Rain in Australia - Next-Day Prediction Model

Student name: T.J. Kyner \ Student pace: Full time \ Instructor: Abhineet Kulkarni \ Cohort: 040521

Project Overview

Data Source

The data used in this project was downloaded from the Kaggle dataset titled Rain in Australia, which itself was originally sourced from the Australian Bureau of Meteorology's Daily Weather Observations. Additional weather metrics for Australia can be found within the bureau's Climate Data Online web app.

Business Problem

Weather, and humankind's ability to accurately predict it, plays a critical role in many aspects of life. From farmers growing crops to a family planning a weekend vacation to logistical decision making within airlines, rain in particular is highly influential regarding plans. In some instances, the impact of rain can have large financial consequences. As a result, there is a strong interest from a plethora of stakeholders in the ability to accurately forecast rain. The goal of this project is to use the available data to create a next-day prediction model for whether or not it will rain. Such a model could be utilized in a weather app for the benefit of the public at large.

Imports & Settings

In [1]:
```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import plot_confusion_matrix, plot_roc_curve, classification_report
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

import joblib
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Setting the default styling attributes for seaborn
sns.set_theme(style='darkgrid')
```

In [3]:
```
# Loading in the dataset
df = pd.read_csv('weatherAUS.csv')
```

Exploratory Data Analysis

Data Preview

In [4]:
```
df.head()
```

Out[4]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity9am	Humidity3pm	Pressure9am	Pressui
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	71.0	22.0	1007.7	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	44.0	25.0	1010.6	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	38.0	30.0	1007.6	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	45.0	16.0	1017.6	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	82.0	33.0	1010.8	

5 rows × 23 columns

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation',
       'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RainTomorrow'],
      dtype='object')
```

Column Definitions

According to the author of the Kaggle dataset and the "Notes to accompany Daily Weather Observations" published by the Australian Bureau of Meteorology, the meanings and units for each of the columns in the dataset are as follows:

Column Name	Definition	Units
`Date`	Date of the observation	N/A
`Location`	Location of the weather station	N/A
`MinTemp`	Minimum temperature in the 24 hours to 9am. Sometimes only known to the nearest whole degree	Degrees Celsius
`MaxTemp`	Maximum temperature in the 24 hours to 9am. Sometimes only known to the nearest whole degree	Degrees Celsius
`Rainfall`	Precipitation (rainfall) in the 24 hours to 9am. Sometimes only known to the nearest whole millimeter	Millimeters
`Evaporation`	"Class A" pan evaporation in the 24 hours to 9am	Millimeters
`Sunshine`	Bright sunshine in the 24 hours to midnight	Hours
`WindGustDir`	Direction of the strongest wind gust in the 24 hours to midnight	16 compass points
`WindGustSpeed`	Speed of the strongest wind gust in the 24 hours to midnight	Kilometers per hour
`WindDir9am`	Direction of the wind at 9am	16 compass points
`WindDir3pm`	Direction of the wind at 3pm	16 compass points
`WindSpeed9am`	Speed of the wind at 9am	Kilometers per hour
`WindSpeed3pm`	Speed of the wind at 3pm	Kilometers per hour
`Humidity9am`	Relative humidity at 9am	Percent
`Humidity3pm`	Relative humidity at 3pm	Percent
`Pressure9am`	Atmospheric pressure reduced to mean sea level at 9am	Hectopascals
`Pressure3pm`	Atmospheric pressure reduced to mean sea level at 3pm	Hectopascals
`Cloud9am`	Fraction of sky obscured by cloud at 9am	Eighths
`Cloud3pm`	Fraction of sky obscured by cloud at 3pm	Eighths
`Temp9am`	Temparature at 9am	Degrees Celsius
`Temp3pm`	Temparature at 3am	Degrees Celsius
`RainToday`	Did the current day receive precipitation exceeding 1mm in the 24 hours to 9am	Binary (0 = No, 1 = Yes)
`RainTomorrow`	Did the next day receive precipitation exceeding 1mm in the 24 hours to 9am	Binary (0 = No, 1 = Yes)

Exploration

Summary Info and Stats

Taking a look at the dataframe info:

In [6]:
```
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
```

Observations:

- The `Date` column needs converted to a datetime datatype
- The datatypes for all other columns look good as is
- There appears to be a large number of missing values across multiple columns

Looking into the number of missing values per column as a percentage:

In [7]:
```python
round(df.isna().sum() / len(df), 3)
```

Out[7]:
```
Date             0.000
Location         0.000
MinTemp          0.010
MaxTemp          0.009
Rainfall         0.022
Evaporation      0.432
Sunshine         0.480
WindGustDir      0.071
WindGustSpeed    0.071
WindDir9am       0.073
WindDir3pm       0.029
WindSpeed9am     0.012
WindSpeed3pm     0.021
Humidity9am      0.018
Humidity3pm      0.031
Pressure9am      0.104
Pressure3pm      0.103
Cloud9am         0.384
Cloud3pm         0.408
Temp9am          0.012
Temp3pm          0.025
RainToday        0.022
RainTomorrow     0.022
dtype: float64
```

Observations:

- `Evaporation`, `Sunshine`, `Cloud9am`, and `Cloud3pm` are all missing more than 35% of their values
- Aside from `Date` and `Location`, all columns are missing at least some values
- These missing values can be handled by either dropping certain columns/rows, imputing the values, or a mix of both

Next, taking a look at some summary statistics:

In [8]:
```python
df.describe()
```

Out[8]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9a
count	143975.000000	144199.000000	142199.000000	82670.000000	75625.000000	135197.000000	143693.000000	142398.000000	142806.000000	140953.000000	130395.000
mean	12.194034	23.221348	2.360918	5.468232	7.611178	40.035230	14.043426	18.662657	68.880831	51.539116	1017.649
std	6.398495	7.119049	8.478060	4.193704	3.785483	13.607062	8.915375	8.809800	19.029164	20.795902	7.106
min	-8.500000	-4.800000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	0.000000	0.000000	980.500
25%	7.600000	17.900000	0.000000	2.600000	4.800000	31.000000	7.000000	13.000000	57.000000	37.000000	1012.900
50%	12.000000	22.600000	0.000000	4.800000	8.400000	39.000000	13.000000	19.000000	70.000000	52.000000	1017.600
75%	16.900000	28.200000	0.800000	7.400000	10.600000	48.000000	19.000000	24.000000	83.000000	66.000000	1022.400
max	33.900000	48.100000	371.000000	145.000000	14.500000	135.000000	130.000000	87.000000	100.000000	100.000000	1041.000

Observations:

- Multiple columns have clear outliers (e.g., the max `Rainfall` value is 371.0 despite the 75th percentile being 0.8)
- Not seeing any values that are immediate cause for concern (such as a negative value for minimum `Rainfall`)

In order to get a better feel for the data and catch any placeholder values that may not have shown up in the summary statistics, I also want to check the top five most frequent values for each column.

In [9]:
```python
for col in df.columns:
    print('\n')
    print(col)
    print('-'*15)
    print(df[col].value_counts(normalize=True).head())
```

```
Date
---------------
2017-03-25    0.000337
2013-10-05    0.000337
2015-04-29    0.000337
2015-10-31    0.000337
2014-04-27    0.000337
Name: Date, dtype: float64
```

```
Location
---------------
Canberra     0.023622
Sydney       0.022989
Hobart       0.021951
Brisbane     0.021951
Melbourne    0.021951
Name: Location, dtype: float64


MinTemp
---------------
11.0    0.006244
10.2    0.006237
9.6     0.006223
10.5    0.006140
10.8    0.006057
Name: MinTemp, dtype: float64


MaxTemp
---------------
20.0    0.006137
19.0    0.005846
19.8    0.005825
20.4    0.005784
19.9    0.005707
Name: MaxTemp, dtype: float64


Rainfall
---------------
0.0    0.640511
0.2    0.061611
0.4    0.026597
0.6    0.018228
0.8    0.014459
Name: Rainfall, dtype: float64


Evaporation
---------------
4.0    0.040390
8.0    0.031559
2.2    0.025342
2.0    0.024580
2.4    0.024229
Name: Evaporation, dtype: float64


Sunshine
---------------
0.0     0.031193
10.7    0.014559
11.0    0.014466
10.8    0.014136
10.5    0.013580
Name: Sunshine, dtype: float64


WindGustDir
---------------
W      0.073372
SE     0.069694
N      0.068917
SSE    0.068199
E      0.067940
Name: WindGustDir, dtype: float64


WindGustSpeed
---------------
35.0    0.068160
39.0    0.065046
31.0    0.062339
37.0    0.059521
33.0    0.058677
Name: WindGustSpeed, dtype: float64


WindDir9am
---------------
N      0.087165
SE     0.068847
E      0.068024
SSE    0.067549
NW     0.064858
Name: WindDir9am, dtype: float64


WindDir3pm
---------------
SE     0.076739
W      0.071584
S      0.070282
WSW    0.067393
SSE    0.066550
Name: WindDir3pm, dtype: float64
```

```
WindSpeed9am
---------------
9.0     0.094987
13.0    0.091389
11.0    0.081618
17.0    0.075077
7.0     0.075042
Name: WindSpeed9am, dtype: float64


WindSpeed3pm
---------------
13.0    0.088344
17.0    0.088056
20.0    0.082255
15.0    0.080640
19.0    0.079095
Name: WindSpeed3pm, dtype: float64


Humidity9am
---------------
99.0    0.023746
70.0    0.021190
69.0    0.021169
65.0    0.021106
68.0    0.021085
Name: Humidity9am, dtype: float64


Humidity3pm
---------------
52.0    0.019517
55.0    0.019425
57.0    0.019354
53.0    0.019134
59.0    0.019084
Name: Humidity3pm, dtype: float64


Pressure9am
---------------
1016.4    0.006258
1017.9    0.006051
1016.3    0.005943
1018.7    0.005943
1017.3    0.005897
Name: Pressure9am, dtype: float64


Pressure3pm
---------------
1015.3    0.006026
1015.5    0.006003
1015.6    0.005949
1015.7    0.005926
1013.5    0.005880
Name: Pressure3pm, dtype: float64


Cloud9am
---------------
7.0    0.222971
1.0    0.175133
8.0    0.164080
0.0    0.096481
6.0    0.091223
Name: Cloud9am, dtype: float64


Cloud3pm
---------------
7.0    0.211714
1.0    0.173933
8.0    0.147035
6.0    0.104272
2.0    0.083924
Name: Cloud3pm, dtype: float64


Temp9am
---------------
17.0    0.006347
13.8    0.006263
14.8    0.006222
16.0    0.006138
14.0    0.006096
Name: Temp9am, dtype: float64


Temp3pm
---------------
20.0    0.006218
19.0    0.006126
18.5    0.006126
18.4    0.006119
17.8    0.006056
Name: Temp3pm, dtype: float64
```

```
RainToday
---------------
No     0.775807
Yes    0.224193
Name: RainToday, dtype: float64


RainTomorrow
---------------
No     0.775819
Yes    0.224181
Name: RainTomorrow, dtype: float64
```

Observations:

- The value counts of the `Date` column need further explored on a non-normalized basis
- There's a disconnect between the `Rainfall` value counts and the `RainToday` / `RainTomorrow` value counts. While roughly 64% of observations had a value of 0 for `Rainfall`, about 77.5% of days did not have rainfall according to the latter two columns. This discrepency is likely due to differences in the number of missing values for each column
- The `RainToday` and `RainTomorrow` columns should be converted to 0s and 1s for easier manipulation

Further exploring the `Date` column:

In [10]:
```
df.Date.value_counts()
```

Out[10]:
```
2017-03-25    49
2013-10-05    49
2015-04-29    49
2015-10-31    49
2014-04-27    49
              ..
2007-11-10     1
2008-01-09     1
2008-01-30     1
2008-01-15     1
2007-11-16     1
Name: Date, Length: 3436, dtype: int64
```

In [11]:
```
df.Location.nunique()
```

Out[11]:
```
49
```

The maximum number of observations for a given date aligns with the number of unique locations within the dataset. This intuitively makes sense because each weather station at the different locations would be reporting their own data for a given day.

Adjusting the `RainToday` and `RainTomorrow` columns:

In [12]:
```
df.RainToday = df.RainToday.map({'No': 0, 'Yes': 1})
df.RainToday.value_counts(normalize=True)
```

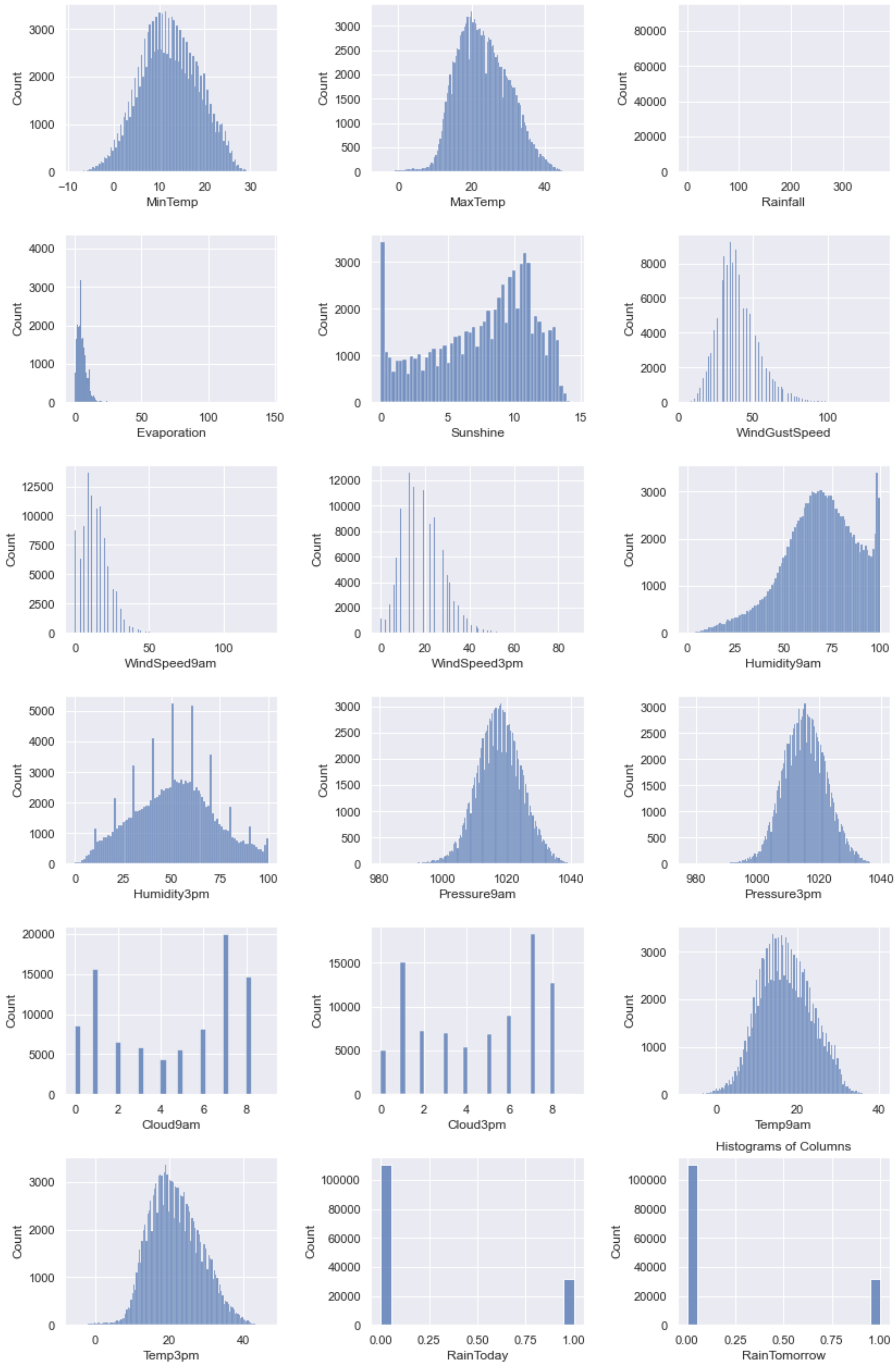
Out[12]:
```
0.0    0.775807
1.0    0.224193
Name: RainToday, dtype: float64
```

In [13]:
```
df.RainTomorrow = df.RainTomorrow.map({'No': 0, 'Yes': 1})
df.RainTomorrow.value_counts(normalize=True)
```

Out[13]:
```
0.0    0.775819
1.0    0.224181
Name: RainTomorrow, dtype: float64
```

Histograms

In [89]:
```
fig, axes = plt.subplots(nrows=6, ncols=3, figsize=(12, 18))
axes = axes.reshape(-1)

continuous = [col for col in df.columns if df[col].dtype != object]
for i, col in enumerate(continuous):
    sns.histplot(df[col], ax=axes[i])

fig.tight_layout(pad=2.0)
plt.title('Histograms of Columns')
plt.savefig('images/histograms.png', facecolor='white', dpi=100);
```



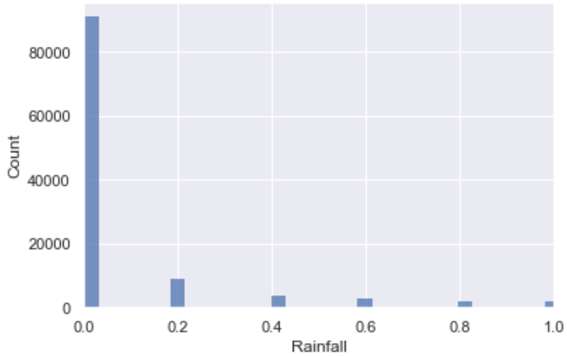
Observations:

- Most features are normally distributed as expected
- The `Rainfall` distribution needs further investigation as the large outlier is likely affecting the ability to plot the data
- The `Sunshine` distribution is interesting but largely explainable:
  - The high frequency of 0 values represents days where it is overcast all day
  - The abrupt decline in frequency after around 11 hours is a reflection of the limited number of days of the year where it is light out for that many hours or longer
- The `Humidity9am` distribution is particularly interesting due to the large spike in frequencies near 100%

Since the summary statistics section showed that the 75th percentile for the `Rainfall` feature is only 0.8, the following plot shows the distribution of values betwen 0 and 1.

In [16]:

```
sns.histplot(df.Rainfall)
plt.xlim(0, 1);
```



Rainy Days by Location

In [17]:

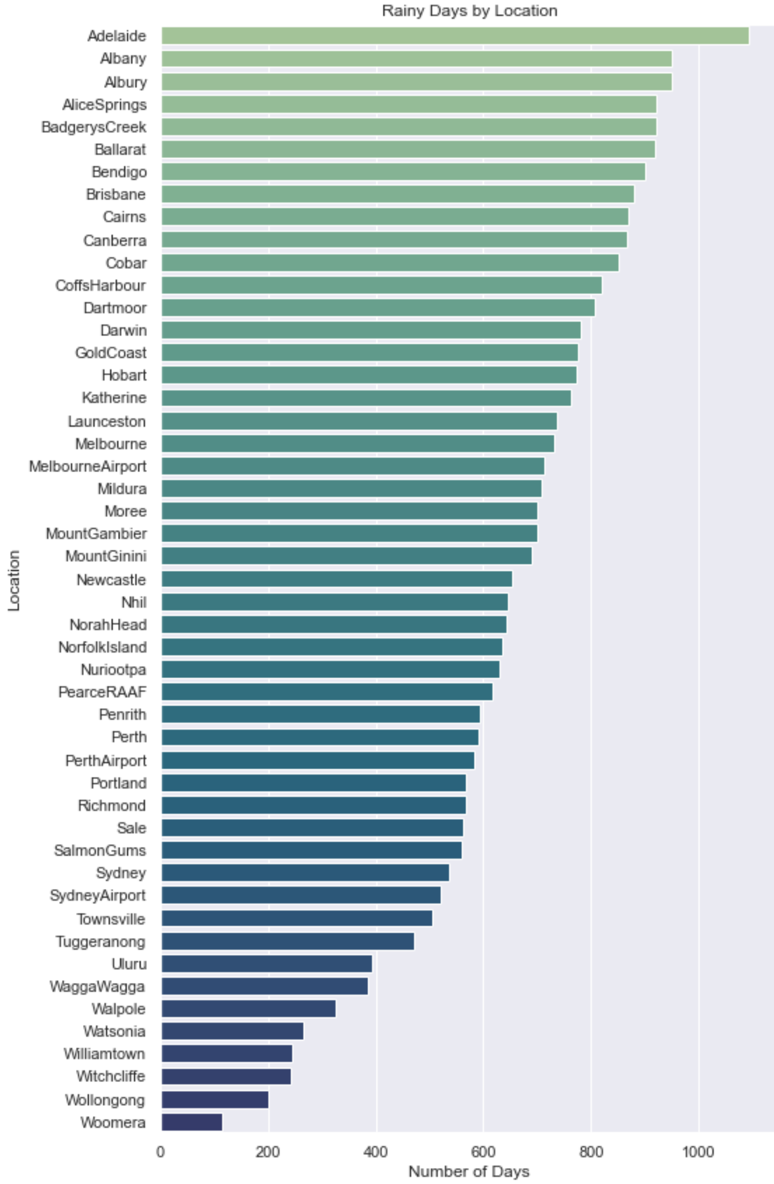
```
df_rain_by_loc = df.groupby(by='Location').sum()
df_rain_by_loc = df_rain_by_loc[['RainToday']]
df_rain_by_loc.head()
```

Out[17]:

Location	RainToday
Adelaide	689.0
Albany	902.0
Albury	617.0
AliceSprings	244.0
BadgerysCreek	583.0

In [88]:

```
plt.figure(figsize=(8, 12))
sns.barplot(x='RainToday',
            y=df_rain_by_loc.index,
            data=df_rain_by_loc.sort_values('RainToday', ascending=False),
            orient='h',
            palette='crest'
           )
plt.xlabel('Number of Days')
plt.title('Rainy Days by Location')
plt.tight_layout()
plt.savefig('images/rainy_days_by_loc.png', facecolor='white', dpi=100);
```



The above chart is useful for a quick check on the differences between locations with regard to the number of rainy days but suffers from one key issue: the number of observations from each location is not exactly the same. Checking the value counts for each location (below) reveals that the locations of `Katherine`, `Nhil`, and `Uluru` should be ignored when analyzing the above plot. The remaining locations have value counts that are close enough to be properly comparable.

In [18]:
```
df.Location.value_counts()
```

Out[18]:
```
Canberra            3436
Sydney              3344
Darwin              3193
Melbourne           3193
Adelaide            3193
Brisbane            3193
Perth               3193
Hobart              3193
Launceston          3040
Bendigo             3040
Cairns              3040
MountGinini         3040
Ballarat            3040
Albury              3040
MountGambier        3040
Albany              3040
Townsville          3040
AliceSprings        3040
Wollongong          3040
GoldCoast           3040
Penrith             3039
Newcastle           3039
Tuggeranong         3039
Mildura             3009
Woomera             3009
Witchcliffe         3009
Watsonia            3009
Cobar               3009
CoffsHarbour        3009
Dartmoor            3009
Williamtown         3009
Portland            3009
MelbourneAirport    3009
WaggaWagga          3009
NorfolkIsland       3009
```

```
PearceRAAF          3009
Sale                3009
Richmond            3009
Moree               3009
PerthAirport        3009
Nuriootpa           3009
BadgerysCreek       3009
SydneyAirport       3009
Walpole             3006
NorahHead           3004
SalmonGums          3001
Katherine           1578
Nhil                1578
Uluru               1578
Name: Location, dtype: int64
```

Seasonality

Rainfall exhibits seasonality in many areas of the world. Through grouping the data by month of the year, the percentage of days that it rains in a given month can be easily calculated. Any sort of trend would indicate that the month of the year is a valuable piece of information for modeling purposes.

In [18]:

```
df_seasonality = df.copy()
df_seasonality['month'] = df_seasonality.Date.apply(lambda x: int(str(x)[5:7]))
df_seasonality[['Date', 'month']].head()
```

Out[18]:

	Date	month
0	2008-12-01	12
1	2008-12-02	12
2	2008-12-03	12
3	2008-12-04	12
4	2008-12-05	12

In [19]:

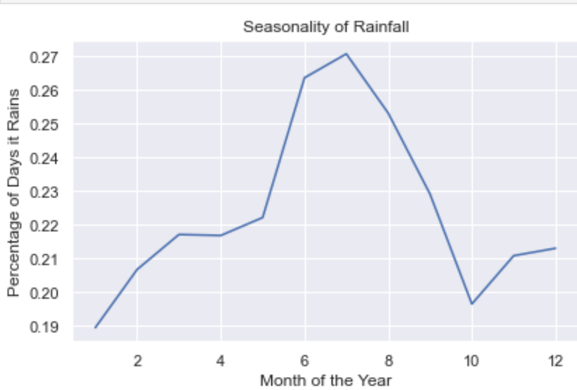
```
df_seasonality_grouped = df_seasonality.groupby('month').mean()
df_seasonality_grouped[['RainToday']]
```

Out[19]:

	RainToday
month	
1	0.189484
2	0.206746
3	0.217135
4	0.216845
5	0.222163
6	0.263638
7	0.270736
8	0.253167
9	0.229135
10	0.196512
11	0.210843
12	0.213037

In [86]:

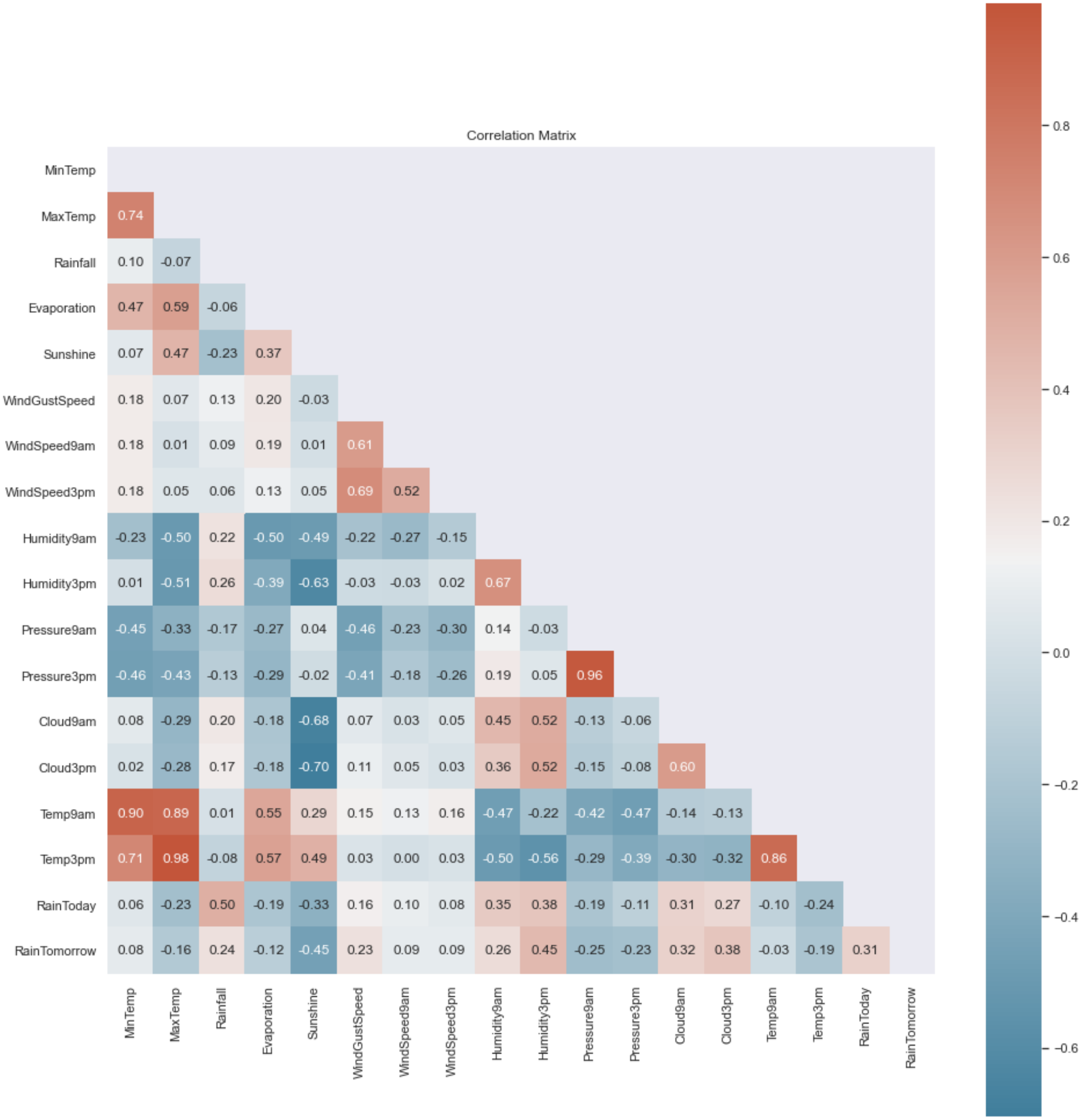
```
sns.lineplot(data=df_seasonality_grouped, x=df_seasonality_grouped.index, y='RainToday')
plt.title('Seasonality of Rainfall')
plt.xlabel('Month of the Year')
plt.ylabel('Percentage of Days it Rains')
plt.tight_layout()
plt.savefig('images/seasonality.png', facecolor='white', dpi=100);
```



Rainfall in Australia clearly has a degree of seasonality.

Correlation Matrix

In [83]:
```
plt.figure(figsize=(14, 14))
plt.title('Correlation Matrix')

# Creating a mask to block the top right half of the heatmap (redundant information)
mask = np.triu(np.ones_like(df.corr()))

# Custom color map
cmap = sns.diverging_palette(230, 20, as_cmap=True)

sns.heatmap(df.corr(), mask=mask, annot=True, fmt='.2f', square=True, cmap=cmap)
plt.tight_layout()
plt.savefig('images/corr_heatmap.png', facecolor='white', dpi=100);
```



Observations:

- Nothing in this correlation heatmap is surprising
- Features with strong correlations (either positive or negative) have intuitive reasons for being so

Data Preprocessing

Missing Values

The primary preprocessing need for this dataset is handling the missing values. Given the strong correlations between certain features, using a multivariate feature imputation method makes sense. While still experimental, the `IterativeImputer` module from `sklearn` is perfect for this use case and appears stable enough. This module...

> "...models each feature with missing values as a function of other features, and uses that estimate for imputation. It does so in an iterated round-robin fashion: at each step, a feature column is designated as output y and the other feature columns are treated as inputs X. A regressor is fit on (X, y) for known y. Then, the regressor is used to predict the missing values of y. This is done for each feature in an iterative fashion, and then is repeated for max_iter imputation rounds. The results of the final imputation round are returned."

Source: 6.4.3. Multivariate feature imputation

I do not want to impute values for the target variable (`RainTomorrow`) since this will detract from the ground truth and have potential negative effects on the model. To start, I'll drop rows in which the `RainTomorrow` value is missing.

In [23]:
```
df_imputed = df.dropna(axis=0, subset=['RainTomorrow'])
df_imputed.isna().sum()
```

Out[23]:
```
Date                0
Location            0
MinTemp           637
MaxTemp           322
Rainfall         1406
Evaporation     60843
Sunshine        67816
WindGustDir      9330
WindGustSpeed    9270
WindDir9am      10013
WindDir3pm       3778
WindSpeed9am     1348
WindSpeed3pm     2630
Humidity9am      1774
Humidity3pm      3610
Pressure9am     14014
Pressure3pm     13981
Cloud9am        53657
Cloud3pm        57094
Temp9am           904
Temp3pm          2726
RainToday        1406
RainTomorrow        0
dtype: int64
```

Continuous Features

For the continuous features, I'll apply the `IterativeImputer`.

In [24]:
```
cont_feats = [col for col in df_imputed.columns if df_imputed[col].dtype != object]
cont_feats.remove('RainTomorrow')
cont_feats
```

Out[24]:
```
['MinTemp',
 'MaxTemp',
 'Rainfall',
 'Evaporation',
 'Sunshine',
 'WindGustSpeed',
 'WindSpeed9am',
 'WindSpeed3pm',
 'Humidity9am',
 'Humidity3pm',
 'Pressure9am',
 'Pressure3pm',
 'Cloud9am',
 'Cloud3pm',
 'Temp9am',
 'Temp3pm',
 'RainToday']
```

In [25]:
```
imputer = IterativeImputer(random_state=42)
df_imputed_cont = imputer.fit_transform(df_imputed[cont_feats])
df_imputed_cont = pd.DataFrame(df_imputed_cont, columns=cont_feats)
df_imputed_cont.head()
```

Out[25]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Cloud9
0	13.4	22.9	0.6	6.497888	7.048211	44.0	20.0	24.0	71.0	22.0	1007.7	1007.1	8.000
1	7.4	25.1	0.0	6.270412	10.863393	44.0	4.0	22.0	44.0	25.0	1010.6	1007.8	1.912
2	12.9	25.7	0.0	8.659380	11.812408	46.0	19.0	26.0	38.0	30.0	1007.6	1008.7	2.014
3	9.2	28.0	0.0	6.764941	11.542532	24.0	11.0	9.0	45.0	16.0	1017.6	1012.8	1.201
4	17.5	32.3	1.0	7.455971	5.520080	41.0	7.0	20.0	82.0	33.0	1010.8	1006.0	7.000

In [26]:
```
df_imputed_cont.isna().sum()
```

Out[26]:
```
MinTemp          0
MaxTemp          0
Rainfall         0
Evaporation      0
Sunshine         0
WindGustSpeed    0
WindSpeed9am     0
```

```
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Cloud9am         0
Cloud3pm         0
Temp9am          0
Temp3pm          0
RainToday        0
dtype: int64
```

Categorical Features

For the categorical features, I'll be replacing the missing values with a randomly chosen option from the unique values of each feature according to their probability distribution.

In [27]:
```
cat_feats = [col for col in df_imputed.columns if col not in cont_feats]
cat_feats.remove('RainTomorrow')

# Also removing Date and Location since no values are missing
cat_feats.remove('Date')
cat_feats.remove('Location')
cat_feats
```

Out[27]:
```
['WindGustDir', 'WindDir9am', 'WindDir3pm']
```

In [28]:
```
df_imputed_cat = df_imputed[cat_feats]

for col in df_imputed_cat.columns:
    values = df_imputed_cat.WindDir3pm.value_counts().reset_index()['index'].values
    probs = df_imputed_cat[col].value_counts(normalize=True).values
    df_imputed_cat[col].replace(np.nan, np.random.choice(a=values, p=probs), inplace=True)

df_imputed_cat.head()
```

Out[28]:

	WindGustDir	WindDir9am	WindDir3pm
0	W	W	WNW
1	WNW	NNW	WSW
2	WSW	W	WSW
3	NE	SE	E
4	W	ENE	NW

In [29]:
```
df_imputed_cat.isna().sum()
```

Out[29]:
```
WindGustDir    0
WindDir9am     0
WindDir3pm     0
dtype: int64
```

Concatenating

Now that the missing values have been handled, I need to place all of the separated dataframes back together into one final dataframe.

In [30]:
```
df_date_loc = df_imputed[['Date', 'Location']]
df_target = df_imputed.RainTomorrow

print(df_date_loc.shape)
print(df_imputed_cont.shape)
print(df_imputed_cat.shape)
print(df_target.shape)
```

```
(142193, 2)
(142193, 17)
(142193, 3)
(142193,)
```

In [31]:
```
df_imputed_final = pd.concat(objs=[df_date_loc.reset_index(drop=True),
                                   df_imputed_cont.reset_index(drop=True),
                                   df_imputed_cat.reset_index(drop=True),
                                   df_target.reset_index(drop=True)
                                  ],
                             axis=1
                            )
df_imputed_final.shape
```

Out[31]:
```
(142193, 23)
```

In [32]:
```
df_imputed_final.head()
```

Out[32]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	...	Pressure3pm	Cloud9am	Cloud3pm	Temp9
0	2008-12-01	Albury	13.4	22.9	0.6	6.497888	7.048211	44.0	20.0	24.0	...	1007.1	8.000000	5.103048	1
1	2008-12-02	Albury	7.4	25.1	0.0	6.270412	10.863393	44.0	4.0	22.0	...	1007.8	1.912027	2.640581	1

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	...	Pressure3pm	Cloud9am	Cloud3pm	Temp9
2	2008-12-03	Albury	12.9	25.7	0.0	8.659380	11.812408	46.0	19.0	26.0	...	1008.7	2.014404	2.000000	2
3	2008-12-04	Albury	9.2	28.0	0.0	6.764941	11.542532	24.0	11.0	9.0	...	1012.8	1.201990	1.993914	1
4	2008-12-05	Albury	17.5	32.3	1.0	7.455971	5.520080	41.0	7.0	20.0	...	1006.0	7.000000	8.000000	1

5 rows × 23 columns

A quick check to ensure all missing values have been handled:

In [33]:
```
df_imputed_final.isna().sum()
```

Out[33]:
```
Date             0
Location         0
MinTemp          0
MaxTemp          0
Rainfall         0
Evaporation      0
Sunshine         0
WindGustSpeed    0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Cloud9am         0
Cloud3pm         0
Temp9am          0
Temp3pm          0
RainToday        0
WindGustDir      0
WindDir9am       0
WindDir3pm       0
RainTomorrow     0
dtype: int64
```

Extracting the Month

As seen in the EDA section, rainfall in Australia exhibits seasonality. Instead of using the full date from the `Date` column, extracting just the month is much more valuable.

In [34]:
```
df_month = df_imputed_final.copy()
df_month.insert(1, 'Month', df_month.Date.apply(lambda x: int(str(x)[5:7])))
df_month.drop(columns='Date', inplace=True)
df_month.head()
```

Out[34]:

	Month	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	...	Pressure3pm	Cloud9am	Cloud3pm	Temp
0	12	Albury	13.4	22.9	0.6	6.497888	7.048211	44.0	20.0	24.0	...	1007.1	8.000000	5.103048	
1	12	Albury	7.4	25.1	0.0	6.270412	10.863393	44.0	4.0	22.0	...	1007.8	1.912027	2.640581	
2	12	Albury	12.9	25.7	0.0	8.659380	11.812408	46.0	19.0	26.0	...	1008.7	2.014404	2.000000	
3	12	Albury	9.2	28.0	0.0	6.764941	11.542532	24.0	11.0	9.0	...	1012.8	1.201990	1.993914	
4	12	Albury	17.5	32.3	1.0	7.455971	5.520080	41.0	7.0	20.0	...	1006.0	7.000000	8.000000	

5 rows × 23 columns

Dummy Variables

All categorical features now need transformed into dummy variables in order to be useable in the modeling section.

In [35]:
```
categoricals = ['Month', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm']
df_dummies = pd.get_dummies(df_month, columns=categoricals)
df_dummies.head()
```

Out[35]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	...	WindDir3pm_NNW	WindDir3pr
0	13.4	22.9	0.6	6.497888	7.048211	44.0	20.0	24.0	71.0	22.0	...	0	
1	7.4	25.1	0.0	6.270412	10.863393	44.0	4.0	22.0	44.0	25.0	...	0	
2	12.9	25.7	0.0	8.659380	11.812408	46.0	19.0	26.0	38.0	30.0	...	0	
3	9.2	28.0	0.0	6.764941	11.542532	24.0	11.0	9.0	45.0	16.0	...	0	
4	17.5	32.3	1.0	7.455971	5.520080	41.0	7.0	20.0	82.0	33.0	...	0	

5 rows × 127 columns

```
In [36]: df_dummies.columns
```

```
Out[36]: Index(['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine',
        'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am',
        'Humidity3pm',
        ...
        'WindDir3pm_NNW', 'WindDir3pm_NW', 'WindDir3pm_S', 'WindDir3pm_SE',
        'WindDir3pm_SSE', 'WindDir3pm_SSW', 'WindDir3pm_SW', 'WindDir3pm_W',
        'WindDir3pm_WNW', 'WindDir3pm_WSW'],
       dtype='object', length=127)
```

Modeling

```
In [37]: df_final = df_dummies.copy()
X = df_final.drop(columns='RainTomorrow')
y = df_final.RainTomorrow

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

print('Train size:', X_train.shape[0])
print('Test size: ', X_test.shape[0])
```

```
Train size: 106644
Test size:  35549
```

Logistic Regression

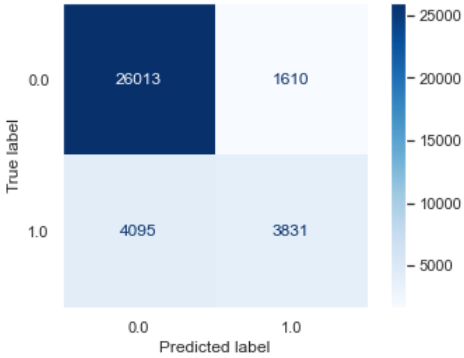
Baseline

```
In [38]: logreg = LogisticRegression(random_state=42)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
y_pred
```

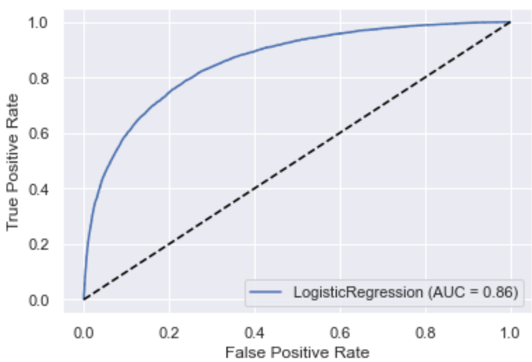
```
Out[38]: array([1., 0., 0., ..., 0., 1., 0.])
```

```
In [39]: def conf_matrix(model, X_test, y_test, cmap='Blues'):
    plot_confusion_matrix(model, X_test, y_test, cmap=cmap)
    plt.grid()
    plt.show()

def roc_curve_custom(model, X_test, y_test):
    plot_roc_curve(model, X_test, y_test)
    plt.plot([0, 1], [0, 1], color='black', linestyle='--')
    plt.show()

def evaluate(model, X_train=X_train, X_test=X_test, y_train=y_train, y_test=y_test, y_pred=y_pred):
    # Confusion Matrix
    print('Confusion Matrix')
    print('-'*53)
    conf_matrix(model, X_test, y_test)
    print('\n')

    # Classification Report
    print('Classification Report')
    print('-'*53)
    print(classification_report(y_test, y_pred))
    print('\n')

    # ROC Curve
    print('ROC Curve')
    print('-'*53)
    roc_curve_custom(model, X_test, y_test)
    print('\n')

    # Checking model fitness
    print('Checking model fitness')
    print('-'*53)
    print('Train score:', round(model.score(X_train, y_train), 4))
    print('Test score: ', round(model.score(X_test, y_test), 4))
    print('\n')

evaluate(logreg)
```

```
Confusion Matrix
-----------------------------------------------------
```




```
Classification Report
-----------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.86      0.94      0.90     27623
         1.0       0.70      0.48      0.57      7926

    accuracy                           0.84     35549
   macro avg       0.78      0.71      0.74     35549
weighted avg       0.83      0.84      0.83     35549


ROC Curve
-----------------------------------------------------
```




```
Checking model fitness
-----------------------------------------------------
Train score: 0.8426
Test score:  0.8395
```

Observations:

- Decent performance for a baseline model
- Recall is the weakest point, particularly for days where it *does* rain tomorrow
- The model is well fit, with both the train and test scores approximately the same

Correcting Class Imbalance

A class imbalance currently exists for the target variable. Correcting for this may help improve model performance. To do so, I will resample the training data using `SMOTE`.

In [40]:
```
X_train_resampled, y_train_resampled = SMOTE().fit_resample(X_train, y_train)

print('Original')
print('-'*20)
print(y_train.value_counts())
print('\n')
print('SMOTE')
print('-'*20)
print(pd.Series(y_train_resampled).value_counts())
```

```
Original
--------------------
0.0    82693
1.0    23951
Name: RainTomorrow, dtype: int64


SMOTE
--------------------
1.0    82693
0.0    82693
Name: RainTomorrow, dtype: int64
```

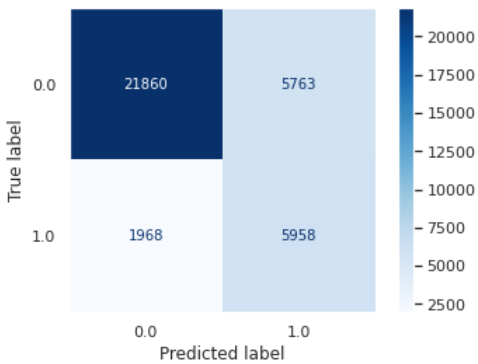
In [102…
```
logreg_smote = LogisticRegression(random_state=42)
logreg_smote.fit(X_train_resampled, y_train_resampled)
```

```
y_pred_smote = logreg_smote.predict(X_test)
y_pred_smote
```

Out[102… array([1., 1., 0., ..., 0., 1., 1.])

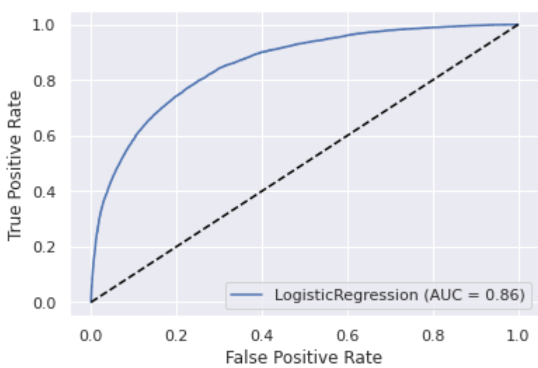
In [103…
```
evaluate(logreg_smote, X_train=X_train_resampled, y_train=y_train_resampled, y_pred=y_pred_smote)
```

```
Confusion Matrix
-----------------------------------------------------
```




```
Classification Report
-----------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.92      0.79      0.85     27623
         1.0       0.51      0.75      0.61      7926

    accuracy                           0.78     35549
   macro avg       0.71      0.77      0.73     35549
weighted avg       0.83      0.78      0.80     35549
```

```
ROC Curve
-----------------------------------------------------
```




```
Checking model fitness
-----------------------------------------------------
Train score: 0.7849
Test score:  0.7825
```

Observations:

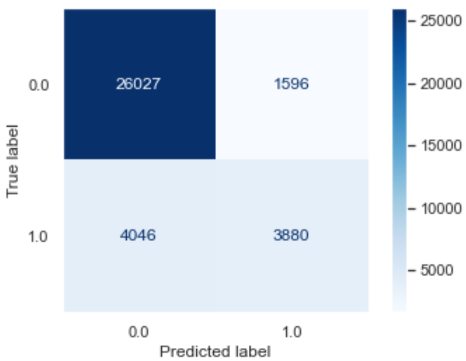
- Despite a slight increase in the positive F1 score, the accuracy of this model sharply decreased
- This model remains well fit but scores for both the train and test sets decreased
- Contrary to my initial thoughts, using `SMOTE` actually had worse performance and will not be utilized in subsequent iterations

Hyperparameter Tuning

In [93]:
```
logreg_params = {
    'C': [1, 1e8, 1e16],
    'fit_intercept': [True, False],
    'max_iter': [50, 100, 150],
    'random_state': [42]
}

logreg_gs = GridSearchCV(logreg, logreg_params, scoring='accuracy', cv=3)
logreg_gs.fit(X_train, y_train)
```

Out[93]:
```
GridSearchCV(cv=3, estimator=LogisticRegression(C=1e+16, random_state=42),
             param_grid={'C': [1, 100000000.0, 1e+16],
                         'fit_intercept': [True, False],
                         'max_iter': [50, 100, 150], 'random_state': [42]},
             scoring='accuracy')
```

Due to the amount of time it takes to run the grid search, I'll be using the `joblib` library to save it to a file for easy access without rerunning the entire training process again.

In [94]:
```
joblib.dump(logreg_gs, 'saved_models/logreg_gs.joblib')
```
Out[94]:
```
['saved_models/logreg_gs.joblib']
```
In [40]:
```
logreg_gs = joblib.load('saved_models/logreg_gs.joblib')
```
In [41]:
```
logreg_gs.best_params_
```
Out[41]:
```
{'C': 1, 'fit_intercept': False, 'max_iter': 150, 'random_state': 42}
```
In [42]:
```
round(logreg_gs.best_score_, 4)
```
Out[42]:
```
0.8433
```
In [43]:
```
y_pred_logreg_gs = logreg_gs.predict(X_test)
y_pred_logreg_gs
```
Out[43]:
```
array([1., 1., 0., ..., 0., 1., 0.])
```
In [44]:
```
evaluate(logreg_gs, y_pred=y_pred_logreg_gs)
```
```
Confusion Matrix
-----------------------------------------------------
```



```
Classification Report
-----------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.87      0.94      0.90     27623
         1.0       0.71      0.49      0.58      7926

    accuracy                           0.84     35549
   macro avg       0.79      0.72      0.74     35549
weighted avg       0.83      0.84      0.83     35549


ROC Curve
-----------------------------------------------------
```



```
Checking model fitness
-----------------------------------------------------
Train score: 0.8441
Test score:  0.8413
```

Observations:

- Slight improvements in precision and model fitness
- Overall, not much improvement over the baseline logreg model

Decision Tree

Baseline

In [43]:
```
clf = DecisionTreeClassifier(random_state=42)
```

```
clf.fit(X_train, y_train)
y_pred_tree = clf.predict(X_test)
y_pred_tree
```

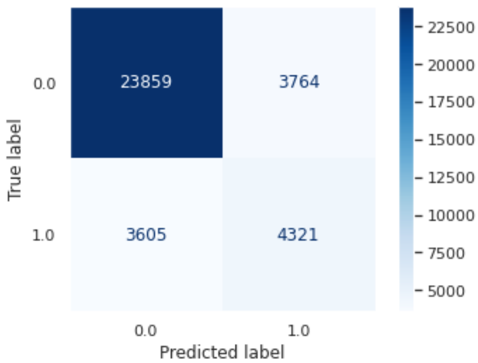
Out[43]:

```
array([1., 1., 0., ..., 0., 0., 0.])
```

In [44]:

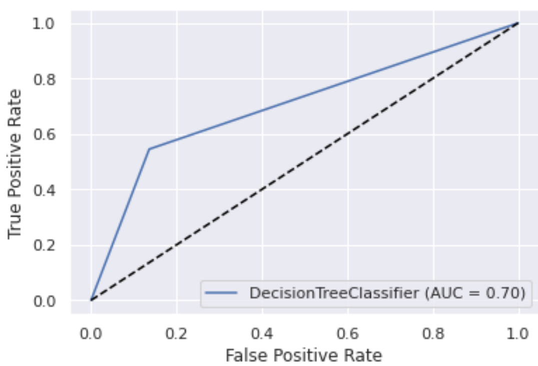
```
evaluate(clf, y_pred=y_pred_tree)
```

```
Confusion Matrix
-----------------------------------------------------
```




```
Classification Report
-----------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.87      0.86      0.87     27623
         1.0       0.53      0.55      0.54      7926

    accuracy                           0.79     35549
   macro avg       0.70      0.70      0.70     35549
weighted avg       0.79      0.79      0.79     35549
```

```
ROC Curve
-----------------------------------------------------
```




```
Checking model fitness
-----------------------------------------------------
Train score: 1.0
Test score:  0.7927
```

Observations:

- The accuracy is lower than the tuned logisitic regression model
- The model is overfit, given by the much higher score for the train data versus the test data

Hyperparameter Tuning

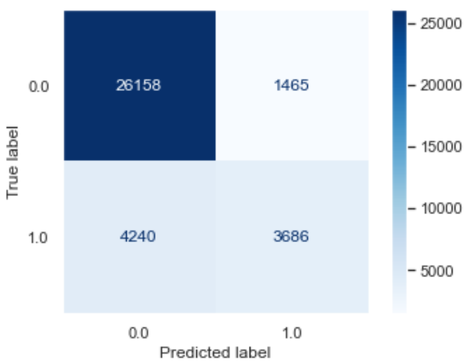
In [65]:

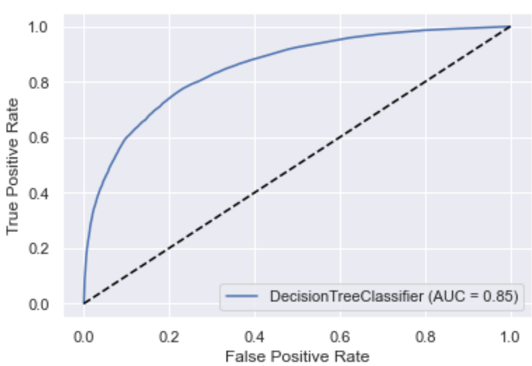
```
params = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 7, 11],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 3, 5],
    'random_state': [42]
}

clf_gs = GridSearchCV(clf, param_grid=params, scoring='accuracy', cv=3)
clf_gs.fit(X_train, y_train)
```

Out[65]:

```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(random_state=42),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 7, 11], 'min_samples_leaf': [1, 3, 5],
                         'min_samples_split': [2, 5, 10],
                         'random_state': [42]},
             scoring='accuracy')
```

Saving the grid search to a file for easy access:

In [83]:
```python
joblib.dump(clf_gs, 'saved_models/clf_gs.joblib')
```
Out[83]:
```
['saved_models/clf_gs.joblib']
```
In [45]:
```python
clf_gs = joblib.load('saved_models/clf_gs.joblib')
```
In [46]:
```python
clf_gs.best_params_
```
Out[46]:
```
{'criterion': 'gini',
 'max_depth': 7,
 'min_samples_leaf': 3,
 'min_samples_split': 10,
 'random_state': 42}
```
In [47]:
```python
round(clf_gs.best_score_, 4)
```
Out[47]:
```
0.8414
```
In [48]:
```python
y_pred_tree_gs = clf_gs.best_estimator_.predict(X_test)
y_pred_tree_gs
```
Out[48]:
```
array([1., 1., 0., ..., 0., 0., 1.])
```
In [49]:
```python
evaluate(clf_gs.best_estimator_, y_pred=y_pred_tree_gs)
```
```
Confusion Matrix
-----------------------------------------------------
```



```
Classification Report
-----------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.86      0.95      0.90     27623
         1.0       0.72      0.47      0.56      7926

    accuracy                           0.84     35549
   macro avg       0.79      0.71      0.73     35549
weighted avg       0.83      0.84      0.83     35549


ROC Curve
-----------------------------------------------------
```



```
Checking model fitness
-----------------------------------------------------
Train score: 0.8475
Test score:  0.8395
```

Observations:

- Solid increases in the evaluation metrics
- The tuned model is much better fit than the baseline model which showed overfitness

Random Forest

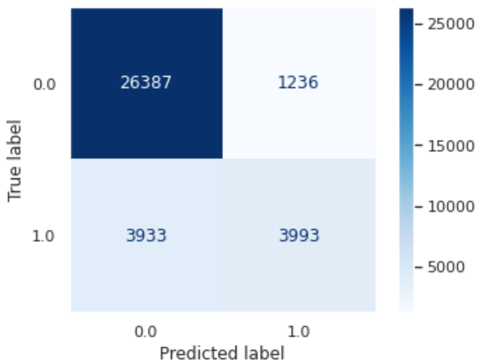
Baseline

In [51]:
```
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
y_pred_rf
```

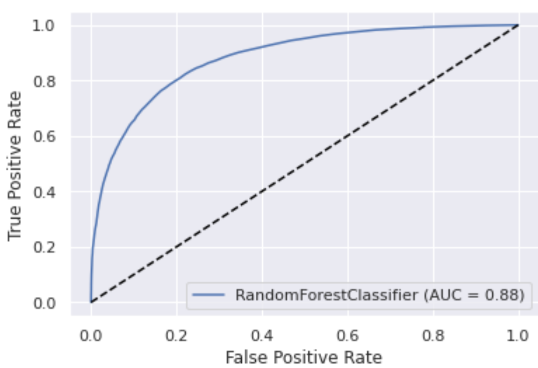
Out[51]:
```
array([1., 0., 0., ..., 0., 0., 1.])
```

In [52]:
```
evaluate(rf, y_pred=y_pred_rf)
```

```
Confusion Matrix
-----------------------------------------------------
```




```
Classification Report
-----------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.87      0.96      0.91     27623
         1.0       0.76      0.50      0.61      7926

    accuracy                           0.85     35549
   macro avg       0.82      0.73      0.76     35549
weighted avg       0.85      0.85      0.84     35549


ROC Curve
-----------------------------------------------------
```




```
Checking model fitness
-----------------------------------------------------
Train score: 1.0
Test score:  0.8546
```

Observations:

- Good scores on the evaluation metrics
- The model is a bit overfit

Hyperparameter Tuning

In [70]:
```
rf_params = {
    'n_estimators': [10, 35, 100],
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 7, 11],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 3, 5],
    'random_state': [42]
}

rf_gs = GridSearchCV(rf, param_grid=rf_params, scoring='accuracy', cv=3)
rf_gs.fit(X_train, y_train)
```

Out[70]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=42),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 7, 11], 'min_samples_leaf': [1, 3, 5],
                         'min_samples_split': [2, 5, 10],
```

```
                        'n_estimators': [10, 35, 100], 'random_state': [42]},
                scoring='accuracy')
```

In [73]:
```
joblib.dump(rf_gs, 'saved_models/rf_gs.joblib')
```

Out[73]:
```
['saved_models/rf_gs.joblib']
```

In [50]:
```
rf_gs = joblib.load('saved_models/rf_gs.joblib')
```

In [51]:
```
rf_gs.best_params_
```

Out[51]:
```
{'criterion': 'gini',
 'max_depth': 11,
 'min_samples_leaf': 1,
 'min_samples_split': 10,
 'n_estimators': 100,
 'random_state': 42}
```

In [52]:
```
round(rf_gs.best_score_, 4)
```

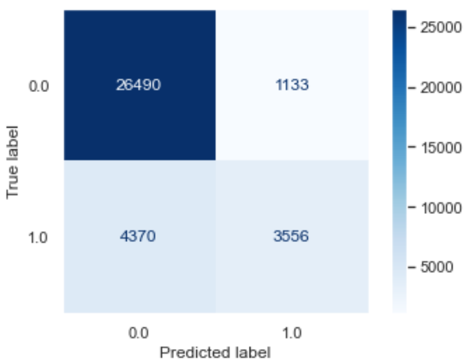
Out[52]:
```
0.8474
```

In [53]:
```
y_pred_rf_gs = rf_gs.predict(X_test)
y_pred_rf_gs
```

Out[53]:
```
array([1., 0., 0., ..., 0., 0., 1.])
```

In [54]:
```
evaluate(rf_gs, y_pred=y_pred_rf_gs)
```

```
Confusion Matrix
-----------------------------------------------------
```




```
Classification Report
-----------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.86      0.96      0.91     27623
         1.0       0.76      0.45      0.56      7926

    accuracy                           0.85     35549
   macro avg       0.81      0.70      0.73     35549
weighted avg       0.84      0.85      0.83     35549
```

```
ROC Curve
-----------------------------------------------------
```


```
Checking model fitness
-----------------------------------------------------
Train score: 0.8619
Test score:  0.8452
```

Observations:

- The accuracy score remained roughly the same while the F1 score decreased
- Small increase in the AUC of the ROC curve
- Furthermore, the tuned model has a much better fit than the baseline model

XGBoost

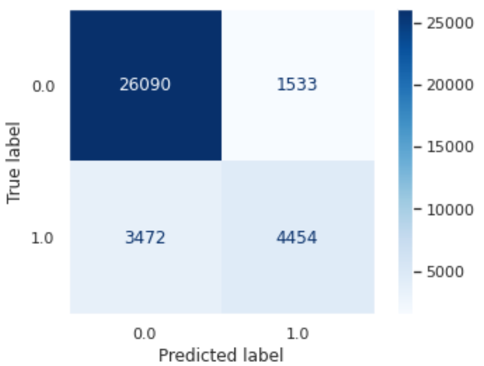
Baseline

In [53]:
```
xgb = XGBClassifier(random_state=42)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
y_pred_xgb
```

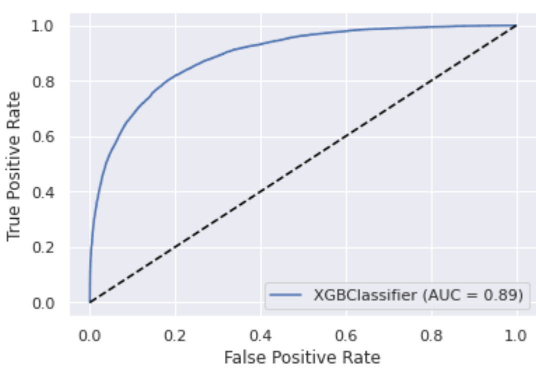
Out[53]:
```
array([1., 0., 0., ..., 0., 0., 0.])
```

In [54]:
```
evaluate(xgb, y_pred=y_pred_xgb)
```

```
Confusion Matrix
-----------------------------------------------------
```




```
Classification Report
-----------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.88      0.94      0.91     27623
         1.0       0.74      0.56      0.64      7926

    accuracy                           0.86     35549
   macro avg       0.81      0.75      0.78     35549
weighted avg       0.85      0.86      0.85     35549


ROC Curve
-----------------------------------------------------
```




```
Checking model fitness
-----------------------------------------------------
Train score: 0.8902
Test score:  0.8592
```

Observations:

- Highest accuracy score yet
- Highest AUC yet
- The model is decently fit

Hyperparameter Tuning

In [82]:
```
xgb_params = {
    'n_estimators': [10, 35, 100],
    'max_depth': [5, 10, 15],
    'learning_rate': [0.01, 0.1, 0.25]
}

xgb_gs = GridSearchCV(xgb, xgb_params, scoring='accuracy', cv=3)
xgb_gs.fit(X_train, y_train)
```

Out[82]:
```
GridSearchCV(cv=3,
```

```
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0, gpu_id=-1,
                                     importance_type='gain',
                                     interaction_constraints='',
                                     learning_rate=0.300000012,
                                     max_delta_step=0, max_depth=6,
                                     min_child_weight=1, missing=nan,
                                     monotone_constraints='()',
                                     n_estimators=100, n_jobs=0,
                                     num_parallel_tree=1, random_state=42,
                                     reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, subsample=1,
                                     tree_method='exact', validate_parameters=1,
                                     verbosity=None),
             param_grid={'learning_rate': [0.01, 0.1, 0.25],
                         'max_depth': [5, 10, 15],
                         'n_estimators': [10, 35, 100]},
             scoring='accuracy')
```

In [84]:
```
joblib.dump(xgb_gs, 'saved_models/xgb_gs.joblib')
```

Out[84]:
```
['saved_models/xgb_gs.joblib']
```

In [55]:
```
xgb_gs = joblib.load('saved_models/xgb_gs.joblib')
```

In [56]:
```
xgb_gs.best_params_
```

Out[56]:
```
{'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 100}
```

In [57]:
```
round(xgb_gs.best_score_, 4)
```

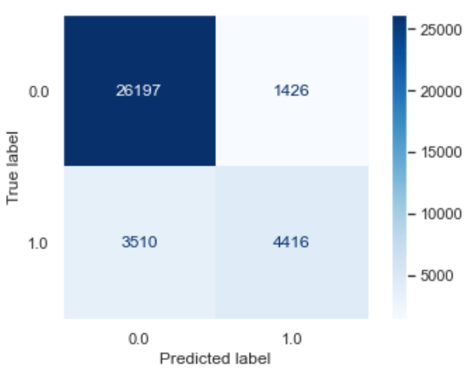
Out[57]:
```
0.8592
```

In [58]:
```
y_pred_xgb_gs = xgb_gs.predict(X_test)
y_pred_xgb_gs
```

Out[58]:
```
array([1., 0., 0., ..., 0., 0., 1.])
```

In [59]:
```
evaluate(xgb_gs, y_pred=y_pred_xgb_gs)
```

```
Confusion Matrix
-----------------------------------------------------
```




```
Classification Report
-----------------------------------------------------
              precision    recall  f1-score   support

         0.0       0.88      0.95      0.91     27623
         1.0       0.76      0.56      0.64      7926

    accuracy                           0.86     35549
   macro avg       0.82      0.75      0.78     35549
weighted avg       0.85      0.86      0.85     35549
```

```
ROC Curve
-----------------------------------------------------
```

```
Checking model fitness
-----------------------------------------------------
Train score: 0.9231
Test score:  0.8611
```

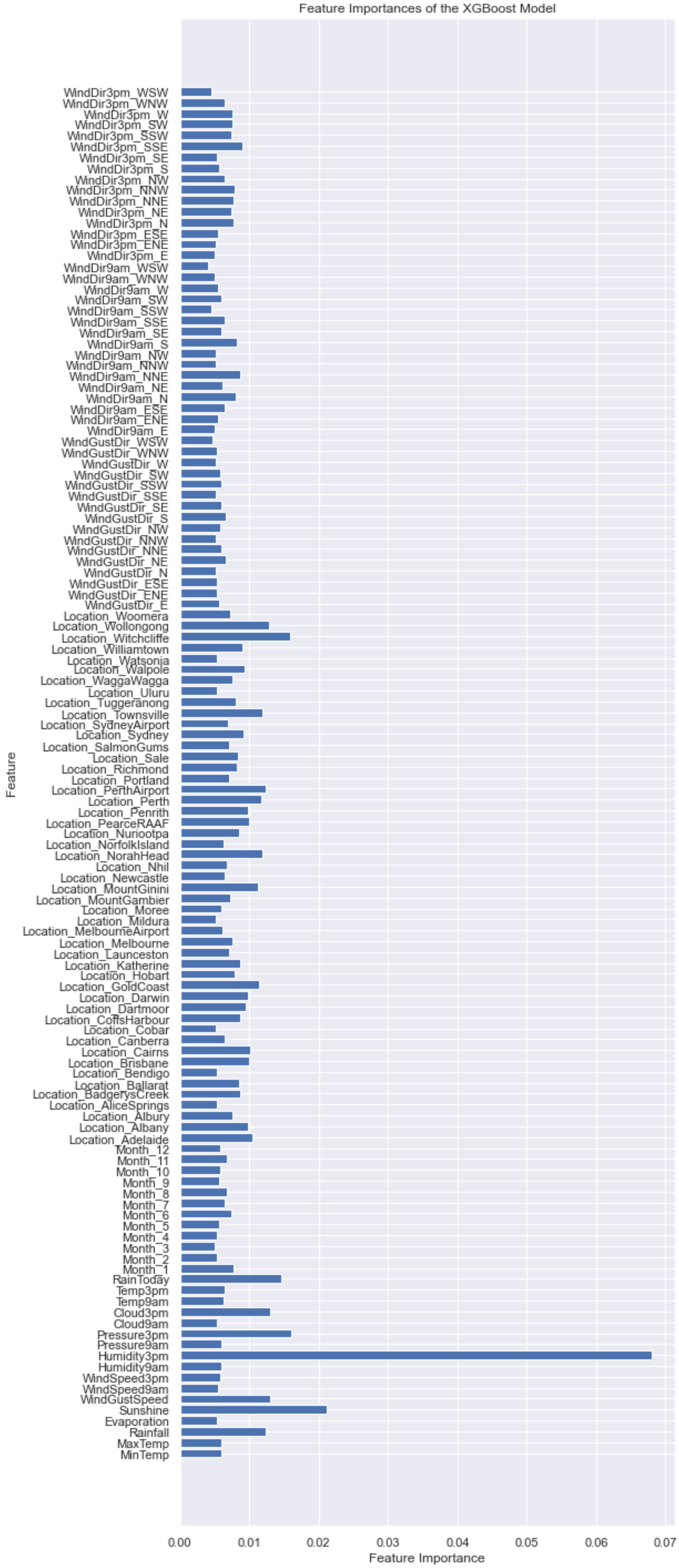
Observations:

- Slight improvement in some metrics but largely the same
- AUC remains the same
- Model fitness slightly decreased
- Overall, not much of an impact

Feature Importances

Since this model achieved the best results, I want to explore the feature importances a bit more in depth.

In [66]:
```
best_xgb = xgb_gs.best_estimator_

plt.figure(figsize=(8, 25))
plt.barh(range(best_xgb.n_features_in_), best_xgb.feature_importances_)
plt.yticks(np.arange(best_xgb.n_features_in_), X_train.columns.values)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importances of the XGBoost Model');
```



Although the dummy variables were necessary for modeling the data, they are not conducive to analyzing the feature importances. As a result, I need to regroup the data into their primary categories to aggregate their category-level importances.

In [67]:
```
feat_imp_df = pd.DataFrame(data={'Feature': df_final.columns.drop('RainTomorrow'),
                                 'Importance': best_xgb.feature_importances_
                                }
                          )
feat_imp_df['Group'] = feat_imp_df.Feature.apply(lambda x: x.split('_')[0])
feat_imp_df
```

Out[67]:

	Feature	Importance	Group
0	MinTemp	0.006021	MinTemp
1	MaxTemp	0.005982	MaxTemp
2	Rainfall	0.012458	Rainfall
3	Evaporation	0.005306	Evaporation
4	Sunshine	0.021196	Sunshine
...	...	...	...
121	WindDir3pm_SSW	0.007425	WindDir3pm
122	WindDir3pm_SW	0.007617	WindDir3pm
123	WindDir3pm_W	0.007605	WindDir3pm
124	WindDir3pm_WNW	0.006416	WindDir3pm
125	WindDir3pm_WSW	0.004475	WindDir3pm

126 rows × 3 columns

In [68]:
```
feat_imp_df.Group.value_counts()
```

Out[68]:
```
Location         49
WindDir3pm       16
WindDir9am       16
WindGustDir      16
Month            12
RainToday         1
Temp9am           1
Humidity3pm       1
Sunshine          1
WindSpeed3pm      1
Cloud3pm          1
Humidity9am       1
Pressure3pm       1
MaxTemp           1
Evaporation       1
Temp3pm           1
Rainfall          1
WindGustSpeed     1
Pressure9am       1
MinTemp           1
Cloud9am          1
WindSpeed9am      1
Name: Group, dtype: int64
```

These value counts align with the number of unique values for the categorical columns in the original dataframe (excluding `Month` which was engineered later), meaning the lambda function worked as expected.

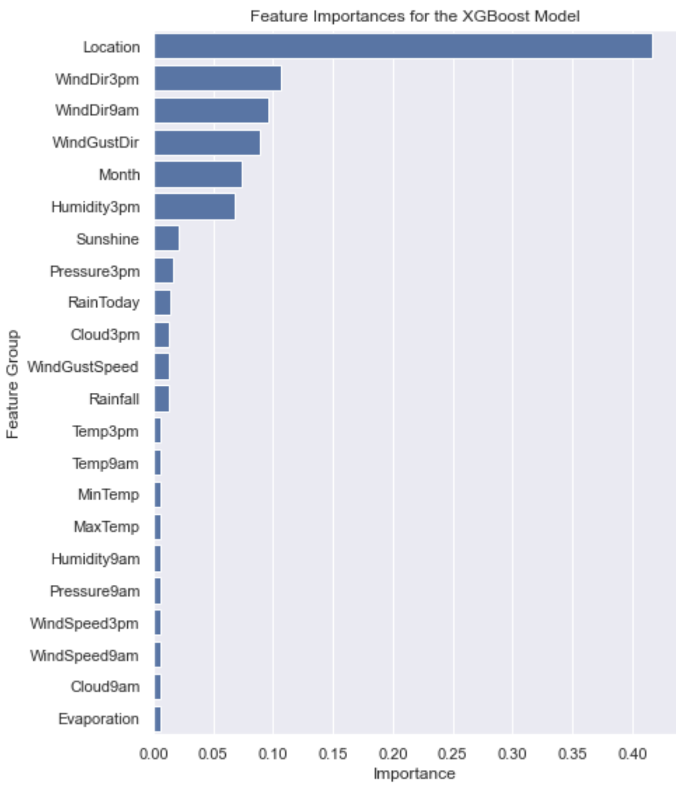
In [71]:
```
feat_imp_df_grouped = feat_imp_df.groupby(by='Group').sum()
feat_imp_df_grouped.sort_values('Importance', ascending=False, inplace=True)
feat_imp_df_grouped
```

Out[71]:

	Importance
Group	
Location	0.417204
WindDir3pm	0.106490
WindDir9am	0.096070
WindGustDir	0.089525
Month	0.073962
Humidity3pm	0.067972
Sunshine	0.021196
Pressure3pm	0.016058
RainToday	0.014569
Cloud3pm	0.012971
WindGustSpeed	0.012962
Rainfall	0.012458
Temp3pm	0.006409
Temp9am	0.006273
MinTemp	0.006021

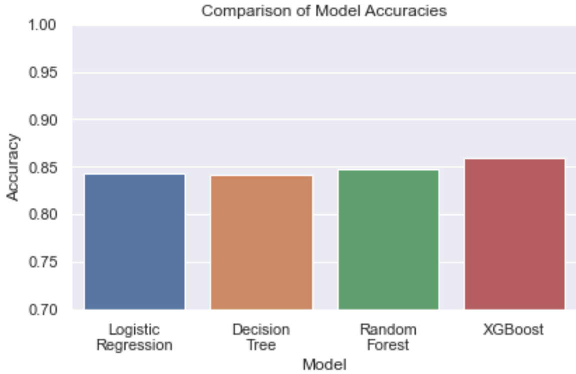
	Importance
Group	
MaxTemp	0.005982
Humidity9am	0.005966
Pressure9am	0.005937
WindSpeed3pm	0.005826
WindSpeed9am	0.005507
Cloud9am	0.005337
Evaporation	0.005306

In [82]:
```
plt.figure(figsize=(7, 8))
sns.barplot(y=feat_imp_df_grouped.index,
            x=feat_imp_df_grouped.Importance,
            orient='h',
            color=sns.color_palette()[0]
           )
plt.title('Feature Importances for the XGBoost Model')
plt.ylabel('Feature Group')
plt.xlabel('Importance')
plt.tight_layout()
plt.savefig('images/feat_importances.png', facecolor='white', dpi=100);
```



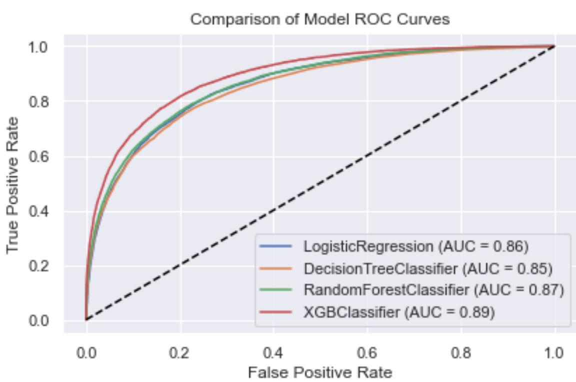
Model Comparisons

In [84]:
```
models = [logreg_gs, clf_gs, rf_gs, xgb_gs]

sns.barplot(x=['Logistic\nRegression', 'Decision\nTree', 'Random\nForest', 'XGBoost'],
            y=[model.best_score_ for model in models]
           )
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0.7, 1.0)
plt.title('Comparison of Model Accuracies')
plt.tight_layout()
plt.savefig('images/model_accuracies.png', facecolor='white', dpi=100);
```



In [85]:

```
fig, ax = plt.subplots()
for model in models:
    plot_roc_curve(model,
                   X_test,
                   y_test,
                   name=type(model.best_estimator_).__name__,
                   ax=ax
                  )
plt.plot([0, 1], [0, 1], color='black', linestyle='--')
plt.title('Comparison of Model ROC Curves')
plt.tight_layout()
plt.savefig('images/model_roc_curves.png', facecolor='white', dpi=100);
```



Conclusion

Results

The best performing model is the hyperparameter-tuned XGBoost model with an accuracy of approximately 86%. The scores for both the training and testing data were similar, reducing concerns of the model being overfit. In terms of feature importances, `Humidity3pm` is the single most important feature. However, when grouping the features back into their original categories, the following groups have the most importance:

- `Location`
- `WindDir3pm`
- `WindDir9am`
- `WindGustDir`
- `Month`
- `Humidity3pm`

Next Steps

While this model is a good starting point for rain prediction in Australia, there are several ways in which the model could be improved upon:

- Further hyperparameter tuning
- Engineering new features such as trailing amounts of rain or sunshine
- Collecting additional data from nearby countries (for example, does rain originating in Indonesia or New Zealand have predictive power?)
- Attempting to predict the *amount* of rainfall