In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```python
df=pd.read_csv("/content/weatherAUS.csv") # loaded the data set
df
```

Out[2]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunsh
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	N
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	N
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	N
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	N
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	N
...	...	...	...	...	...	...	
145455	2017-06-21	Uluru	2.8	23.4	0.0	NaN	N
145456	2017-06-22	Uluru	3.6	25.3	0.0	NaN	N
145457	2017-06-23	Uluru	5.4	26.9	0.0	NaN	N
145458	2017-06-24	Uluru	7.8	27.0	0.0	NaN	N
145459	2017-06-25	Uluru	14.9	NaN	0.0	NaN	N

145460 rows × 23 columns

In [4]:
```python
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
```

In [5]:
```
df.describe()
```

Out[5]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
count	143975.000000	144199.000000	142199.000000	82670.000000	75625.000000
mean	12.194034	23.221348	2.360918	5.468232	7.611178
std	6.398495	7.119049	8.478060	4.193704	3.785483
min	-8.500000	-4.800000	0.000000	0.000000	0.000000
25%	7.600000	17.900000	0.000000	2.600000	4.800000
50%	12.000000	22.600000	0.000000	4.800000	8.400000
75%	16.900000	28.200000	0.800000	7.400000	10.600000
max	33.900000	48.100000	371.000000	145.000000	14.500000

In [6]:
```
df.isnull().sum()
```

Out[6]:

	0
Date	0
Location	0
MinTemp	1485
MaxTemp	1261
Rainfall	3261
Evaporation	62790
Sunshine	69835
WindGustDir	10326
WindGustSpeed	10263
WindDir9am	10566
WindDir3pm	4228
WindSpeed9am	1767
WindSpeed3pm	3062
Humidity9am	2654
Humidity3pm	4507
Pressure9am	15065
Pressure3pm	15028
Cloud9am	55888
Cloud3pm	59358
Temp9am	1767
Temp3pm	3609
RainToday	3261
RainTomorrow	3267

dtype: int64

In [7]:

```
df.dropna(axis=0, subset=['RainToday', 'RainTomorrow'], inplace=True) # Becaus
```

In [8]:

```
df.isnull().sum()
```

Out[8]:

	0
Date	0
Location	0
MinTemp	468
MaxTemp	307
Rainfall	0
Evaporation	59694
Sunshine	66805
WindGustDir	9163
WindGustSpeed	9105
WindDir9am	9660
WindDir3pm	3670
WindSpeed9am	1055
WindSpeed3pm	2531
Humidity9am	1517
Humidity3pm	3501
Pressure9am	13743
Pressure3pm	13769
Cloud9am	52625
Cloud3pm	56094
Temp9am	656
Temp3pm	2624
RainToday	0
RainTomorrow	0

dtype: int64

In [10]:
```
df.drop(['Evaporation','Sunshine','Cloud9am','Cloud3pm'], axis=1, inplace=True
```

Why drop them?

Too many guesses needed

Neural networks hate noisy data

Improves model accuracy

In [11]:

```
num_cols = df.select_dtypes(include=['float64','int64']).columns
df[num_cols] = df[num_cols].fillna(df[num_cols].median())
```

In [12]:

```
cat_cols = ['WindGustDir','WindDir9am','WindDir3pm']

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

```
/tmp/ipython-input-3546851707.py:4: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df[col].fillna(df[col].mode()[0], inplace=True)
```

In [13]:

```
df['Date'] = pd.to_datetime(df['Date'])
df['Month'] = df['Date'].dt.month
df.drop('Date', axis=1, inplace=True)
```

In [14]:

```
df.isnull().sum()
```

Out[14]:

	0
Location	0
MinTemp	0
MaxTemp	0
Rainfall	0
WindGustDir	0
WindGustSpeed	0
WindDir9am	0
WindDir3pm	0
WindSpeed9am	0
WindSpeed3pm	0
Humidity9am	0
Humidity3pm	0
Pressure9am	0
Pressure3pm	0
Temp9am	0
Temp3pm	0
RainToday	0
RainTomorrow	0
Month	0

dtype: int64

In [15]:

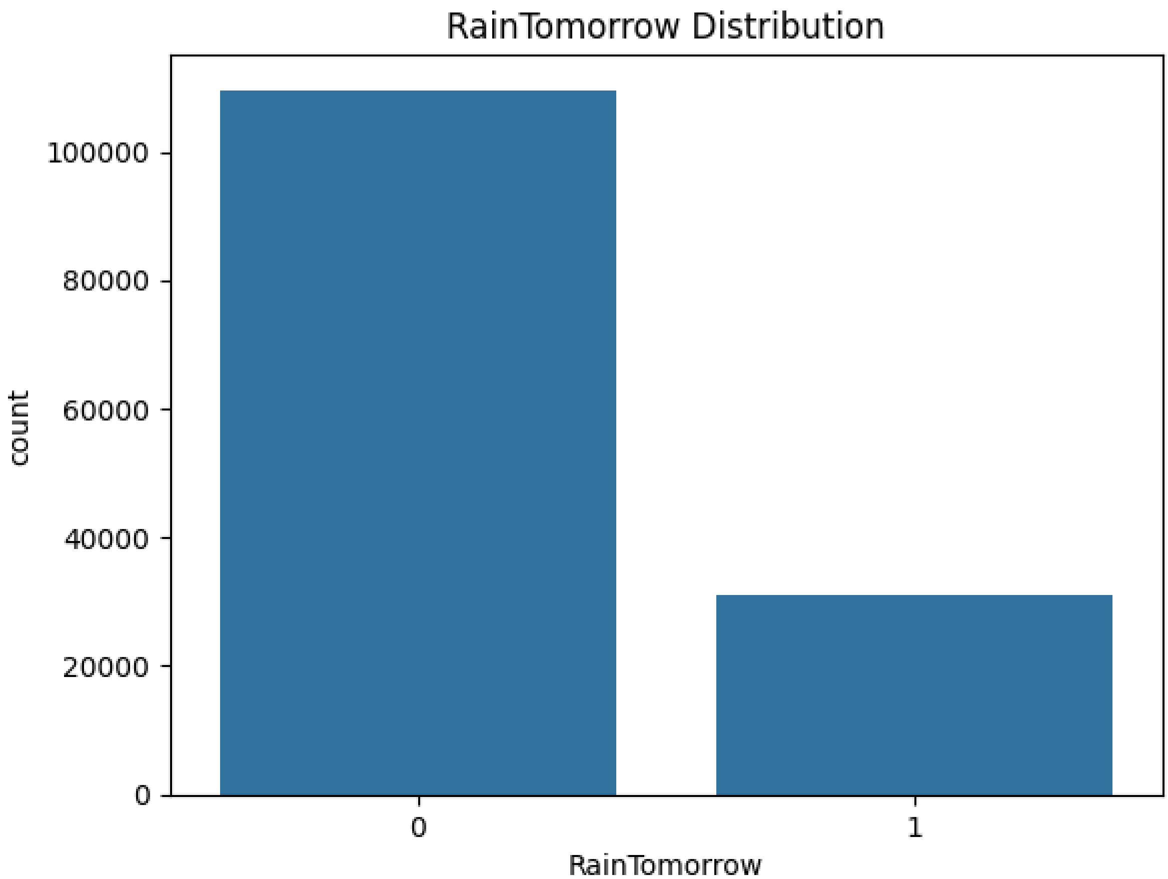
```
df = pd.get_dummies(
    df,
    columns=['Location','WindGustDir','WindDir9am','WindDir3pm'],
    drop_first=True
)
```

In [16]:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df['RainToday'] = le.fit_transform(df['RainToday'])
df['RainTomorrow'] = le.fit_transform(df['RainTomorrow'])
```

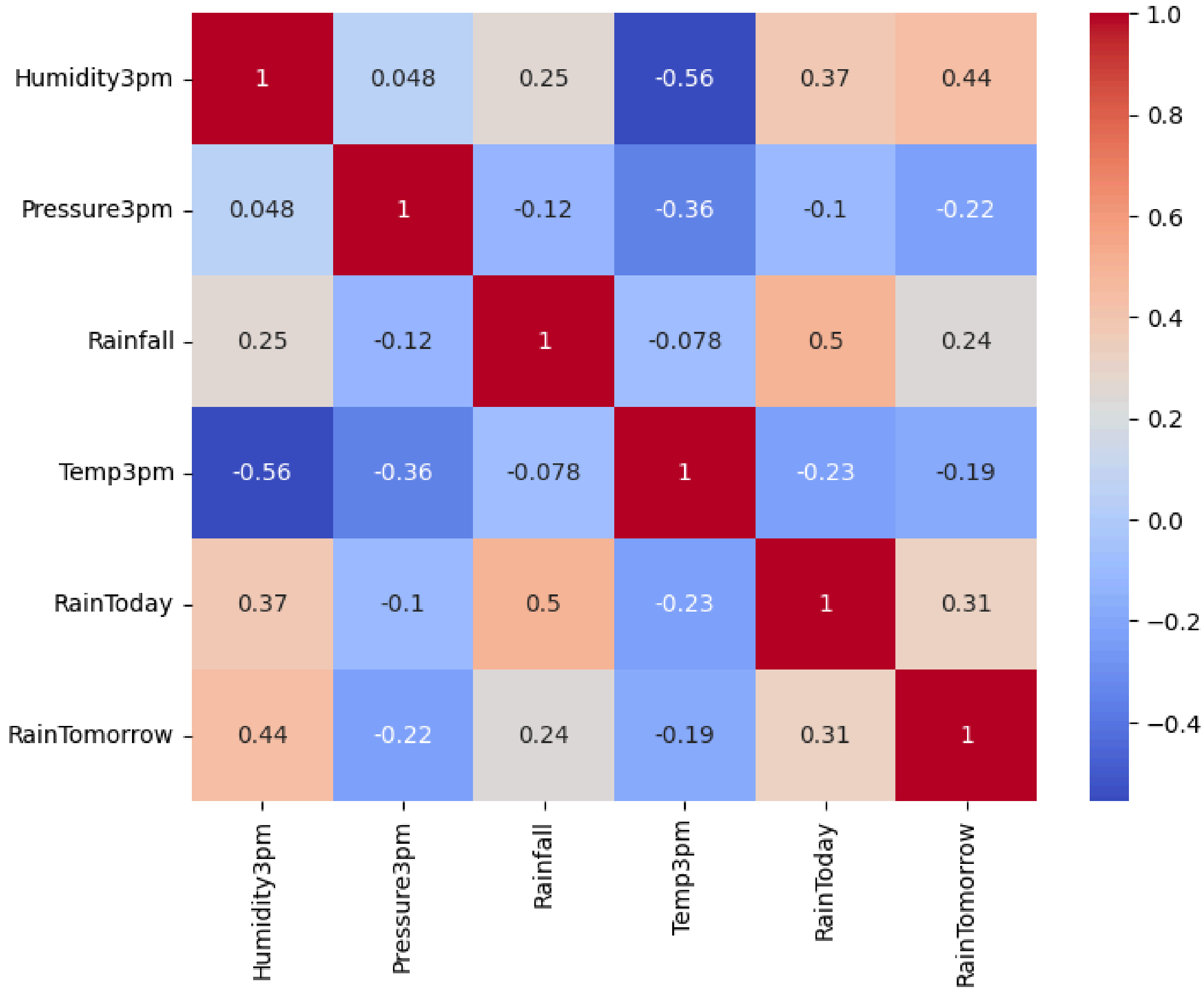
Target Variable Distribution

In [18]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x='RainTomorrow', data=df)
plt.title("RainTomorrow Distribution")
plt.show()
```



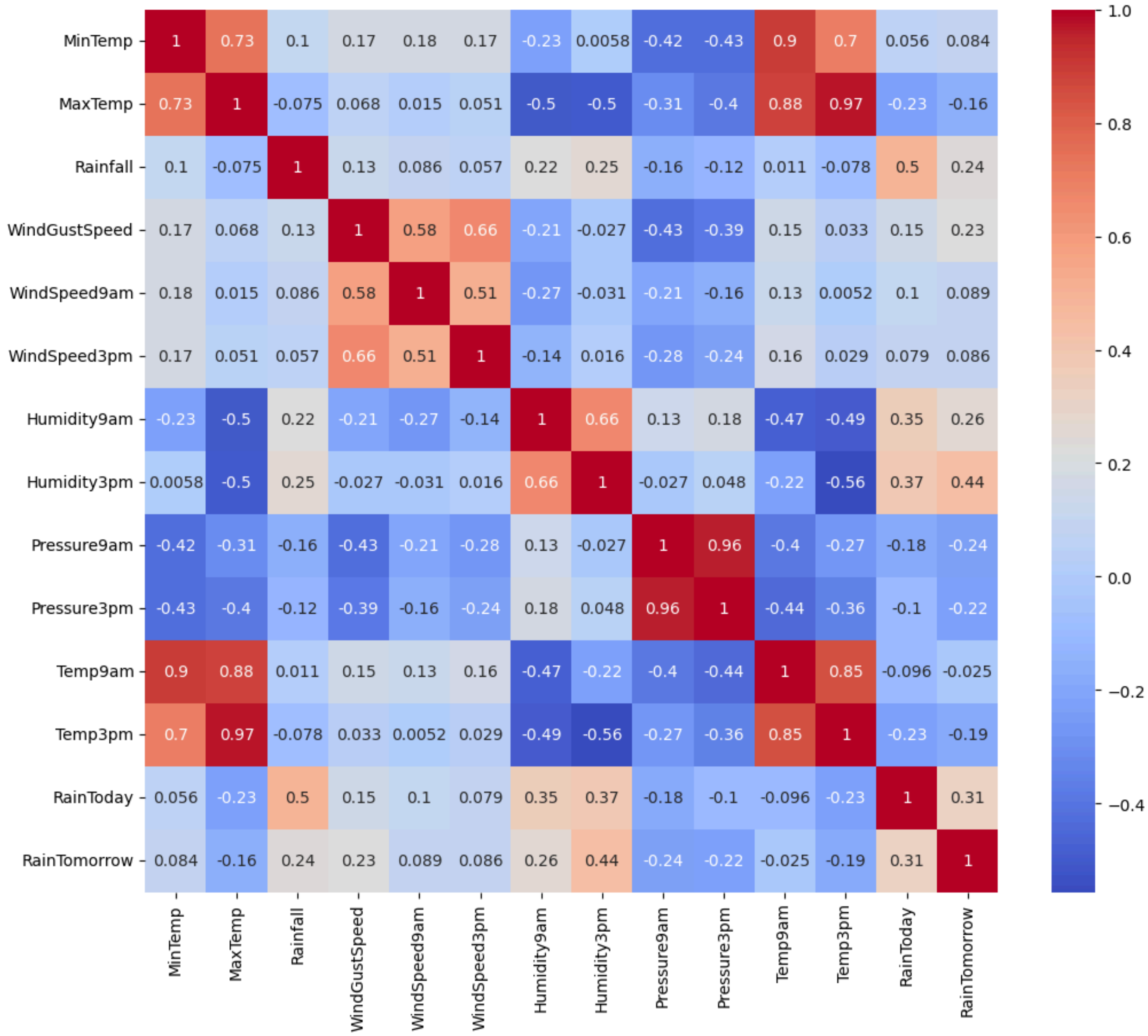
Correlation Heatmap (Feature Relationships)

In [25]:
```
important_cols = [
    'Humidity3pm','Pressure3pm','Rainfall',
    'Temp3pm','RainToday','RainTomorrow'
]

plt.figure(figsize=(8,6))
sns.heatmap(df[important_cols].corr(), annot=True, cmap="coolwarm")
plt.show()
```



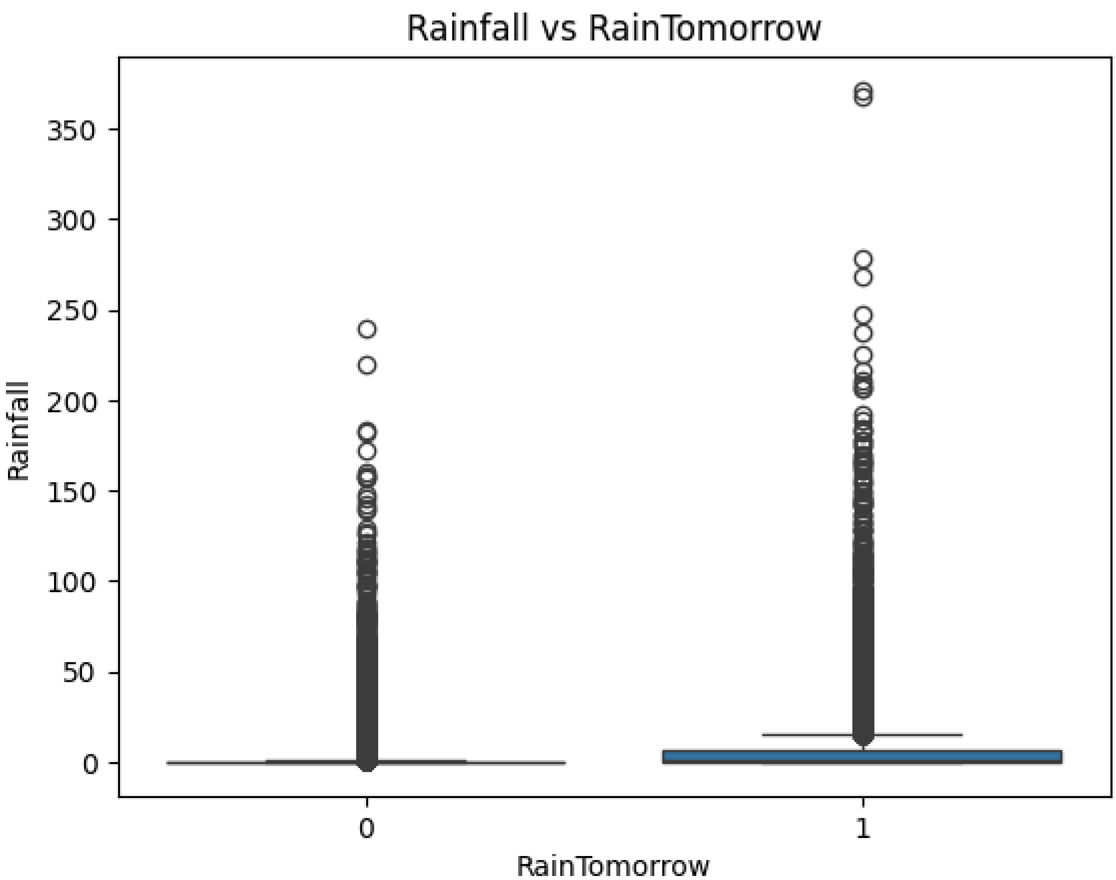
In [28]:

```
num_df = df.select_dtypes(include=['float64','int64'])

plt.figure(figsize=(12,10))
sns.heatmap(num_df.corr(),annot=True, cmap="coolwarm")
plt.show()
```



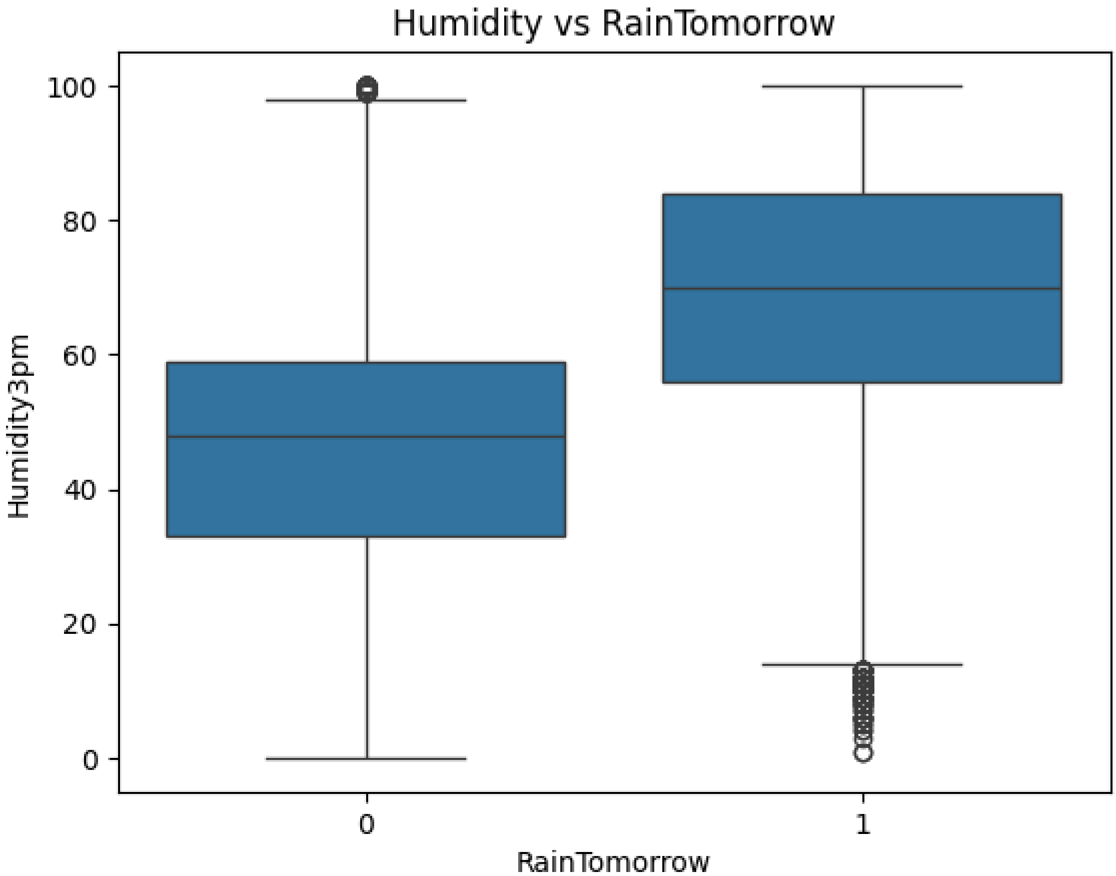
Rainfall vs RainTomorrow

In [20]:
```
sns.boxplot(x='RainTomorrow', y='Rainfall', data=df)
plt.title("Rainfall vs RainTomorrow")
plt.show()
```



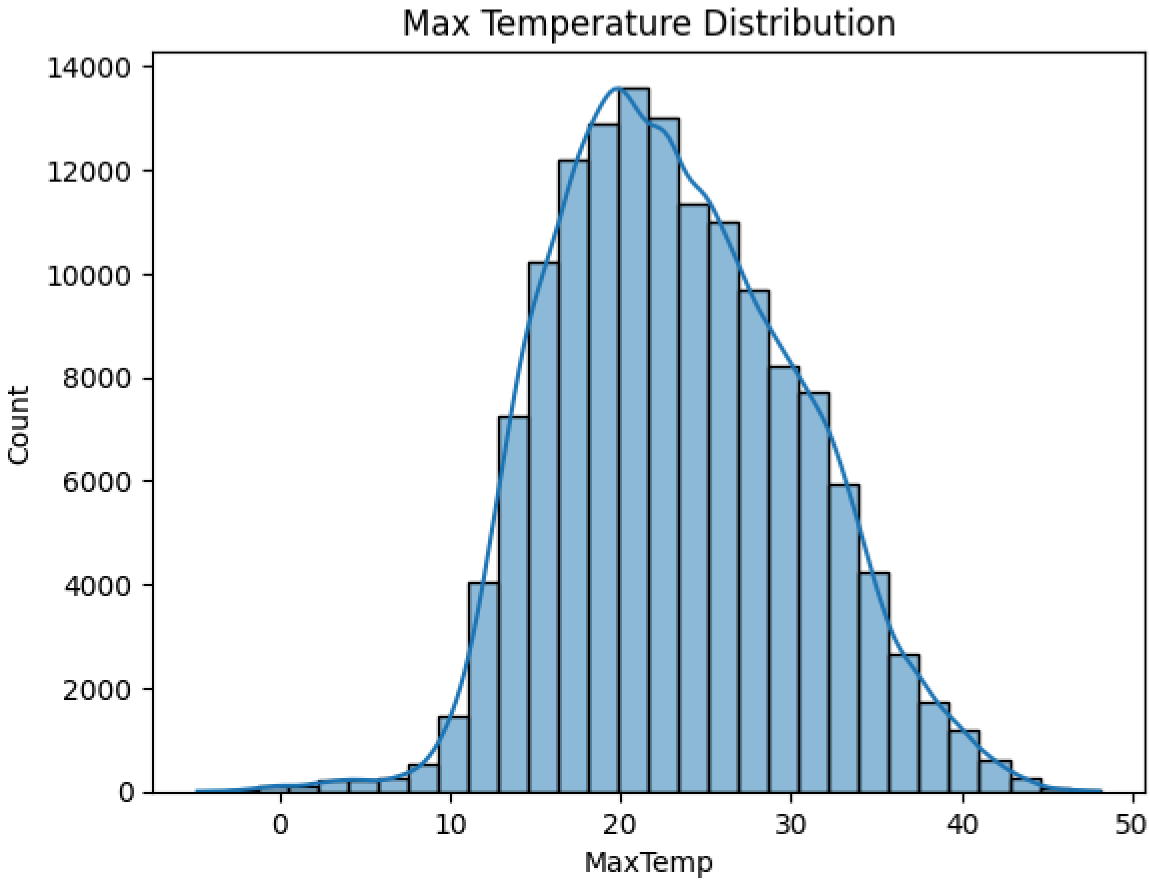
Humidity vs RainTomorrow (Strong Predictor)

In [21]:
```
sns.boxplot(x='RainTomorrow', y='Humidity3pm', data=df)
plt.title("Humidity vs RainTomorrow")
plt.show()
```



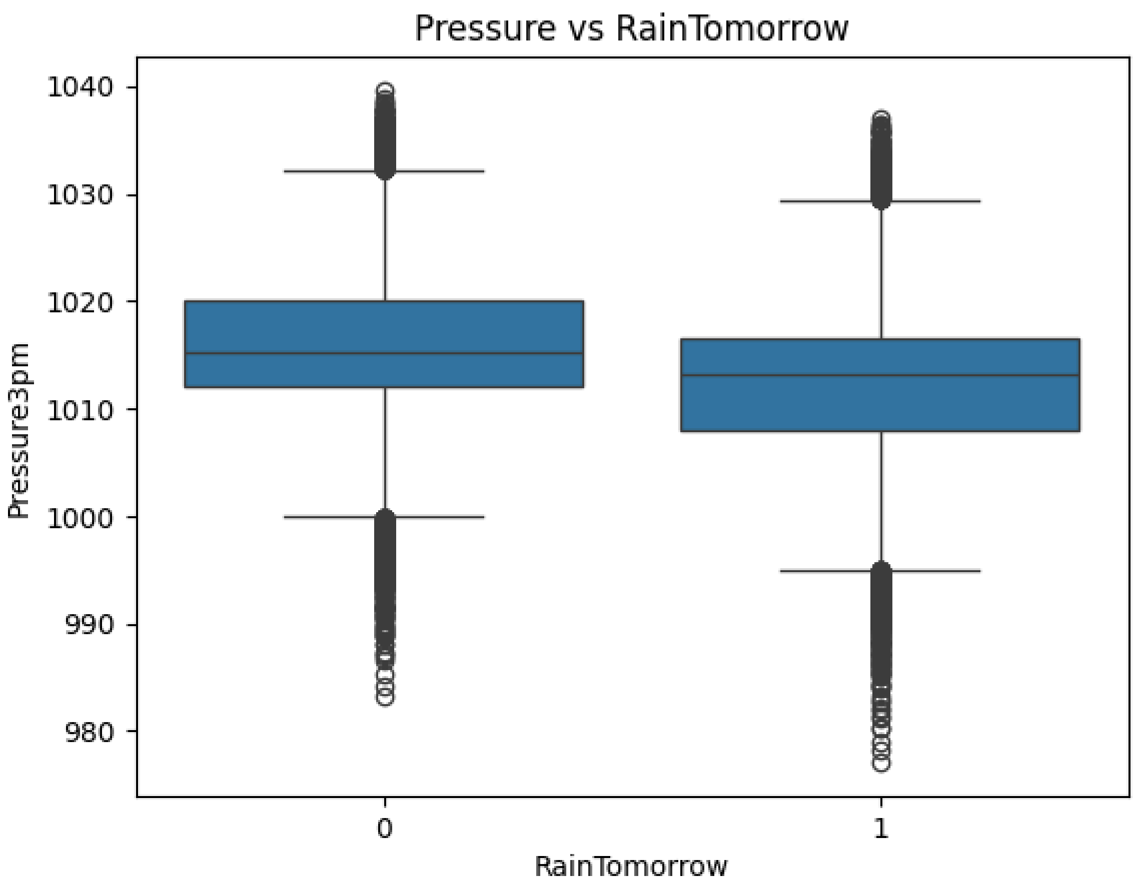
Temperature Distribution

In [22]:
```
sns.histplot(df['MaxTemp'], bins=30, kde=True)
plt.title("Max Temperature Distribution")
plt.show()
```



Pressure vs RainTomorrow

In [30]:
```
sns.boxplot(x='RainTomorrow', y='Pressure3pm', data=df)
plt.title("Pressure vs RainTomorrow")
plt.show()
```



Split Features and Target

In [31]:
```
X = df.drop("RainTomorrow", axis=1)
y = df["RainTomorrow"]
```

In [32]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

In [33]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [34]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model = Sequential()

# Input + Hidden Layer 1
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
```

```
# Hidden Layer 2
model.add(Dense(32, activation='relu'))

# Output Layer
model.add(Dense(1, activation='sigmoid'))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When usi
ng Sequential models, prefer using an `Input(shape)` object as the first layer
in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [35]:

```
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

In [36]:

```
history = model.fit(
    X_train,
    y_train,
    epochs=20,
    batch_size=32,
    validation_split=0.2
)
```

```
Epoch 1/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.8267 - loss: 0.3921 -
val_accuracy: 0.8567 - val_loss: 0.3373
Epoch 2/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8582 - loss: 0.3277 -
val_accuracy: 0.8583 - val_loss: 0.3343
Epoch 3/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8637 - loss: 0.3186 -
val_accuracy: 0.8595 - val_loss: 0.3323
Epoch 4/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8666 - loss: 0.3116 -
val_accuracy: 0.8584 - val_loss: 0.3297
Epoch 5/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8706 - loss: 0.3049 -
val_accuracy: 0.8608 - val_loss: 0.3311
Epoch 6/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8729 - loss: 0.3004 -
val_accuracy: 0.8602 - val_loss: 0.3285
Epoch 7/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - accuracy: 0.8743 - loss: 0.2954 -
val_accuracy: 0.8594 - val_loss: 0.3316
Epoch 8/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8739 - loss: 0.2969 -
val_accuracy: 0.8618 - val_loss: 0.3311
Epoch 9/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8769 - loss: 0.2915 -
val_accuracy: 0.8588 - val_loss: 0.3329
Epoch 10/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8785 - loss: 0.2861 -
val_accuracy: 0.8603 - val_loss: 0.3325
Epoch 11/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8783 - loss: 0.2856 -
val_accuracy: 0.8594 - val_loss: 0.3354
Epoch 12/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8801 - loss: 0.2835 -
val_accuracy: 0.8584 - val_loss: 0.3358
Epoch 13/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8812 - loss: 0.2791 -
val_accuracy: 0.8570 - val_loss: 0.3394
Epoch 14/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8825 - loss: 0.2759 -
val_accuracy: 0.8580 - val_loss: 0.3388
Epoch 15/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8824 - loss: 0.2755 -
val_accuracy: 0.8583 - val_loss: 0.3435
Epoch 16/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8822 - loss: 0.2765 -
val_accuracy: 0.8567 - val_loss: 0.3466
Epoch 17/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8853 - loss: 0.2702 -
val_accuracy: 0.8562 - val_loss: 0.3438
Epoch 18/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8867 - loss: 0.2676 -
val_accuracy: 0.8582 - val_loss: 0.3465
```

```
Epoch 19/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 10s 2ms/step - accuracy: 0.8873 - loss: 0.2654 -
val_accuracy: 0.8547 - val_loss: 0.3511
Epoch 20/20
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.8885 - loss: 0.2657 -
val_accuracy: 0.8537 - val_loss: 0.3527
```

In [37]:
```
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", accuracy)
```

```
880/880 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8502 - loss: 0.3585
Test Accuracy: 0.8519781231880188
```

In [39]:
```
from sklearn.metrics import classification_report

y_pred_proba = model.predict(X_test)
y_pred = (y_pred_proba > 0.5).astype(int)
print(classification_report(y_test, y_pred))
```

```
880/880 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
              precision    recall  f1-score   support

           0       0.88      0.93      0.91     21897
           1       0.71      0.57      0.63      6261

    accuracy                           0.85     28158
   macro avg       0.80      0.75      0.77     28158
weighted avg       0.84      0.85      0.85     28158
```

Why the Problem is Classification and Not Regression

In this project, the objective is to predict whether it will rain the next day using the Australian weather dataset. The target variable **RainTomorrow** contains two possible outcomes: *Yes* or *No*. Since the output is a discrete category rather than a continuous numerical value, the problem is classified as a **binary classification problem**.

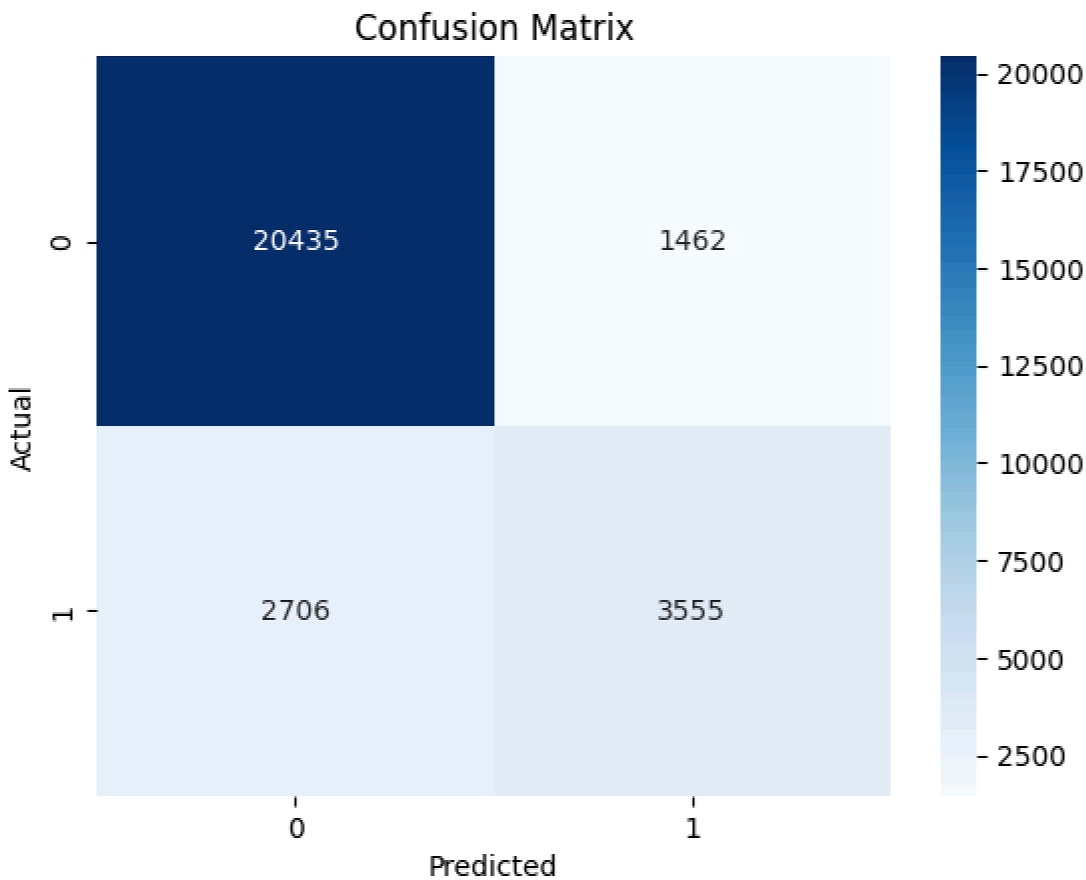
Classification problems involve predicting categorical labels, whereas regression problems involve predicting continuous values such as temperature, rainfall amount, or pressure. In this dataset, the model does not estimate the quantity of rainfall but instead determines the occurrence of rainfall.

Because the target variable is categorical, classification algorithms and evaluation metrics are used. The Artificial Neural Network model uses a sigmoid activation function in the output layer to produce probabilities between 0 and 1, which are then classified into rain or no rain. The model performance is evaluated using classification metrics such as accuracy, confusion matrix, precision, recall, and

F1-score rather than regression metrics like Mean Squared Error (MSE) or R^2 score.

Therefore, rainfall prediction in this project is treated as a classification task instead of a regression task.

In [41]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



In [ ]: