```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
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_score
from keras import callbacks
```

```
df=pd.read_csv("weatherAUS.csv")
```

```
df
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGus
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

```
df.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
```

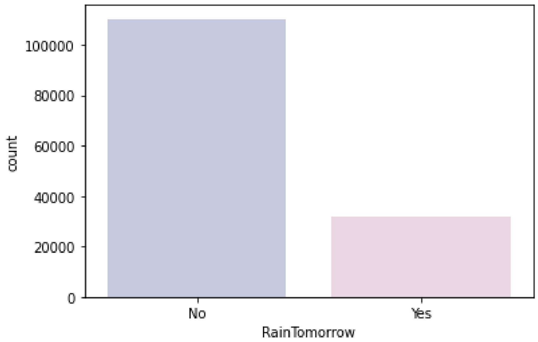
```
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
 20  Temp3pm        141851 non-null  float64
 21  RainToday      142199 non-null  object
 22  RainTomorrow   142193 non-null  object
dtypes: float64(16), object(7)
memory usage: 25.5+ MB
```

```
#Points to notice:

#There are missing values in the dataset
#Dataset includes numeric and categorical values
```

```
#DATA VISUALIZATION AND CLEANING
```

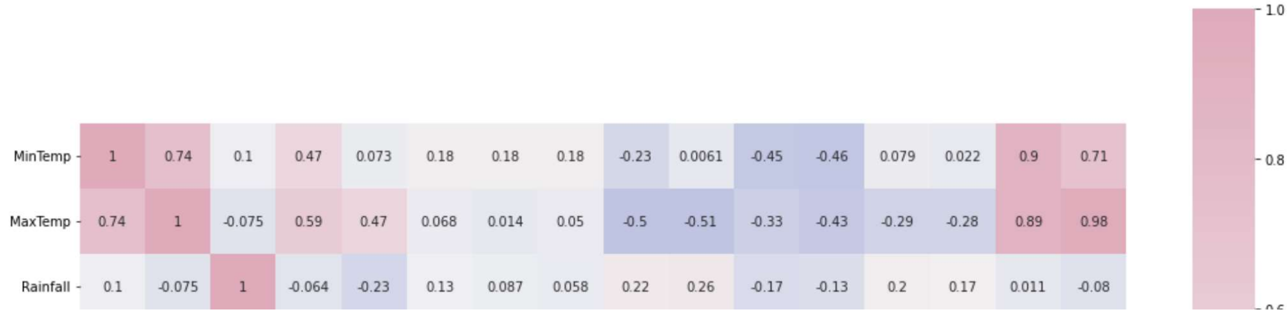
```
cols= ["#C2C4E2","#EED4E5"]
sns.countplot(x= df["RainTomorrow"], palette= cols)
```

```
<Axes: xlabel='RainTomorrow', ylabel='count'>
```




```
corrmat = df.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)
```

```
<Axes: >
```




```
lengths = df["Date"].str.len()
lengths.value_counts()
```

```
10    145460
Name: Date, dtype: int64
```

```
df['Date']= pd.to_datetime(df["Date"])
df['year'] = df.Date.dt.year
```

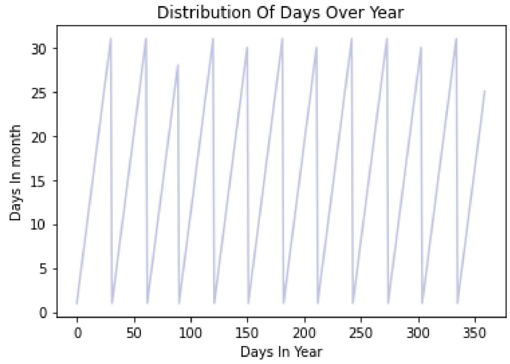
```
def encode(df, col, max_val):
    df[col + '_sin'] = np.sin(2 * np.pi * df[col]/max_val)
    df[col + '_cos'] = np.cos(2 * np.pi * df[col]/max_val)
    return df

df['month'] = df.Date.dt.month
df = encode(df, 'month', 12)

df['day'] = df.Date.dt.day
df = encode(df, 'day', 31)

df.head()
```

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

5 rows × 30 columns

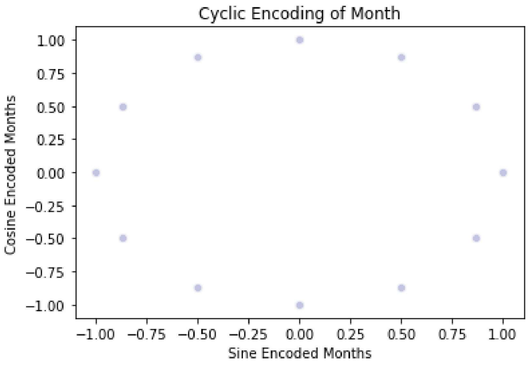
```
section = df[:360]
tm = section["day"].plot(color="#C2C4E2")
tm.set_title("Distribution Of Days Over Year")
tm.set_ylabel("Days In month")
tm.set_xlabel("Days In Year")
```

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



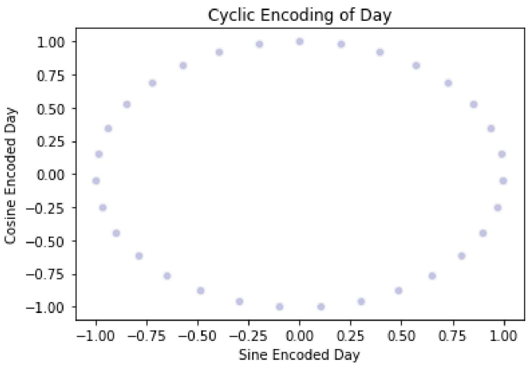
```
cyclic_month = sns.scatterplot(x="month_sin",y="month_cos",data=df, color="#C2C4E2")
cyclic_month.set_title("Cyclic Encoding of Month")
cyclic_month.set_ylabel("Cosine Encoded Months")
cyclic_month.set_xlabel("Sine Encoded Months")
```

```
Text(0.5, 0, 'Sine Encoded Months')
```




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

```
Text(0.5, 0, 'Sine Encoded Day')
```




```
#Filling missing values with mode of the column value
```

```
#Get list of categorical variables
s = (df.dtypes == "object")
object_cols = list(s[s].index)

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

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

```
# Missing values in categorical variables

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

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

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

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

```
#Numerical variables

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

```
# Get list of neumeric variables
t = (df.dtypes == "float64")
```

```
num_cols = list(t[t].index)

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

```
Neumeric variables:
['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humi
```

```
# Missing values in numeric variables

for i in num_cols:
    print(i, df[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
```

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

for i in num_cols:
    df[i].fillna(df[i].median(), inplace=True)

df.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
```

```
#DATA PREPROCESSING
```

```
#Label encoding the catagorical varable
```

```
# Apply label encoder to each column with categorical data
```

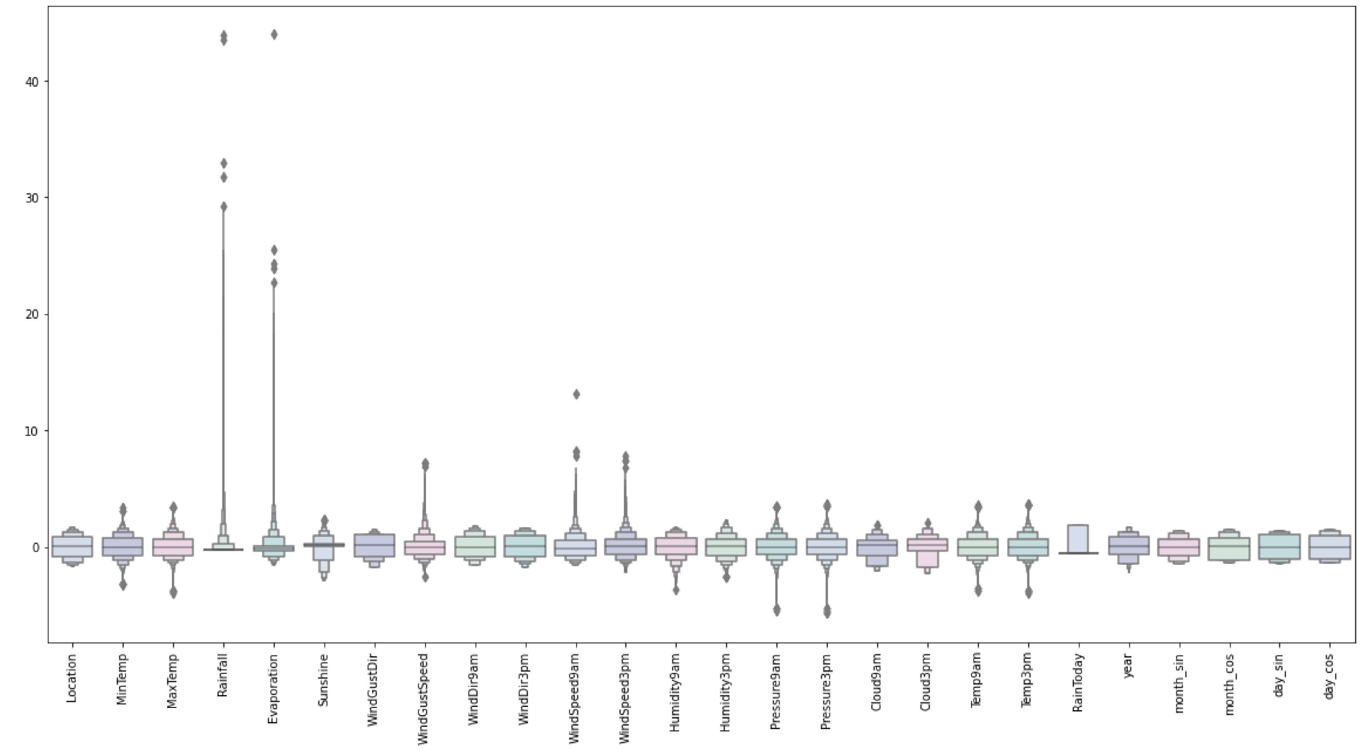
```
label_encoder = LabelEncoder()
for i in object_cols:
    df[i] = label_encoder.fit_transform(df[i])

df.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
```

```
# Prepairing attributes of scale data


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

target = df['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
```

	count	mean	std	min	25%	50%	75%	max
Location	145460.0	7.815677e-18	1.000003	-1.672228	-0.899139	0.014511	0.857881	1.701250
MinTemp	145460.0	-4.501830e-16	1.000003	-3.250525	-0.705659	-0.030170	0.723865	3.410112
MaxTemp	145460.0	3.001220e-16	1.000003	-3.952405	-0.735852	-0.086898	0.703133	3.510563
Rainfall	145460.0	7.815677e-18	1.000003	-0.275097	-0.275097	-0.275097	-0.203581	43.945571
Evaporation	145460.0	-3.282584e-17	1.000003	-1.629472	-0.371139	-0.119472	0.006361	43.985108
Sunshine	145460.0	-5.424080e-16	1.000003	-2.897217	0.076188	0.148710	0.257494	2.360634
WindGustDir	145460.0	6.252542e-18	1.000003	-1.724209	-0.872075	0.193094	1.045228	1.471296
WindGustSpeed	145460.0	1.824961e-16	1.000003	-2.588407	-0.683048	-0.073333	0.460168	7.243246
WindDir9am	145460.0	7.190423e-17	1.000003	-1.550000	-0.885669	0.000105	0.885879	1.771653
WindDir3pm	145460.0	8.284618e-17	1.000003	-1.718521	-0.837098	0.044324	0.925747	1.586813
WindSpeed9am	145460.0	5.627287e-17	1.000003	1.583291	0.793380	0.116314	0.560752	13.086472

```
#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()
```




```
features["RainTomorrow"] = target
```

```
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
```

    (145460, 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,
```

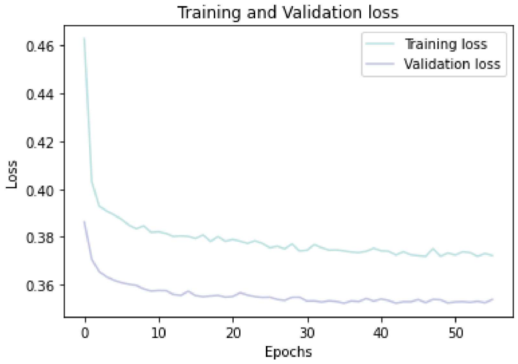
```
)

# Initialising the NN
model = Sequential()

# layers

model.add(Dense(units = 32,activation = 'relu', input_dim = 26))
model.add(Dense(units = 32,activation = 'relu'))
model.add(Dense(units = 16,activation = 'relu'))
model.add(Dropout(0.25))
model.add(Dense(units = 1,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=50, callbacks=[early_stopping], validation_split=0.2)
```

```
Epoch 28/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3750 - accuracy: 0.8339 - val_loss: 0.3535 - val_accuracy: 0.8
Epoch 29/150
2910/2910 [==============================] - 11s 4ms/step - loss: 0.3771 - accuracy: 0.8337 - val_loss: 0.3548 - val_accuracy: 0.
Epoch 30/150
2910/2910 [==============================] - 8s 3ms/step - loss: 0.3742 - accuracy: 0.8330 - val_loss: 0.3549 - val_accuracy: 0.8
Epoch 31/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3745 - accuracy: 0.8332 - val_loss: 0.3533 - val_accuracy: 0.8
Epoch 32/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3768 - accuracy: 0.8330 - val_loss: 0.3533 - val_accuracy: 0.8
Epoch 33/150
2910/2910 [==============================] - 10s 3ms/step - loss: 0.3755 - accuracy: 0.8328 - val_loss: 0.3528 - val_accuracy: 0.
Epoch 34/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3745 - accuracy: 0.8337 - val_loss: 0.3534 - val_accuracy: 0.8
Epoch 35/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3746 - accuracy: 0.8342 - val_loss: 0.3531 - val_accuracy: 0.8
Epoch 36/150
2910/2910 [==============================] - 11s 4ms/step - loss: 0.3742 - accuracy: 0.8331 - val_loss: 0.3524 - val_accuracy: 0.
Epoch 37/150
2910/2910 [==============================] - 10s 4ms/step - loss: 0.3737 - accuracy: 0.8334 - val_loss: 0.3533 - val_accuracy: 0.
Epoch 38/150
2910/2910 [==============================] - 8s 3ms/step - loss: 0.3735 - accuracy: 0.8341 - val_loss: 0.3530 - val_accuracy: 0.8
Epoch 39/150
2910/2910 [==============================] - 10s 3ms/step - loss: 0.3740 - accuracy: 0.8332 - val_loss: 0.3543 - val_accuracy: 0.
Epoch 40/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3753 - accuracy: 0.8334 - val_loss: 0.3532 - val_accuracy: 0.8
Epoch 41/150
2910/2910 [==============================] - 10s 3ms/step - loss: 0.3742 - accuracy: 0.8331 - val_loss: 0.3541 - val_accuracy: 0.
Epoch 42/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3741 - accuracy: 0.8339 - val_loss: 0.3535 - val_accuracy: 0.8
Epoch 43/150
2910/2910 [==============================] - 12s 4ms/step - loss: 0.3725 - accuracy: 0.8342 - val_loss: 0.3524 - val_accuracy: 0.
Epoch 44/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3738 - accuracy: 0.8328 - val_loss: 0.3529 - val_accuracy: 0.8
Epoch 45/150
2910/2910 [==============================] - 10s 3ms/step - loss: 0.3726 - accuracy: 0.8346 - val_loss: 0.3529 - val_accuracy: 0.
Epoch 46/150
2910/2910 [==============================] - 8s 3ms/step - loss: 0.3722 - accuracy: 0.8334 - val_loss: 0.3539 - val_accuracy: 0.8
Epoch 47/150
2910/2910 [==============================] - 10s 3ms/step - loss: 0.3719 - accuracy: 0.8339 - val_loss: 0.3526 - val_accuracy: 0.
Epoch 48/150
2910/2910 [==============================] - 10s 3ms/step - loss: 0.3751 - accuracy: 0.8328 - val_loss: 0.3540 - val_accuracy: 0.
Epoch 49/150
2910/2910 [==============================] - 8s 3ms/step - loss: 0.3719 - accuracy: 0.8333 - val_loss: 0.3538 - val_accuracy: 0.8
Epoch 50/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3733 - accuracy: 0.8342 - val_loss: 0.3525 - val_accuracy: 0.8
Epoch 51/150
2910/2910 [==============================] - 11s 4ms/step - loss: 0.3725 - accuracy: 0.8339 - val_loss: 0.3529 - val_accuracy: 0.
Epoch 52/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3738 - accuracy: 0.8334 - val_loss: 0.3530 - val_accuracy: 0.8
Epoch 53/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3735 - accuracy: 0.8344 - val_loss: 0.3527 - val_accuracy: 0.8
Epoch 54/150
2910/2910 [==============================] - 10s 3ms/step - loss: 0.3719 - accuracy: 0.8338 - val_loss: 0.3531 - val_accuracy: 0.
Epoch 55/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3731 - accuracy: 0.8347 - val_loss: 0.3526 - val_accuracy: 0.8
Epoch 56/150
2910/2910 [==============================] - 9s 3ms/step - loss: 0.3722 - accuracy: 0.8342 - val_loss: 0.3539 - val_accuracy: 0.8
```

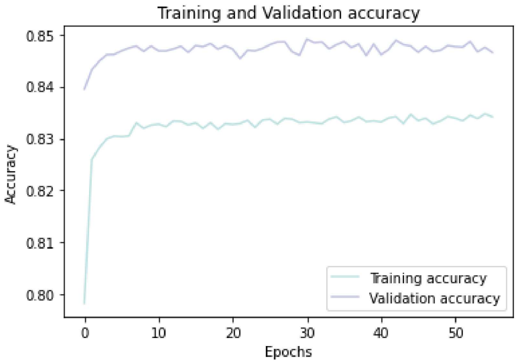
```
history_df = pd.DataFrame(history.history)

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")
```

```
<matplotlib.legend.Legend at 0x7fc8ec0f9850>
```




```
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()
```

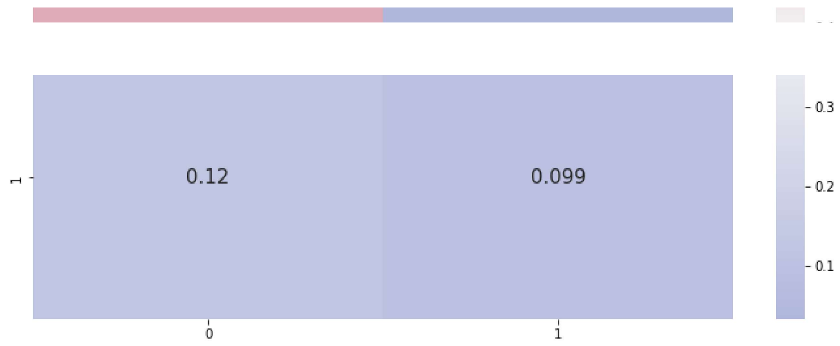



```
# Predicting the test set results
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)
# confusion matrix
cmap1 = sns.diverging_palette(260,-10,s=50, l=75, n=5, as_cmap=True)
plt.subplots(figsize=(12,8))
cf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(cf_matrix/np.sum(cf_matrix), cmap = cmap1, annot = True, annot_kws = {'size':15})
```

```
910/910 [==============================] - 2s 2ms/step
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.96      0.91     22672
           1       0.75      0.45      0.56      6420

    accuracy                           0.85     29092
   macro avg       0.80      0.70      0.73     29092
weighted avg       0.84      0.85      0.83     29092
```



✓ 0s completed at 12:53 PM