Rain Prediction ANN

This is the model predict Wheather in next day there will be rain or not

In [1]:
```
# import all libraries
import matplotlib.pyplot as plt # for visualisation
import seaborn as sns # for statistical visualisation
import datetime # for date and time
from sklearn.preprocessing import LabelEncoder # for encode the categorical valu
from sklearn.preprocessing import StandardScaler # for scale the value to -3 to 
from sklearn.model_selection import train_test_split # to split the dependent an
# Dense: for conected layer to the neuron
# BatchNormalization: Normalizes the inputs of each layer to improve training st
# Droupout: Prevents overfitting by randomly dropping a fraction of neurons duri
# LSTM: A layer for Long Short-Term Memory networks, designed for sequential dat
from keras.layers import Dense, BatchNormalization, Dropout,LSTM
from keras.models import Sequential #Used to define a linear stack of layers.
from keras.utils import to_categorical #Converts class labels (integers) into on
from keras.optimizers import Adam # optimizer for ANN
from tensorflow.keras import regularizers # feature selection
from sklearn.metrics import precision_score, recall_score,confusion_matrix,class
from keras import callbacks
import pandas as pd # for dataset
import numpy as np # for nD array

# remove warning
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# load the dataset
data=pd.read_csv(r"C:\Users\sunil\Downloads\weatherAUS.csv\weatherAUS.csv")
```

In [3]:
```
data.head() # head of dataset
```

Out[3]:

	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

The dataset contains about 10 years of daily weather observations from different locations across Australia. 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 or not it will rain the next day or not.

In [4]:

```
data.info() # information about the data
```

```
<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
```

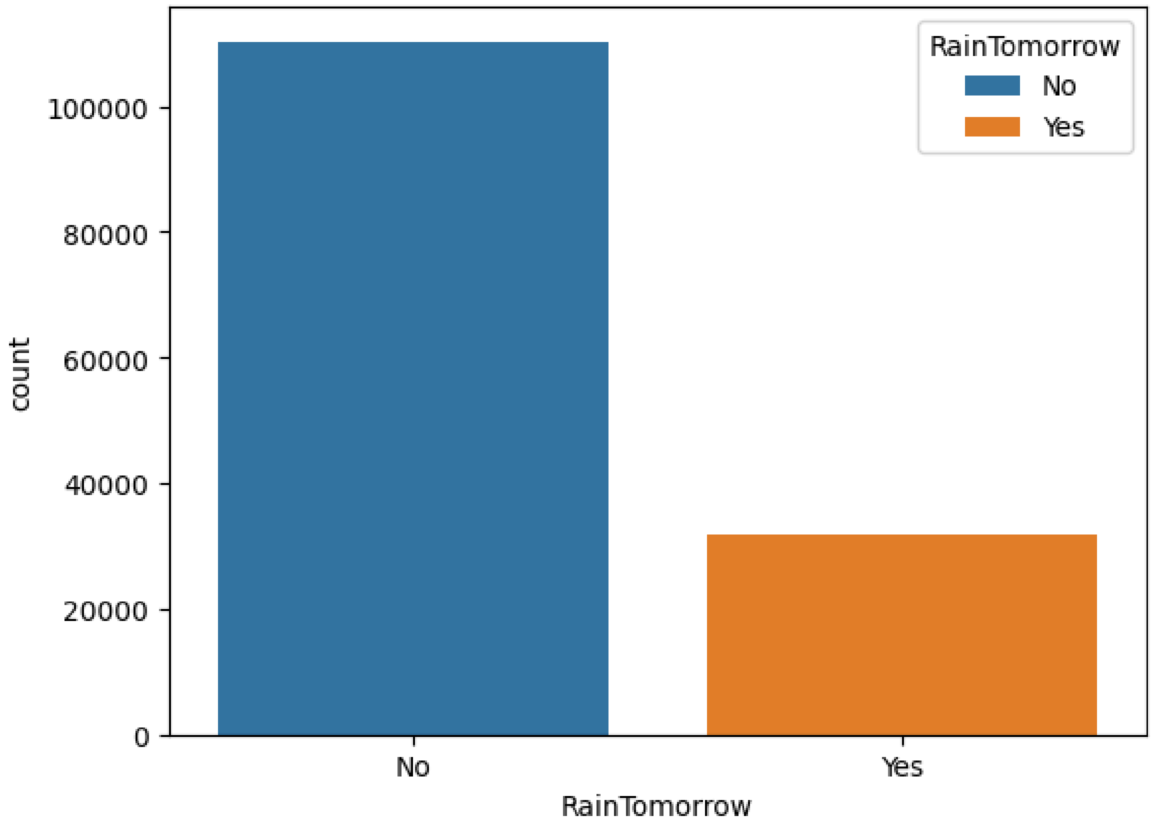
There are some missing values in the data set where both numeric and categorical value are missing

Visualisaition for cleaning

In [5]:
```
# as our target value is raintomorrow will hapen r not
# lets check wheather its balance or not
```

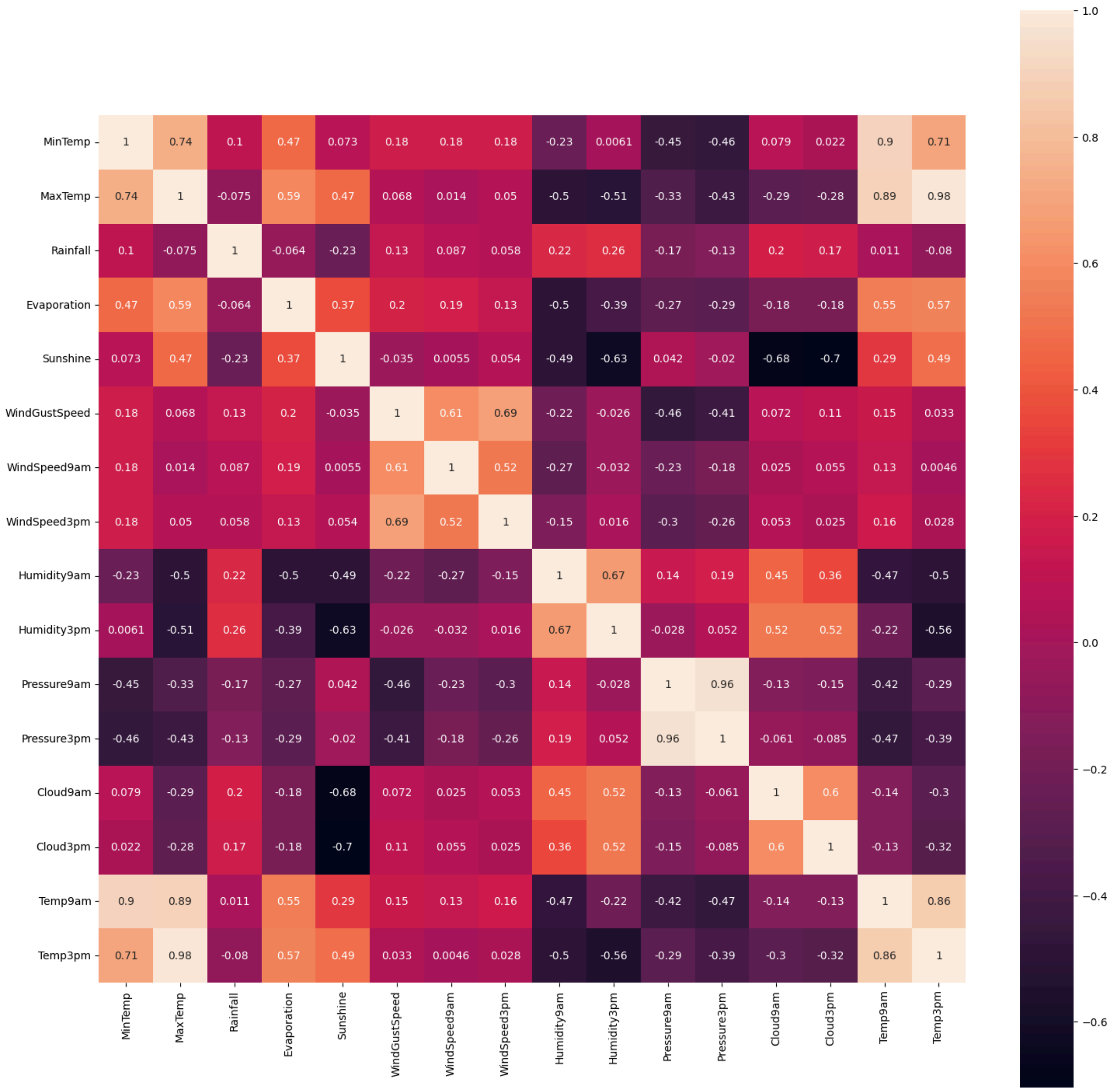
In [6]:
```
sns.countplot(x=data['RainTomorrow'],hue=data["RainTomorrow"])
```

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



In [7]:
```
# take all numerical value one side to find the correlation between them
numeric_data=data[["MinTemp","MaxTemp","Rainfall","Evaporation","Sunshine","Wind
                   "Humidity3pm","Pressure9am","Pressure3pm","Cloud9am","Cloud3p
cor = numeric_data.corr() # find correlation
plt.subplots(figsize=(18,18)) # figure size
sns.heatmap(cor,annot=True,square=True) # represnted by heat map
```

Out[7]:
```
<Axes: >
```



Encode date to months and day sith sin and cos combination to get cyclic continous feature

In [8]:
```
# Parsing Date time
lengths = data["Date"].str.len()
lengths.value_counts()
```

Out[8]:
```
Date
10    145460
Name: count, dtype: int64
```

In [9]:
```
#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[9]:

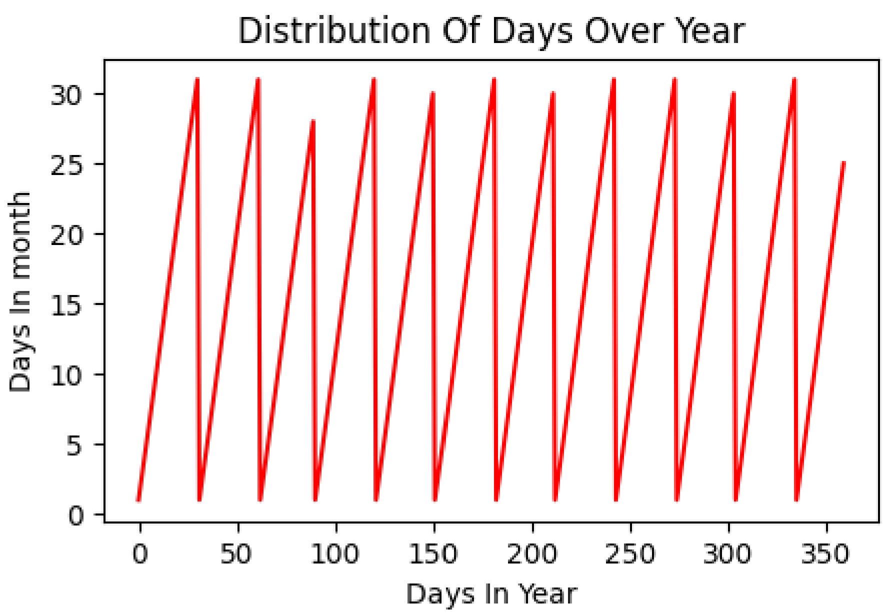
	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W

5 rows × 30 columns

In [10]:

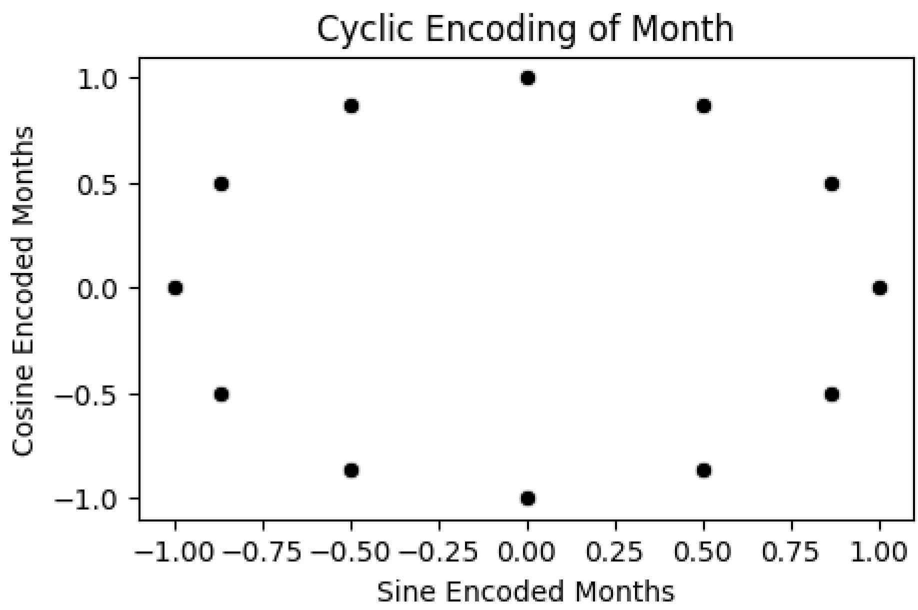
```
# roughly a year's span section
section = data[:360]
plt.subplots(figsize=(5,3))
tm = section["day"].plot(color="red")
tm.set_title("Distribution Of Days Over Year")
tm.set_ylabel("Days In month")
tm.set_xlabel("Days In Year")
```

Out[10]: Text(0.5, 0, 'Days In Year')



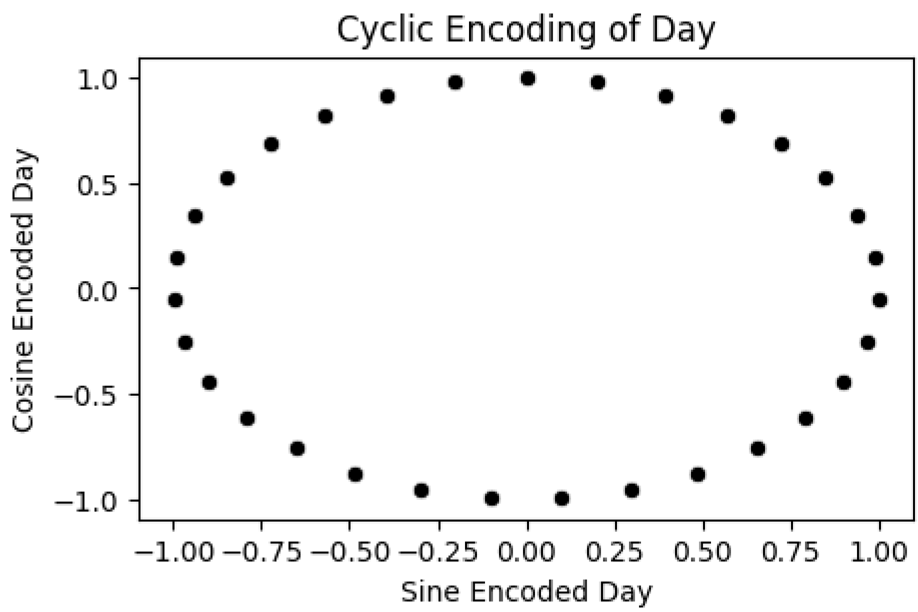
In [11]:
```
plt.subplots(figsize=(5,3))
cyclic_month = sns.scatterplot(x="month_sin",y="month_cos",data=data, color="bla
cyclic_month.set_title("Cyclic Encoding of Month")
cyclic_month.set_ylabel("Cosine Encoded Months")
cyclic_month.set_xlabel("Sine Encoded Months")
```

Out[11]: Text(0.5, 0, 'Sine Encoded Months')



In [12]:
```
plt.subplots(figsize=(5,3))
cyclic_day = sns.scatterplot(x='day_sin',y='day_cos',data=data, color="black")
cyclic_day.set_title("Cyclic Encoding of Day")
cyclic_day.set_ylabel("Cosine Encoded Day")
cyclic_day.set_xlabel("Sine Encoded Day")
```

Out[12]: Text(0.5, 0, 'Sine Encoded Day')



Deal with missing value in the dataset

Categorical value missing treatments

In [13]:
```
# 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 [14]:
```
# check  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 [15]:
```
# Filling missing values with mode of the column in value as categorical

for i in object_cols:
    data[i].fillna(data[i].mode()[0])
```

Numeric value attribute in the dataste

In [16]:
```
# 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 [17]:
```
# checking Missing values in numeric value attribute

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 [18]:

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

for i in num_cols:
    data[i].fillna(data[i].median())

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        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  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
```

Successfully fill the missing value

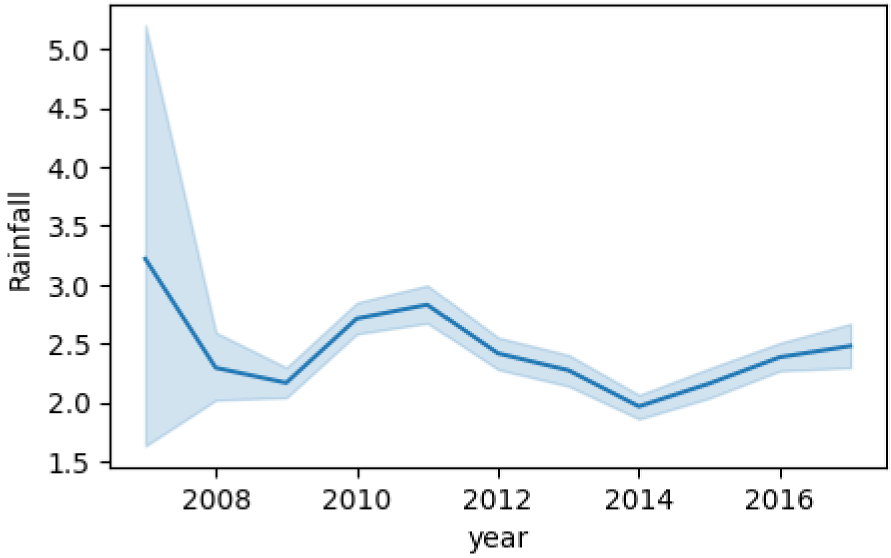
visualisation

In [19]:

```
# line plot to see out year wise rain fall
plt.subplots(figsize=(5,3))
sns.lineplot(x=data["year"],y=data["Rainfall"])
```

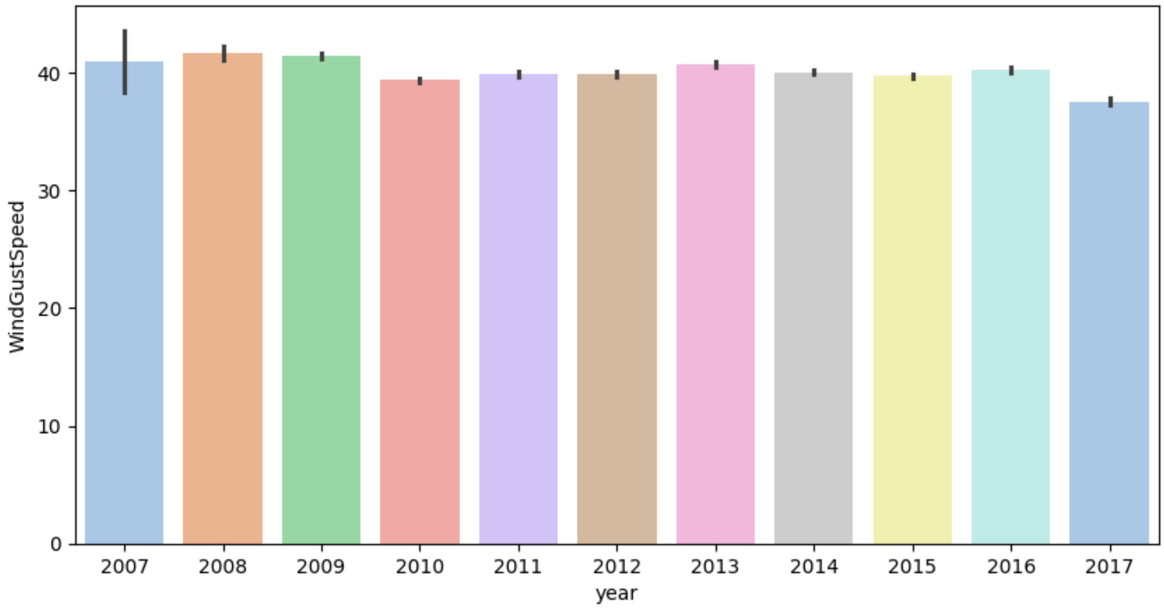
Out[19]:

```
<Axes: xlabel='year', ylabel='Rainfall'>
```



In [20]:
```
# wind speed by the year
plt.subplots(figsize=(10,5))
sns.barplot(x=data["year"],y=data["WindGustSpeed"],palette='pastel')
```

Out[20]: <Axes: xlabel='year', ylabel='WindGustSpeed'>



Data preprocessing

- Lable encoder
- Scaling the data
- Detecting outlier
- Drop outlier

In [21]:
```
# LableEncoder
# 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])
```

```python
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        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    145460 non-null  int32
 8   WindGustSpeed  135197 non-null  float64
 9   WindDir9am     145460 non-null  int32
 10  WindDir3pm     145460 non-null  int32
 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      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 [22]:

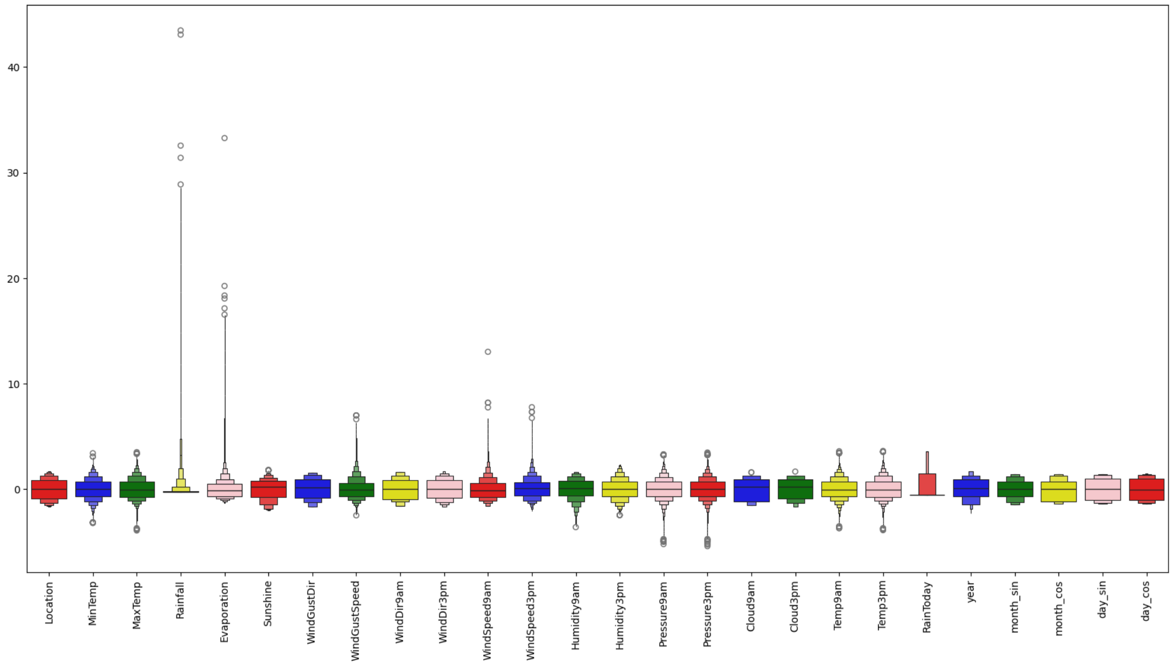
```python
# Scaling the data
# Prepairing attributes of scale data

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 = StandardScaler()
features = s_scaler.fit_transform(features)
features = pd.DataFrame(features, columns=col_names)

features.describe().T
```

Out[22]:

	count	mean	std	min	25%	50%	7
Location	145460.0	7.815677e-18	1.000003	-1.672228	-0.899139	0.014511	0.857
MinTemp	143975.0	-3.790219e-16	1.000003	-3.234215	-0.717989	-0.030325	0.735
MaxTemp	144199.0	-1.387588e-16	1.000003	-3.936122	-0.747483	-0.087280	0.699
Rainfall	142199.0	6.875624e-17	1.000004	-0.278475	-0.278475	-0.278475	-0.184
Evaporation	82670.0	1.732738e-16	1.000006	-1.303922	-0.683942	-0.159343	0.460
Sunshine	75625.0	1.443165e-16	1.000007	-2.010636	-0.742625	0.208382	0.789
WindGustDir	145460.0	6.252542e-18	1.000003	-1.670768	-0.866215	0.139476	0.944
WindGustSpeed	135197.0	-1.731198e-16	1.000004	-2.501301	-0.664013	-0.076081	0.585
WindDir9am	145460.0	4.064152e-17	1.000003	-1.614034	-1.004491	0.011413	0.824
WindDir3pm	145460.0	-7.503050e-17	1.000003	-1.688306	-0.844398	-0.000489	0.843
WindSpeed9am	143693.0	-3.560304e-17	1.000003	-1.575197	-0.790034	-0.117037	0.555
WindSpeed3pm	142398.0	1.812309e-16	1.000004	-2.118405	-0.642770	0.038292	0.605
Humidity9am	142806.0	-2.292747e-16	1.000004	-3.619763	-0.624351	0.058814	0.741
Humidity3pm	140953.0	4.516727e-17	1.000004	-2.478339	-0.699136	0.022162	0.695
Pressure9am	130395.0	-1.277457e-14	1.000004	-5.227598	-0.668393	-0.007027	0.668
Pressure3pm	130432.0	-9.532011e-15	1.000004	-5.421883	-0.690013	-0.007942	0.674
Cloud9am	89572.0	1.167685e-16	1.000006	-1.540437	-1.194074	0.191379	0.884
Cloud3pm	86102.0	-2.376673e-17	1.000006	-1.657854	-0.922653	0.180150	0.915
Temp9am	143693.0	3.797658e-17	1.000003	-3.728099	-0.722889	-0.044790	0.710
Temp3pm	141851.0	5.770458e-16	1.000004	-3.904404	-0.732833	-0.084103	0.679
RainToday	145460.0	9.378812e-18	1.000003	-0.539860	-0.539860	-0.539860	-0.539

	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 [23]:
```
# Detecting outliers
#looking at the scaled features
colours = ["red", "blue", "green", "yellow", "pink"]
plt.figure(figsize=(20,10))
sns.boxenplot(data = features,palette = colours)
plt.xticks(rotation=90)
plt.show()
```



In [24]:
```
# Drop outlier
#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[24]:
```
(52851, 27)
```

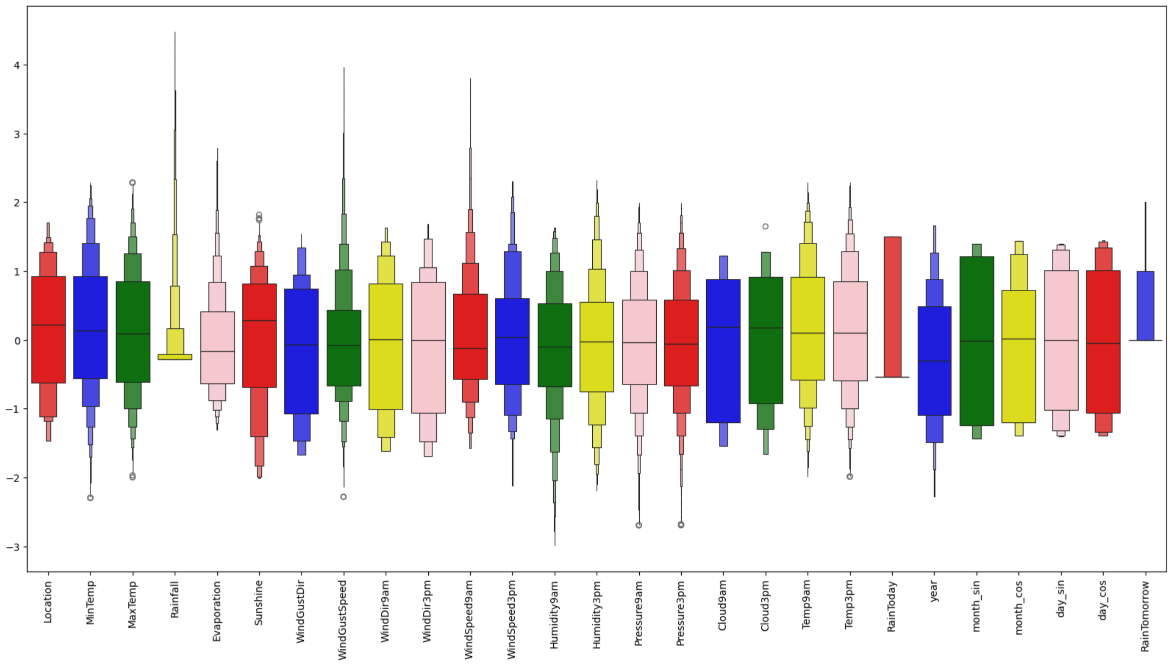
In [25]:
```
features.head()
```

Out[25]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir
6049	-0.96942	0.891770	1.682626	-0.278475	1.557527	1.238641	0.541753
6050	-0.96942	0.969914	0.797673	-0.278475	2.225199	1.423559	-0.061662
6052	-0.96942	1.126201	2.019750	-0.278475	1.271382	0.789554	-0.665077
6053	-0.96942	1.516919	2.132125	-0.278475	1.414455	1.212224	1.145167
6056	-0.96942	1.735721	1.514063	-0.278475	1.032928	1.317891	0.541753

5 rows × 27 columns

In [26]:
```
# visualise afer drop outlier
#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

In this project, we build an artificial neural network.

Following steps are involved in the model building

- Assining X and y the status of attributes and tags
- Splitting test and training sets
- Initialising the neural network
- Defining by adding layers
- Compiling the neural network
- Train the neural network

In [27]:

```
# dependent and independet variable
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[27]: (52851, 26)

In [28]:

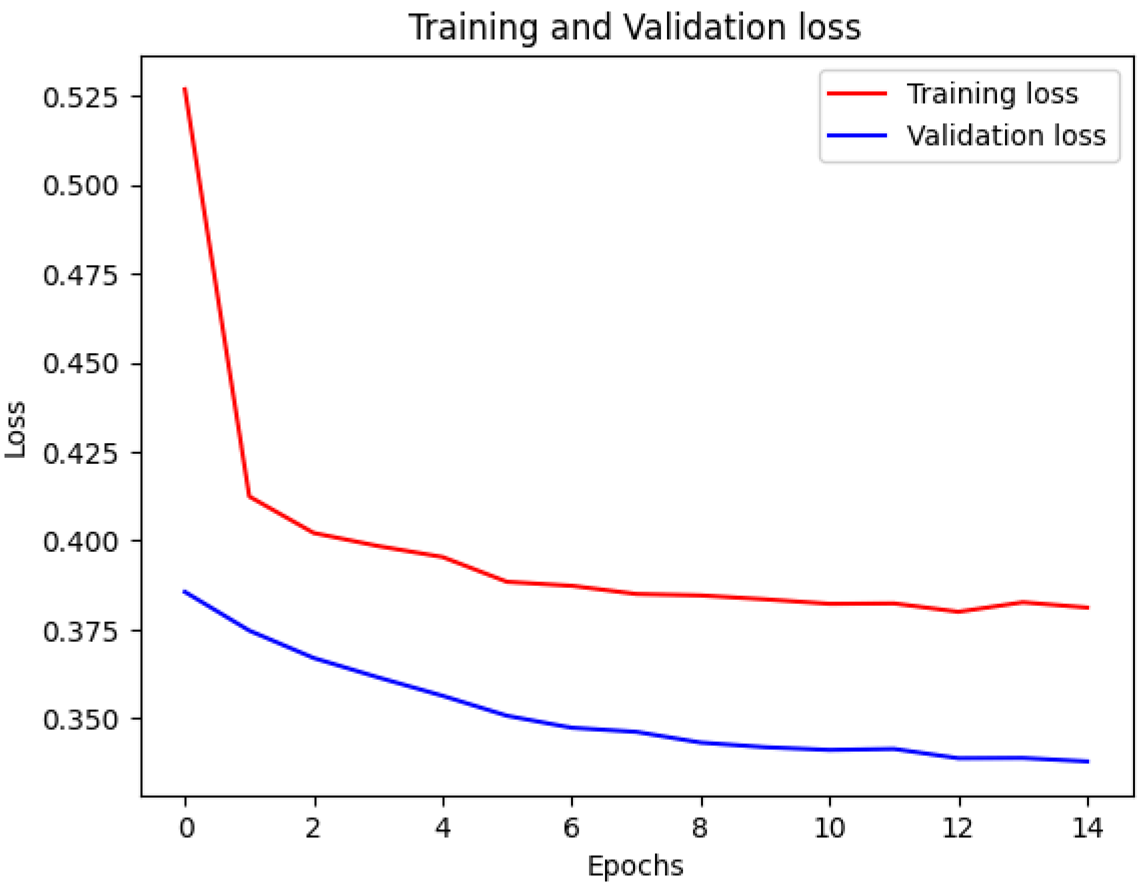
```
# Initialising the ANN
model = Sequential()

# layers

model.add(Dense(units = 32, kernel_initializer = 'uniform', activation = 'relu',
```

```python
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 with 15 epochs
history = model.fit(X_train, y_train, batch_size = 32, epochs = 15, validation_s
```

```
Epoch 1/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.7805 - loss: 0.6223 - va
l_accuracy: 0.7835 - val_loss: 0.3856
Epoch 2/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.7817 - loss: 0.4125 - va
l_accuracy: 0.7835 - val_loss: 0.3747
Epoch 3/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.7993 - loss: 0.4039 - va
l_accuracy: 0.8479 - val_loss: 0.3670
Epoch 4/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.8142 - loss: 0.3956 - va
l_accuracy: 0.8491 - val_loss: 0.3615
Epoch 5/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8127 - loss: 0.3921 - va
l_accuracy: 0.8499 - val_loss: 0.3564
Epoch 6/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.8104 - loss: 0.3910 - va
l_accuracy: 0.8512 - val_loss: 0.3507
Epoch 7/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.8142 - loss: 0.3902 - va
l_accuracy: 0.8521 - val_loss: 0.3474
Epoch 8/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.8191 - loss: 0.3759 - va
l_accuracy: 0.8530 - val_loss: 0.3462
Epoch 9/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.8106 - loss: 0.3873 - va
l_accuracy: 0.8550 - val_loss: 0.3432
Epoch 10/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8156 - loss: 0.3859 - va
l_accuracy: 0.8551 - val_loss: 0.3419
Epoch 11/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8096 - loss: 0.3909 - va
l_accuracy: 0.8561 - val_loss: 0.3411
Epoch 12/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.8231 - loss: 0.3752 - va
l_accuracy: 0.8569 - val_loss: 0.3414
Epoch 13/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.8199 - loss: 0.3794 - va
l_accuracy: 0.8556 - val_loss: 0.3388
Epoch 14/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.8159 - loss: 0.3847 - va
l_accuracy: 0.8566 - val_loss: 0.3389
Epoch 15/15
1057/1057 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8215 - loss: 0.3799 - va
l_accuracy: 0.8567 - val_loss: 0.3379
```

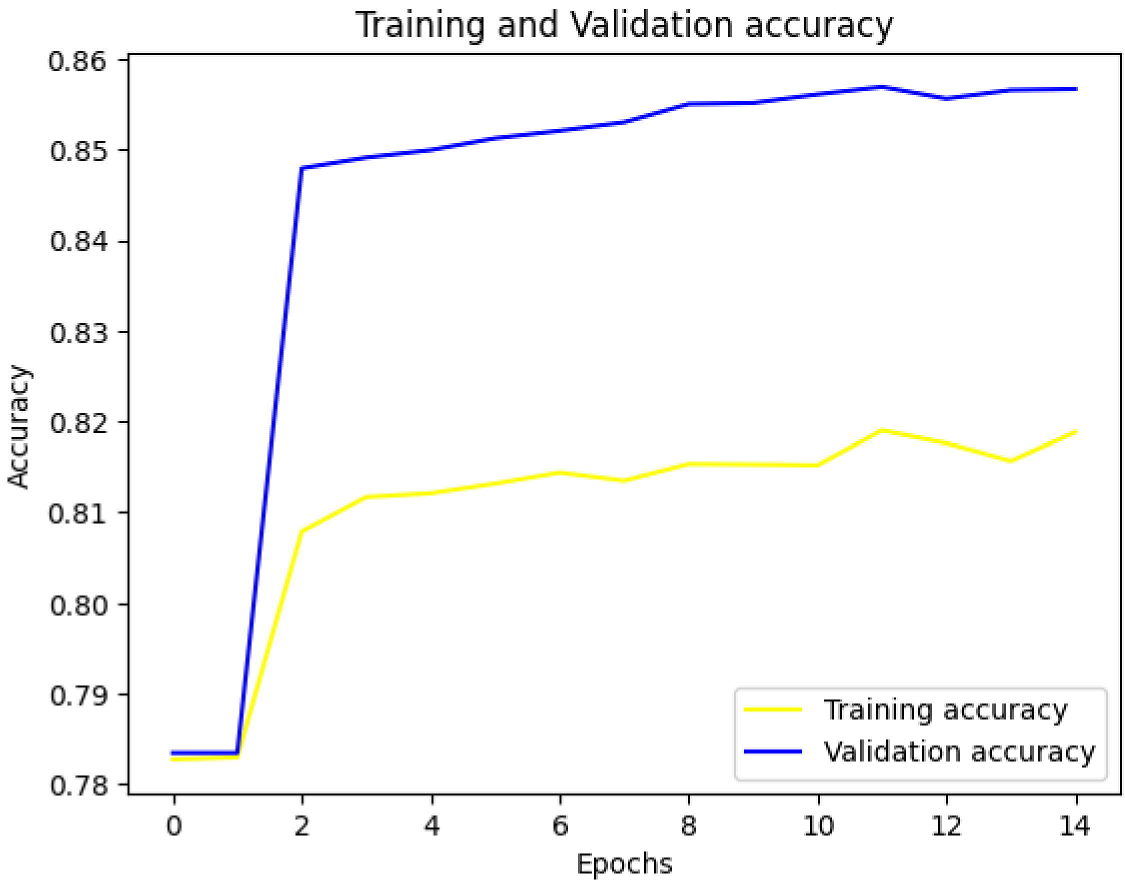
Plotting training and validation loss over epochs

In [29]:
```
history_df = pd.DataFrame(history.history)

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

plt.show()
```



Plotting training and validation accuracy over epochs

In [30]:
```
history_df = pd.DataFrame(history.history)

plt.plot(history_df.loc[:, ['accuracy']], "yellow", label='Training accuracy')
plt.plot(history_df.loc[:, ['val_accuracy']], "blue", label='Validation accuracy

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



Conclusion

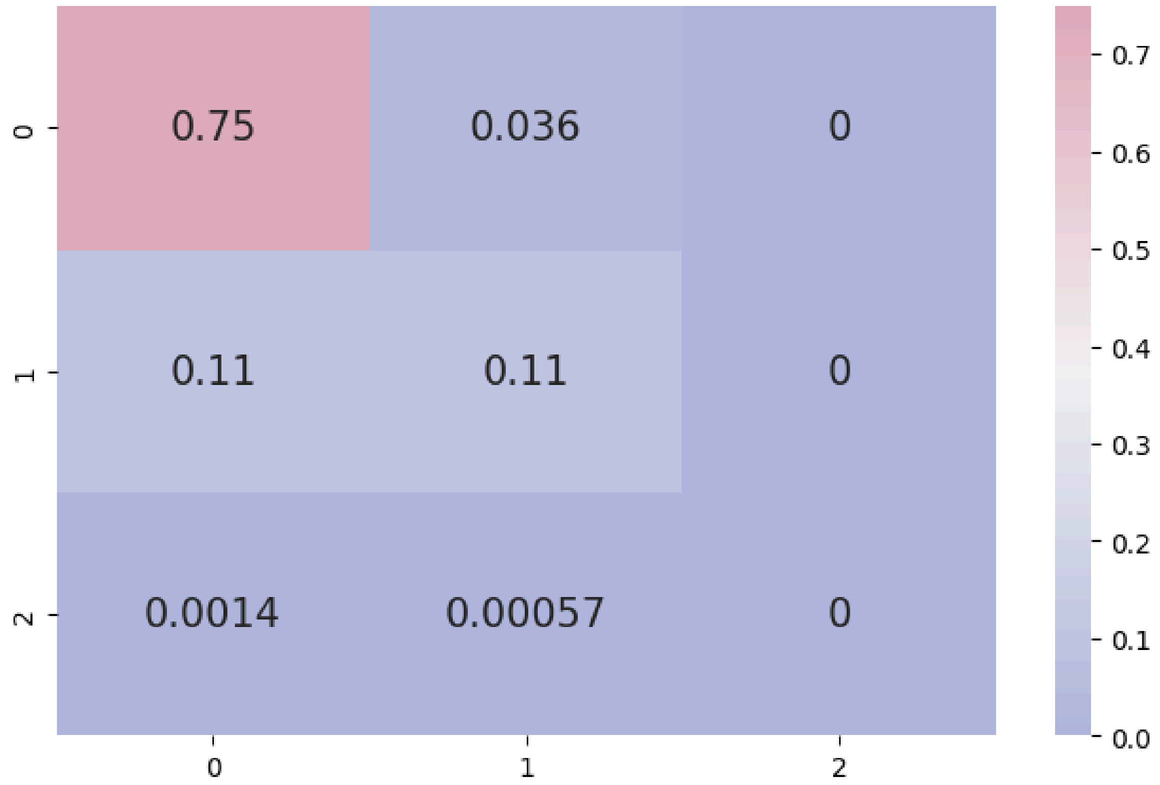
- Testing on the test set
- Evaluating the confusion matrix
- Evaluating the classification report

In [31]:
```
# Predicting the test set results
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)
```

```
331/331 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
```

In [32]:
```
# confusion matrix
cmap1 = sns.diverging_palette(260,-10,s=50, l=75, n=5, as_cmap=True)
plt.subplots(figsize=(8,5))
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix/np.sum(cf_matrix), cmap = cmap1, annot = True, annot_kws =
```

Out[32]:
```
<Axes: >
```



In [33]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.91      8296
           1       0.75      0.51      0.60      2254
           2       0.00      0.00      0.00        21

    accuracy                           0.86     10571
   macro avg       0.54      0.49      0.51     10571
weighted avg       0.85      0.86      0.84     10571
```

The model train with 85% accuracy

complete

In [ ]: