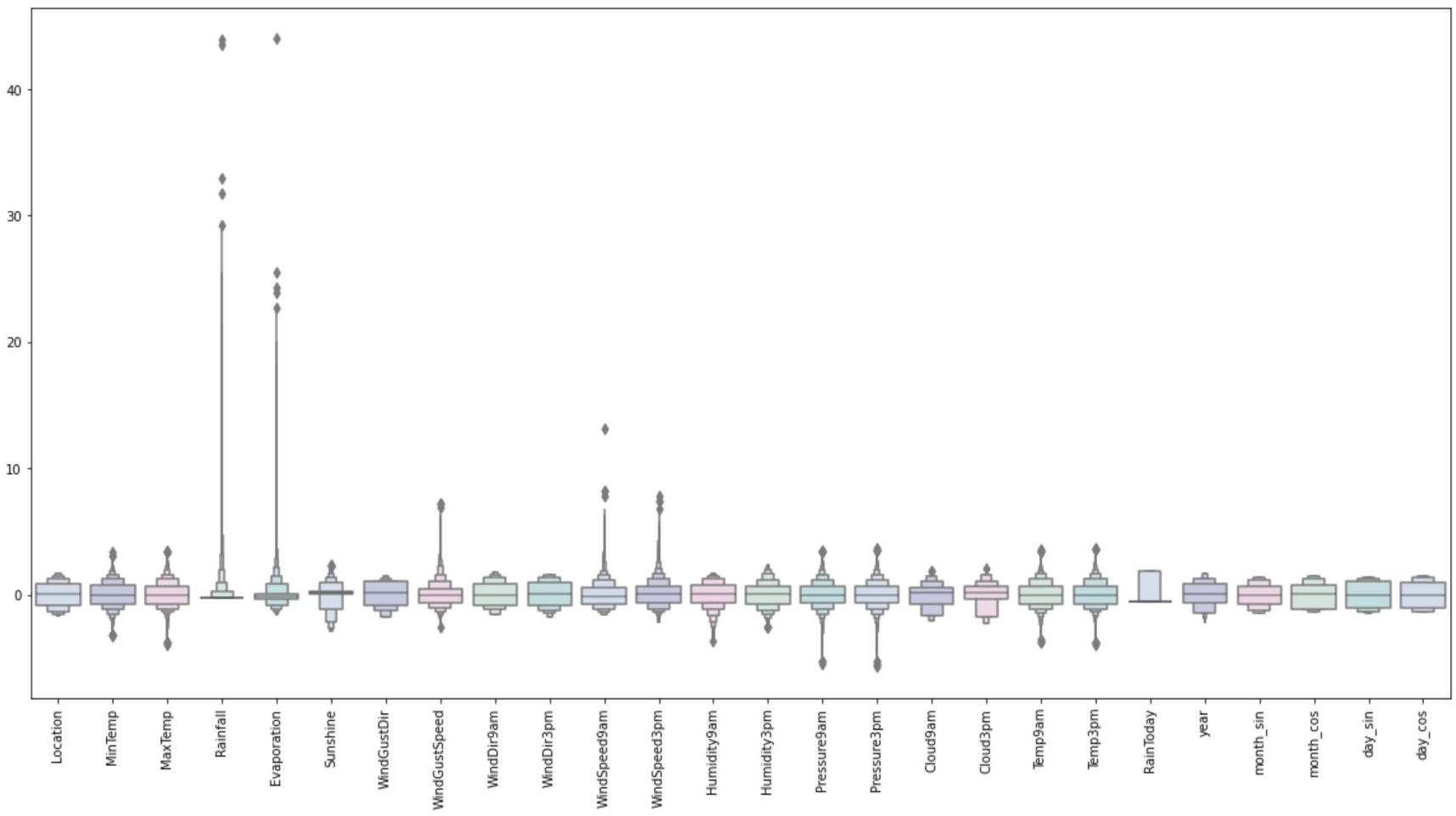
In [21]:

```python
# Prepairing attributes of scale data

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

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[21]:

	count	mean	std	min	25%	50%	75%	max
Location	145460.0	-5.633017e-14	1.000003	-1.672228	-0.899139	0.014511	0.857881	1.701250
MinTemp	145460.0	-4.243854e-15	1.000003	-3.250525	-0.705659	-0.030170	0.723865	3.410112
MaxTemp	145460.0	6.513740e-16	1.000003	-3.952405	-0.735852	-0.086898	0.703133	3.510563
Rainfall	145460.0	9.152711e-15	1.000003	-0.275097	-0.275097	-0.275097	-0.203581	43.945571
Evaporation	145460.0	1.352327e-14	1.000003	-1.629472	-0.371139	-0.119472	0.006361	43.985108
Sunshine	145460.0	-4.338304e-15	1.000003	-2.897217	0.076188	0.148710	0.257494	2.360634
WindGustDir	145460.0	1.864381e-14	1.000003	-1.724209	-0.872075	0.193094	1.045228	1.471296
WindGustSpeed	145460.0	-1.167921e-14	1.000003	-2.588407	-0.683048	-0.073333	0.460168	7.243246
WindDir9am	145460.0	-7.433272e-15	1.000003	-1.550000	-0.885669	0.000105	0.885879	1.771653
WindDir3pm	145460.0	1.791486e-15	1.000003	-1.718521	-0.837098	0.044324	0.925747	1.586813
WindSpeed9am	145460.0	-3.422029e-14	1.000003	-1.583291	-0.793380	-0.116314	0.560752	13.086472
WindSpeed3pm	145460.0	1.618238e-14	1.000003	-2.141841	-0.650449	0.037886	0.611499	7.839016
Humidity9am	145460.0	-4.803490e-15	1.000003	-3.654212	-0.631189	0.058273	0.747734	1.649338
Humidity3pm	145460.0	-6.041889e-15	1.000003	-2.518329	-0.710918	0.021816	0.656852	2.366565
Pressure9am	145460.0	2.313398e-14	1.000003	-5.520544	-0.616005	-0.006653	0.617561	3.471111
Pressure3pm	145460.0	4.709575e-15	1.000003	-5.724832	-0.622769	-0.007520	0.622735	3.653960
Cloud9am	145460.0	-2.525820e-14	1.000003	-2.042425	-0.727490	0.149133	0.587445	1.902380
Cloud3pm	145460.0	4.796901e-15	1.000003	-2.235619	-0.336969	0.137693	0.612356	2.036343
Temp9am	145460.0	-3.332880e-15	1.000003	-3.750358	-0.726764	-0.044517	0.699753	3.599302
Temp3pm	145460.0	-2.901899e-15	1.000003	-3.951301	-0.725322	-0.083046	0.661411	3.653834
RainToday	145460.0	1.263303e-14	1.000003	-0.529795	-0.529795	-0.529795	-0.529795	1.887521
year	145460.0	1.663818e-14	1.000003	-2.273637	-0.697391	0.090732	0.878855	1.666978
month_sin	145460.0	1.653870e-15	1.000003	-1.434333	-0.725379	-0.016425	0.692529	1.401483
month_cos	145460.0	4.043483e-16	1.000003	-1.388032	-1.198979	0.023080	0.728636	1.434192
day_sin	145460.0	-1.982159e-17	1.000003	-1.403140	-1.019170	-0.003198	1.012774	1.396744
day_cos	145460.0	-9.540621e-19	1.000003	-1.392587	-1.055520	-0.044639	1.011221	1.455246

In [22]:
```
#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()
```

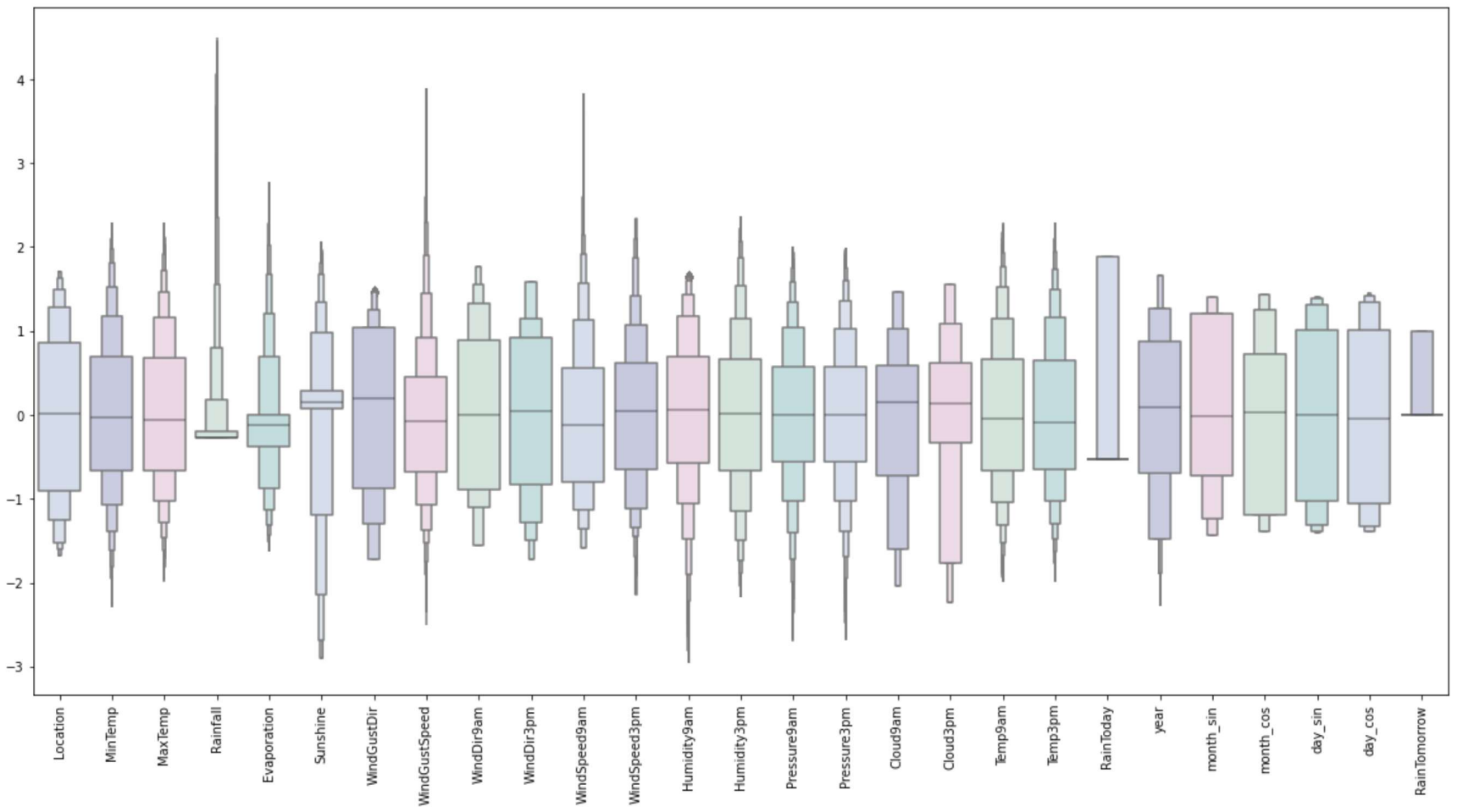


In [23]:
```
#full data for
features["RainTomorrow"] = target

#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[23]:
```
(127536, 27)
```

In [24]:
```
#looking at the scaled features without outliers

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



Model Building

Steps Involved in Model Building:

Assigning X and y to represent attributes and target labels, respectively.

Splitting the dataset into training and testing sets.

Initializing the neural network.

Defining the architecture by adding layers to the network.

Compiling the neural network.

Training the neural network.

In [25]:
```
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, random_state = 42)

X.shape
```

Out[25]:
```
(127536, 26)
```

In [26]:
```
#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 ANN
model = Sequential()
```

```
# layers

model.add(Dense(units = 32, kernel_initializer = 'uniform', activation = 'relu', input_dim = 26))
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 = ['accuracy'])

# Train the ANN
history = model.fit(X_train, y_train, batch_size = 32, epochs = 150, callbacks=[early_stopping], validation_split=0.2)
```

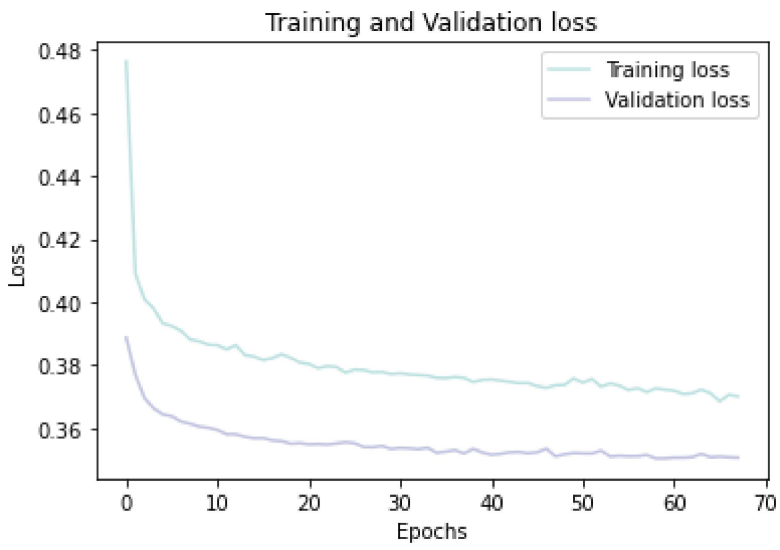
```
Epoch 1/150
2551/2551 [==============================] - 8s 3ms/step - loss: 0.5613 - accuracy: 0.7848 - val_loss: 0.3887 - val_acc
uracy: 0.8245
Epoch 2/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.4118 - accuracy: 0.8119 - val_loss: 0.3769 - val_acc
uracy: 0.8381
Epoch 3/150
2551/2551 [==============================] - 6s 3ms/step - loss: 0.3979 - accuracy: 0.8283 - val_loss: 0.3695 - val_acc
uracy: 0.8406
Epoch 4/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3978 - accuracy: 0.8293 - val_loss: 0.3663 - val_acc
uracy: 0.8426
Epoch 5/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3945 - accuracy: 0.8290 - val_loss: 0.3643 - val_acc
uracy: 0.8432
Epoch 6/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3938 - accuracy: 0.8285 - val_loss: 0.3637 - val_acc
uracy: 0.8447
Epoch 7/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3897 - accuracy: 0.8333 - val_loss: 0.3621 - val_acc
uracy: 0.8447
Epoch 8/150
2551/2551 [==============================] - 6s 3ms/step - loss: 0.3891 - accuracy: 0.8283 - val_loss: 0.3614 - val_acc
uracy: 0.8448
Epoch 9/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3853 - accuracy: 0.8334 - val_loss: 0.3605 - val_acc
uracy: 0.8452
Epoch 10/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3879 - accuracy: 0.8310 - val_loss: 0.3601 - val_acc
uracy: 0.8449
Epoch 11/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3838 - accuracy: 0.8331 - val_loss: 0.3594 - val_acc
uracy: 0.8446
Epoch 12/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3824 - accuracy: 0.8346 - val_loss: 0.3580 - val_acc
uracy: 0.8454
Epoch 13/150
2551/2551 [==============================] - 7s 3ms/step - loss: 0.3881 - accuracy: 0.8298 - val_loss: 0.3580 - val_acc
uracy: 0.8444
Epoch 14/150
2551/2551 [==============================] - 6s 3ms/step - loss: 0.3878 - accuracy: 0.8297 - val_loss: 0.3572 - val_acc
uracy: 0.8447
Epoch 15/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3823 - accuracy: 0.8324 - val_loss: 0.3568 - val_acc
uracy: 0.8448
Epoch 16/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3807 - accuracy: 0.8332 - val_loss: 0.3568 - val_acc
uracy: 0.8433
Epoch 17/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3846 - accuracy: 0.8310 - val_loss: 0.3561 - val_acc
uracy: 0.8443
Epoch 18/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3798 - accuracy: 0.8331 - val_loss: 0.3559 - val_acc
uracy: 0.8433
Epoch 19/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3853 - accuracy: 0.8298 - val_loss: 0.3550 - val_acc
uracy: 0.8447
Epoch 20/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3802 - accuracy: 0.8316 - val_loss: 0.3552 - val_acc
uracy: 0.8442
Epoch 21/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3798 - accuracy: 0.8325 - val_loss: 0.3548 - val_acc
uracy: 0.8438
Epoch 22/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3817 - accuracy: 0.8311 - val_loss: 0.3549 - val_acc
uracy: 0.8434
Epoch 23/150
2551/2551 [==============================] - 6s 3ms/step - loss: 0.3790 - accuracy: 0.8330 - val_loss: 0.3548 - val_acc
uracy: 0.8439
Epoch 24/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3809 - accuracy: 0.8328 - val_loss: 0.3551 - val_acc
uracy: 0.8438
Epoch 25/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3777 - accuracy: 0.8304 - val_loss: 0.3555 - val_acc
uracy: 0.8433
Epoch 26/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3784 - accuracy: 0.8310 - val_loss: 0.3552 - val_acc
uracy: 0.8435
Epoch 27/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3751 - accuracy: 0.8337 - val_loss: 0.3540 - val_acc
uracy: 0.8445
Epoch 28/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3744 - accuracy: 0.8340 - val_loss: 0.3539 - val_acc
uracy: 0.8437
Epoch 29/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3769 - accuracy: 0.8340 - val_loss: 0.3543 - val_acc
uracy: 0.8434
Epoch 30/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3764 - accuracy: 0.8325 - val_loss: 0.3534 - val_acc
uracy: 0.8436
Epoch 31/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3783 - accuracy: 0.8313 - val_loss: 0.3537 - val_acc
uracy: 0.8439
Epoch 32/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3760 - accuracy: 0.8333 - val_loss: 0.3535 - val_acc
```

```
uracy: 0.8437
Epoch 33/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3732 - accuracy: 0.8349 - val_loss: 0.3533 - val_acc
uracy: 0.8433
Epoch 34/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3769 - accuracy: 0.8318 - val_loss: 0.3537 - val_acc
uracy: 0.8430
Epoch 35/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3754 - accuracy: 0.8324 - val_loss: 0.3521 - val_acc
uracy: 0.8448
Epoch 36/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3787 - accuracy: 0.8298 - val_loss: 0.3525 - val_acc
uracy: 0.8446
Epoch 37/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3747 - accuracy: 0.8350 - val_loss: 0.3530 - val_acc
uracy: 0.8449
Epoch 38/150
2551/2551 [==============================] - 6s 3ms/step - loss: 0.3787 - accuracy: 0.8309 - val_loss: 0.3519 - val_acc
uracy: 0.8448
Epoch 39/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3757 - accuracy: 0.8313 - val_loss: 0.3534 - val_acc
uracy: 0.8432
Epoch 40/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3780 - accuracy: 0.8325 - val_loss: 0.3522 - val_acc
uracy: 0.8448
Epoch 41/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3736 - accuracy: 0.8337 - val_loss: 0.3515 - val_acc
uracy: 0.8467
Epoch 42/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3749 - accuracy: 0.8336 - val_loss: 0.3518 - val_acc
uracy: 0.8456
Epoch 43/150
2551/2551 [==============================] - 6s 3ms/step - loss: 0.3696 - accuracy: 0.8331 - val_loss: 0.3523 - val_acc
uracy: 0.8455
Epoch 44/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3753 - accuracy: 0.8312 - val_loss: 0.3524 - val_acc
uracy: 0.8450
Epoch 45/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3723 - accuracy: 0.8344 - val_loss: 0.3520 - val_acc
uracy: 0.8453
Epoch 46/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3784 - accuracy: 0.8312 - val_loss: 0.3523 - val_acc
uracy: 0.8448
Epoch 47/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3693 - accuracy: 0.8351 - val_loss: 0.3534 - val_acc
uracy: 0.8440
Epoch 48/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3720 - accuracy: 0.8343 - val_loss: 0.3510 - val_acc
uracy: 0.8458
Epoch 49/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3753 - accuracy: 0.8321 - val_loss: 0.3517 - val_acc
uracy: 0.8453
Epoch 50/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3769 - accuracy: 0.8313 - val_loss: 0.3521 - val_acc
uracy: 0.8456
Epoch 51/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3715 - accuracy: 0.8356 - val_loss: 0.3520 - val_acc
uracy: 0.8455
Epoch 52/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3721 - accuracy: 0.8345 - val_loss: 0.3519 - val_acc
uracy: 0.8455
Epoch 53/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3723 - accuracy: 0.8347 - val_loss: 0.3528 - val_acc
uracy: 0.8450
Epoch 54/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3764 - accuracy: 0.8311 - val_loss: 0.3510 - val_acc
uracy: 0.8465
Epoch 55/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3711 - accuracy: 0.8340 - val_loss: 0.3512 - val_acc
uracy: 0.8469
Epoch 56/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3723 - accuracy: 0.8328 - val_loss: 0.3510 - val_acc
uracy: 0.8463
Epoch 57/150
2551/2551 [==============================] - 6s 3ms/step - loss: 0.3759 - accuracy: 0.8309 - val_loss: 0.3511 - val_acc
uracy: 0.8459
Epoch 58/150
2551/2551 [==============================] - 7s 3ms/step - loss: 0.3715 - accuracy: 0.8345 - val_loss: 0.3514 - val_acc
uracy: 0.8457
Epoch 59/150
2551/2551 [==============================] - 6s 3ms/step - loss: 0.3708 - accuracy: 0.8351 - val_loss: 0.3503 - val_acc
uracy: 0.8470
Epoch 60/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3715 - accuracy: 0.8338 - val_loss: 0.3504 - val_acc
uracy: 0.8471
Epoch 61/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3736 - accuracy: 0.8339 - val_loss: 0.3506 - val_acc
uracy: 0.8469
Epoch 62/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3676 - accuracy: 0.8352 - val_loss: 0.3506 - val_acc
uracy: 0.8461
Epoch 63/150
2551/2551 [==============================] - 7s 3ms/step - loss: 0.3686 - accuracy: 0.8351 - val_loss: 0.3508 - val_acc
uracy: 0.8462
Epoch 64/150
```

```
2551/2551 [==============================] - 6s 3ms/step - loss: 0.3712 - accuracy: 0.8341 - val_loss: 0.3518 - val_acc
uracy: 0.8447
Epoch 65/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3720 - accuracy: 0.8318 - val_loss: 0.3507 - val_acc
uracy: 0.8460
Epoch 66/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3639 - accuracy: 0.8378 - val_loss: 0.3510 - val_acc
uracy: 0.8463
Epoch 67/150
2551/2551 [==============================] - 6s 2ms/step - loss: 0.3721 - accuracy: 0.8325 - val_loss: 0.3507 - val_acc
uracy: 0.8459
Epoch 68/150
2551/2551 [==============================] - 6s 3ms/step - loss: 0.3659 - accuracy: 0.8375 - val_loss: 0.3506 - val_acc
uracy: 0.8469
```

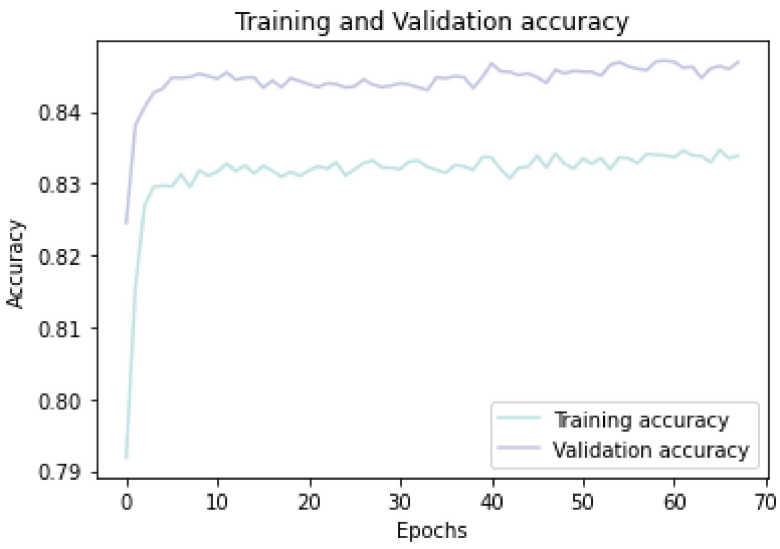
Creating a plot to display the training and validation loss across epochs.

In [27]:
```
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()
```



Generating a plot to visualize the training and validation accuracy throughout the epochs.

In [28]:
```
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 accuracy')

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



Conclusion

Testing on the test set

Evaluating the confusion matrix

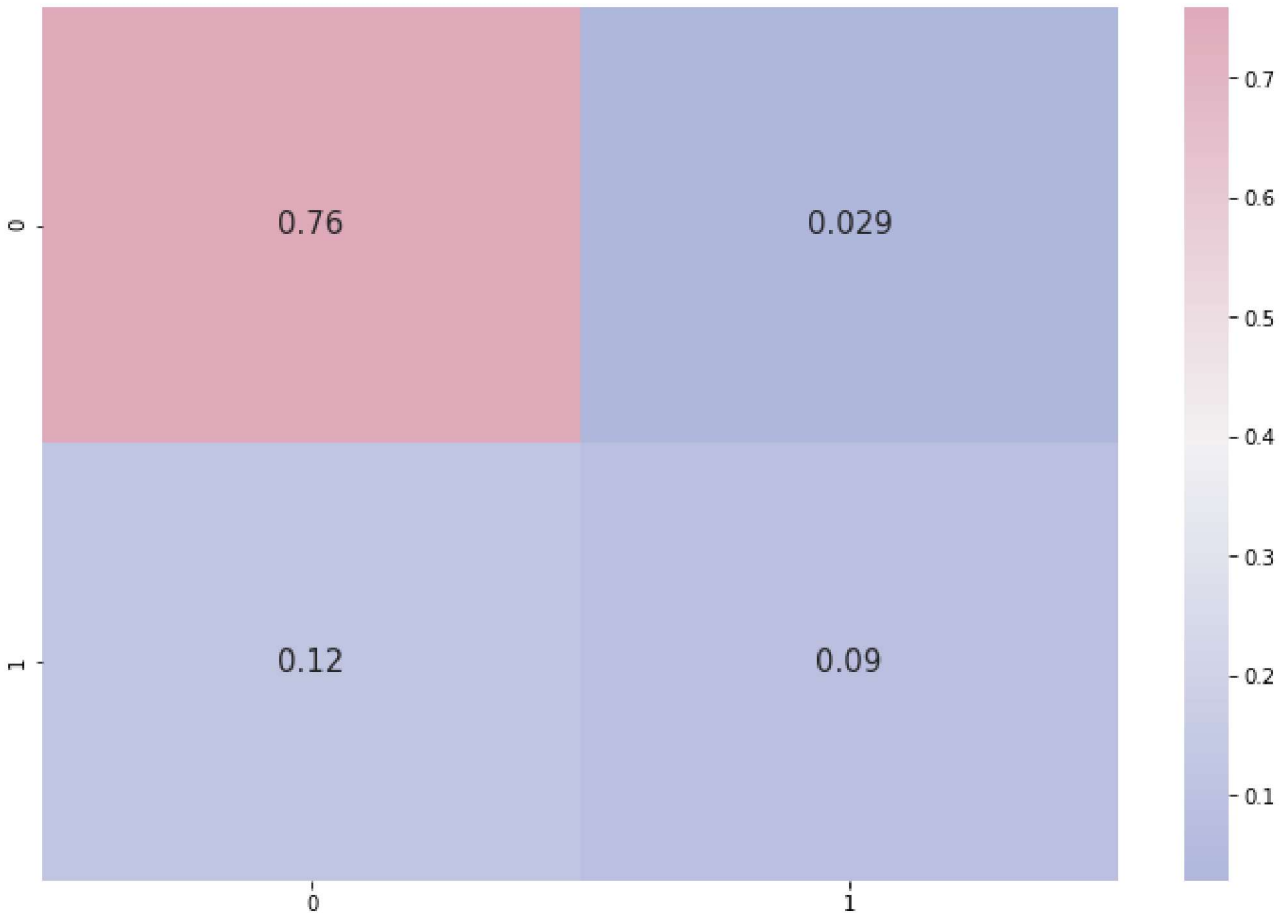
Assessing the classification report

In [29]:
```
# Predicting the test set results
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)
```

In [30]:
```
# 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 = {'size':15})
```

Out[30]: <AxesSubplot:>



In [31]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.96      0.91     20110
           1       0.76      0.43      0.55      5398

    accuracy                           0.85     25508
   macro avg       0.81      0.70      0.73     25508
weighted avg       0.84      0.85      0.83     25508
```