# Data Cleaning

## Imports

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.under_sampling import RandomUnderSampler
from sklearn.preprocessing import StandardScaler
```

## Loading Original Data

In [ ]:

```
df = pd.read_csv('weatherAUS.csv')
df.head()
```

Out[ ]:

| | Date | Location | MinTemp | MaxTemp | Rainfall | Evaporation | Sunshine | WindGustDir | Winc |
|---|---|---|---|---|---|---|---|---|---|
| **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

## Binary Encoding of RainToday/Tomorrow

In [ ]:

```
# One Hot Encoding

# Update RainToday column
raintoday = pd.get_dummies(df['RainToday'])
df['RainToday'] = raintoday['Yes']

# Update RainTomorrow column
raintomorrow = pd.get_dummies(df['RainTomorrow'])
df['RainTomorrow'] = raintomorrow['Yes']

df.head()
```

Out[ ]:

| | Date | Location | MinTemp | MaxTemp | Rainfall | Evaporation | Sunshine | WindGustDir | Winc |
|---|---|---|---|---|---|---|---|---|---|
| **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

## Remove NaNs

In [ ]:

```
# Remove columns with over 1/3 NaNs
df.drop(['Evaporation', 'Sunshine', 'Rainfall', 'Cloud9am', 'Cloud3pm', 'Loc

df.head()
```

Out[ ]:

| | Date | MinTemp | MaxTemp | WindGustDir | WindGustSpeed | WindDir9am | WindDir3pm | Wi |
|---|---|---|---|---|---|---|---|---|
| **0** | 2008-12-01 | 13.4 | 22.9 | W | 44.0 | W | WNW | |
| **1** | 2008-12-02 | 7.4 | 25.1 | WNW | 44.0 | NNW | WSW | |
| **2** | 2008-12-03 | 12.9 | 25.7 | WSW | 46.0 | W | WSW | |
| **3** | 2008-12-04 | 9.2 | 28.0 | NE | 24.0 | SE | E | |
| **4** | 2008-12-05 | 17.5 | 32.3 | W | 41.0 | ENE | NW | |

In [ ]:

```
# Drop NaN values
df.dropna(inplace = True)

df.head()
```

Out[ ]:

| | Date | MinTemp | MaxTemp | WindGustDir | WindGustSpeed | WindDir9am | WindDir3pm | Wi |
|---|---|---|---|---|---|---|---|---|
| **0** | 2008-12-01 | 13.4 | 22.9 | W | 44.0 | W | WNW | |
| **1** | 2008-12-02 | 7.4 | 25.1 | WNW | 44.0 | NNW | WSW | |
| **2** | 2008-12-03 | 12.9 | 25.7 | WSW | 46.0 | W | WSW | |
| **3** | 2008-12-04 | 9.2 | 28.0 | NE | 24.0 | SE | E | |
| **4** | 2008-12-05 | 17.5 | 32.3 | W | 41.0 | ENE | NW | |

---

## Encode Remaining Categorical Variables

In [ ]:

```
# Update Date Column

df['year'] = pd.DatetimeIndex(df['Date']).year
df['month'] = pd.DatetimeIndex(df['Date']).month
df['day'] = pd.DatetimeIndex(df['Date']).day
df.drop(['Date'], axis = 1, inplace = True)
df.head()
```

Out[ ]:

| | MinTemp | MaxTemp | WindGustDir | WindGustSpeed | WindDir9am | WindDir3pm | WindSpee |
|---|---|---|---|---|---|---|---|
| **0** | 13.4 | 22.9 | W | 44.0 | W | WNW | |
| **1** | 7.4 | 25.1 | WNW | 44.0 | NNW | WSW | |
| **2** | 12.9 | 25.7 | WSW | 46.0 | W | WSW | |
| **3** | 9.2 | 28.0 | NE | 24.0 | SE | E | |
| **4** | 17.5 | 32.3 | W | 41.0 | ENE | NW | |

In [ ]:

```
df.WindGustDir.value_counts().sort_values(ascending=False)
```

Out[ ]:

```
W      8542
SE     8215
E      7995
SSE    7925
S      7912
N      7896
WSW    7821
SW     7773
SSW    7522
WNW    6982
ENE    6904
NW     6553
ESE    6315
NE     6148
NNE    5604
NNW    5308
Name: WindGustDir, dtype: int64
```

In [ ]:

```
def encoding_N(row):
    if 'N' in row[column]:
        return 1
    else:
        return 0

def encoding_E(row):
    if 'E' in row[column]:
        return 1
    else:
        return 0

def encoding_S(row):
    if 'S' in row[column]:
        return 1
    else:
        return 0

def encoding_W(row):
    if 'W' in row[column]:
        return 1
    else:
        return 0
```

In [ ]:

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

for column in update_columns:
    df[column+'N'] = df.apply(lambda row: encoding_N(row), axis=1)
    df[column+'E'] = df.apply(lambda row: encoding_E(row), axis=1)
    df[column+'S'] = df.apply(lambda row: encoding_S(row), axis=1)
    df[column+'W'] = df.apply(lambda row: encoding_W(row), axis=1)

df.drop(update_columns, axis = 1, inplace = True)
```

In [ ]:

```
df.head()
```

Out[ ]:

| | MinTemp | MaxTemp | WindGustSpeed | WindSpeed9am | WindSpeed3pm | Humidity9am | Hu |
|---|---|---|---|---|---|---|---|
| **0** | 13.4 | 22.9 | 44.0 | 20.0 | 24.0 | 71.0 | |
| **1** | 7.4 | 25.1 | 44.0 | 4.0 | 22.0 | 44.0 | |
| **2** | 12.9 | 25.7 | 46.0 | 19.0 | 26.0 | 38.0 | |
| **3** | 9.2 | 28.0 | 24.0 | 11.0 | 9.0 | 45.0 | |
| **4** | 17.5 | 32.3 | 41.0 | 7.0 | 20.0 | 82.0 | |

5 rows × 28 columns

# Data Exploration

## Initial Descriptions

In [ ]:

```
df.describe()
```

Out[ ]:

| | MinTemp | MaxTemp | WindGustSpeed | WindSpeed9am | WindSpeed3pm | Hu |
|---|---|---|---|---|---|---|
| **count** | 115415.000000 | 115415.000000 | 115415.000000 | 115415.000000 | 115415.000000 | 1154 |
| **mean** | 12.672407 | 23.644513 | 40.828081 | 15.216688 | 19.516371 | |
| **std** | 6.238970 | 6.971647 | 13.338630 | 8.362088 | 8.586030 | |
| **min** | -8.200000 | 2.600000 | 7.000000 | 2.000000 | 2.000000 | |
| **25%** | 8.100000 | 18.300000 | 31.000000 | 9.000000 | 13.000000 | |
| **50%** | 12.400000 | 23.100000 | 39.000000 | 13.000000 | 19.000000 | |
| **75%** | 17.200000 | 28.700000 | 48.000000 | 20.000000 | 24.000000 | |
| **max** | 33.900000 | 48.100000 | 135.000000 | 87.000000 | 87.000000 | 1 |

8 rows × 28 columns

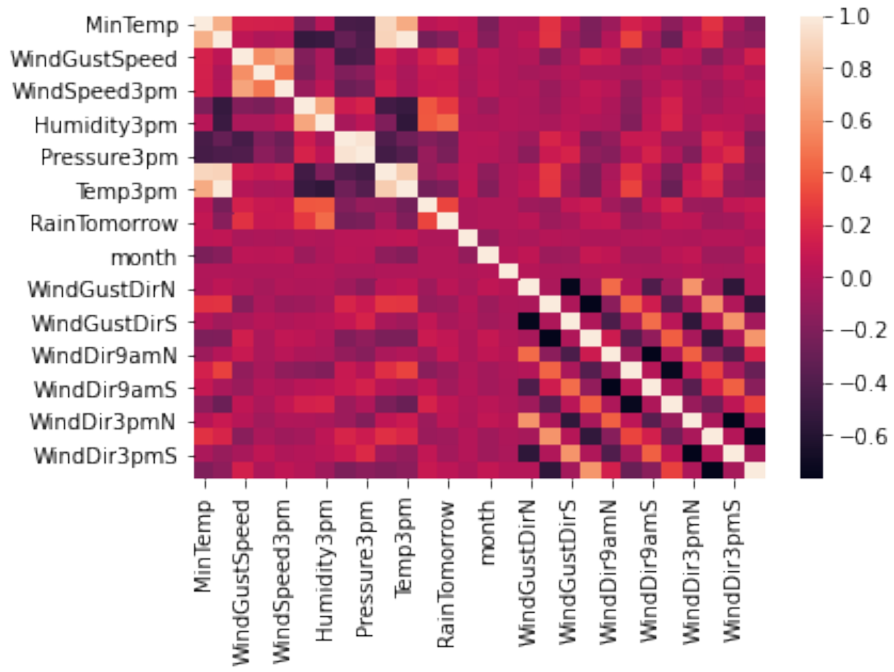
## Correlation Matrix

In [ ]:

```
# Correlation Matrix attributes with other attributes

corr_map = sns.heatmap(df.corr())
corr_map
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb552977910>
```



In [ ]:

```
correlate = df.corr()
correlate
```

Out[ ]:

| | MinTemp | MaxTemp | WindGustSpeed | WindSpeed9am | WindSpeed3pm | H |
|---|---|---|---|---|---|---|
| **MinTemp** | 1.000000 | 0.728610 | 0.159282 | 0.150052 | 0.145654 | |
| **MaxTemp** | 0.728610 | 1.000000 | 0.056850 | -0.005989 | 0.013626 | |
| **WindGustSpeed** | 0.159282 | 0.056850 | 1.000000 | 0.590132 | 0.678569 | |
| **WindSpeed9am** | 0.150052 | -0.005989 | 0.590132 | 1.000000 | 0.498255 | |
| **WindSpeed3pm** | 0.145654 | 0.013626 | 0.678569 | 0.498255 | 1.000000 | |
| **Humidity9am** | -0.217067 | -0.515588 | -0.183934 | -0.222205 | -0.093797 | |
| **Humidity3pm** | 0.026907 | -0.497169 | -0.021263 | -0.018744 | 0.052981 | |
| **Pressure9am** | -0.433600 | -0.312804 | -0.446087 | -0.202463 | -0.285172 | |
| **Pressure3pm** | -0.448056 | -0.411641 | -0.400743 | -0.151562 | -0.245085 | |
| **Temp9am** | 0.898502 | 0.884949 | 0.124632 | 0.089087 | 0.126241 | |
| **Temp3pm** | 0.703989 | 0.984276 | 0.018299 | -0.019171 | -0.011790 | |
| **RainToday** | 0.043948 | -0.240498 | 0.152262 | 0.097250 | 0.086494 | |
| **RainTomorrow** | 0.077364 | -0.167040 | 0.234628 | 0.091620 | 0.095698 | |
| **year** | 0.045828 | 0.062946 | -0.027563 | -0.014091 | -0.027539 | |
| **month** | -0.211504 | -0.171276 | 0.051801 | 0.042637 | 0.057476 | |
| **day** | 0.001575 | -0.001384 | -0.008812 | -0.006671 | -0.009200 | |
| **WindGustDirN** | -0.006184 | 0.058115 | 0.003732 | -0.039966 | 0.004605 | |
| **WindGustDirE** | 0.236121 | 0.239784 | -0.167436 | -0.022673 | -0.092648 | |
| **WindGustDirS** | 0.011241 | -0.064723 | -0.021431 | 0.028567 | 0.001690 | |
| **WindGustDirW** | -0.208689 | -0.203830 | 0.140730 | -0.021348 | 0.071292 | |
| **WindDir9amN** | -0.082341 | 0.017123 | 0.094629 | -0.021269 | 0.034362 | |
| **WindDir9amE** | 0.150988 | 0.294265 | -0.125451 | -0.033222 | -0.133435 | |
| **WindDir9amS** | 0.090011 | -0.025456 | -0.086797 | 0.010572 | -0.011101 | |
| **WindDir9amW** | -0.145901 | -0.283775 | 0.062170 | -0.020491 | 0.069909 | |
| **WindDir3pmN** | 0.020486 | 0.101639 | 0.024387 | -0.064041 | -0.001086 | |
| **WindDir3pmE** | 0.232103 | 0.175630 | -0.157909 | -0.023885 | -0.074840 | |
| **WindDir3pmS** | -0.019005 | -0.102426 | -0.033484 | 0.063355 | -0.004145 | |
| **WindDir3pmW** | -0.197892 | -0.138989 | 0.144229 | -0.014892 | 0.064907 | |

28 rows × 28 columns

In [ ]:

```
correlated_features = set()

for i in range(len(correlate.columns)):
    for j in range(i):
        if abs(correlate.iloc[i, j]) > 0.85:
            colname = correlate.columns[i]
            correlated_features.add(colname)

df.drop(labels=correlated_features, axis=1, inplace=True)
```

In [ ]:

```
df.head()
```

Out[ ]:

| | MinTemp | MaxTemp | WindGustSpeed | WindSpeed9am | WindSpeed3pm | Humidity9am | Hu |
|---|---|---|---|---|---|---|---|
| **0** | 13.4 | 22.9 | 44.0 | 20.0 | 24.0 | 71.0 | |
| **1** | 7.4 | 25.1 | 44.0 | 4.0 | 22.0 | 44.0 | |
| **2** | 12.9 | 25.7 | 46.0 | 19.0 | 26.0 | 38.0 | |
| **3** | 9.2 | 28.0 | 24.0 | 11.0 | 9.0 | 45.0 | |
| **4** | 17.5 | 32.3 | 41.0 | 7.0 | 20.0 | 82.0 | |

5 rows × 25 columns

# Balancing Our Data

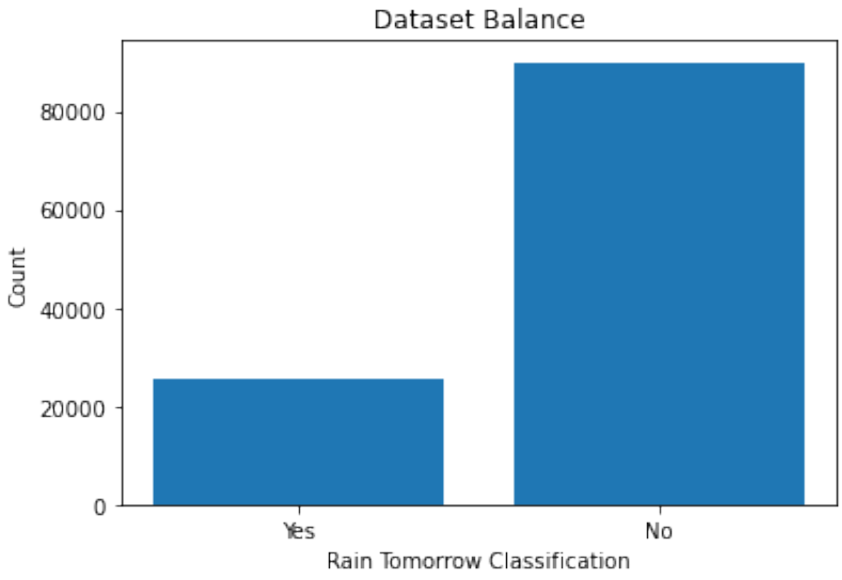
## Undersampling

In [ ]:

```
# Barplot to display original balance of data

sum_yes = df['RainTomorrow'].sum()
sum_no = len(df) - sum_yes

x_labels = ['Yes', 'No']
y_labels = [sum_yes, sum_no]

plt.bar(x_labels, y_labels)
plt.xlabel('Rain Tomorrow Classification')
plt.ylabel('Count')
plt.title('Dataset Balance')
```

Out[ ]:

```
Text(0.5, 1.0, 'Dataset Balance')
```



In [ ]:

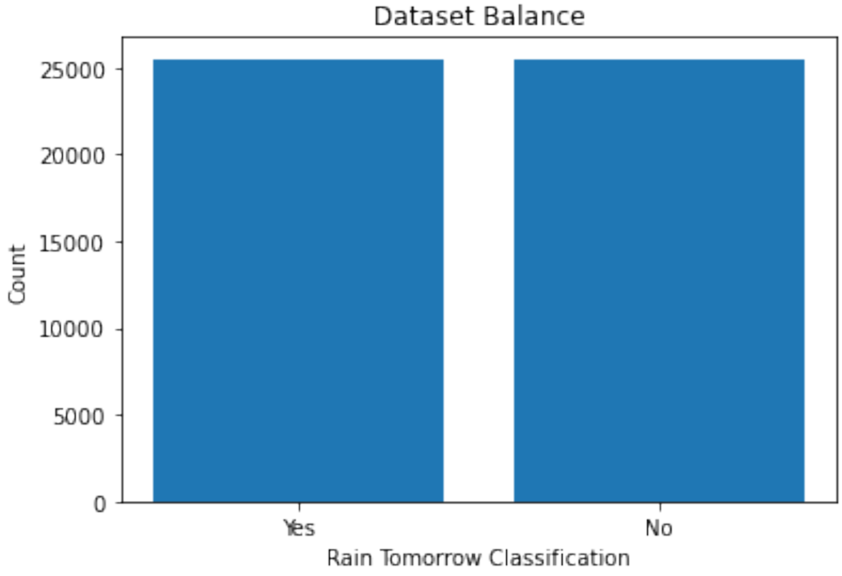
```python
# Undersampling
X = df.drop("RainTomorrow", axis=1)
y = df["RainTomorrow"]
RUS = RandomUnderSampler(random_state=42)
X_und, y_und = RUS.fit_sample(X, y)
```

In [ ]:

```python
# Barplot of data after undersampling

sum_yes = y_und.sum()
sum_no = len(y_und) - sum_yes

x_labels = ['Yes', 'No']
y_labels = [sum_yes, sum_no]

plt.bar(x_labels, y_labels)
plt.xlabel('Rain Tomorrow Classification')
plt.ylabel('Count')
plt.title('Dataset Balance')
```

Out[ ]:

```
Text(0.5, 1.0, 'Dataset Balance')
```



## Standardize Data

```
In [ ]: scaler = StandardScaler()
        scaler.fit(X_und)
        standardized_X = scaler.transform(features)
```

## Download the New Dataset

```
In [ ]: # Download
        standardized_X.to_csv("cleaned_rain_x.csv")
        y_und.to_csv("cleaned_rain_y.csv")
```