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

In [2]:
```
df = pd.read_csv('weatherAUS.csv')
df.head()
```

Out[2]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDi
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	

5 rows × 23 columns

In [3]:
```
df.columns
```

Out[3]:
```
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')
```

In [4]:
```
df.size
```

Out[4]:
```
3345580
```

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

Out[5]:
```
(145460, 23)
```

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

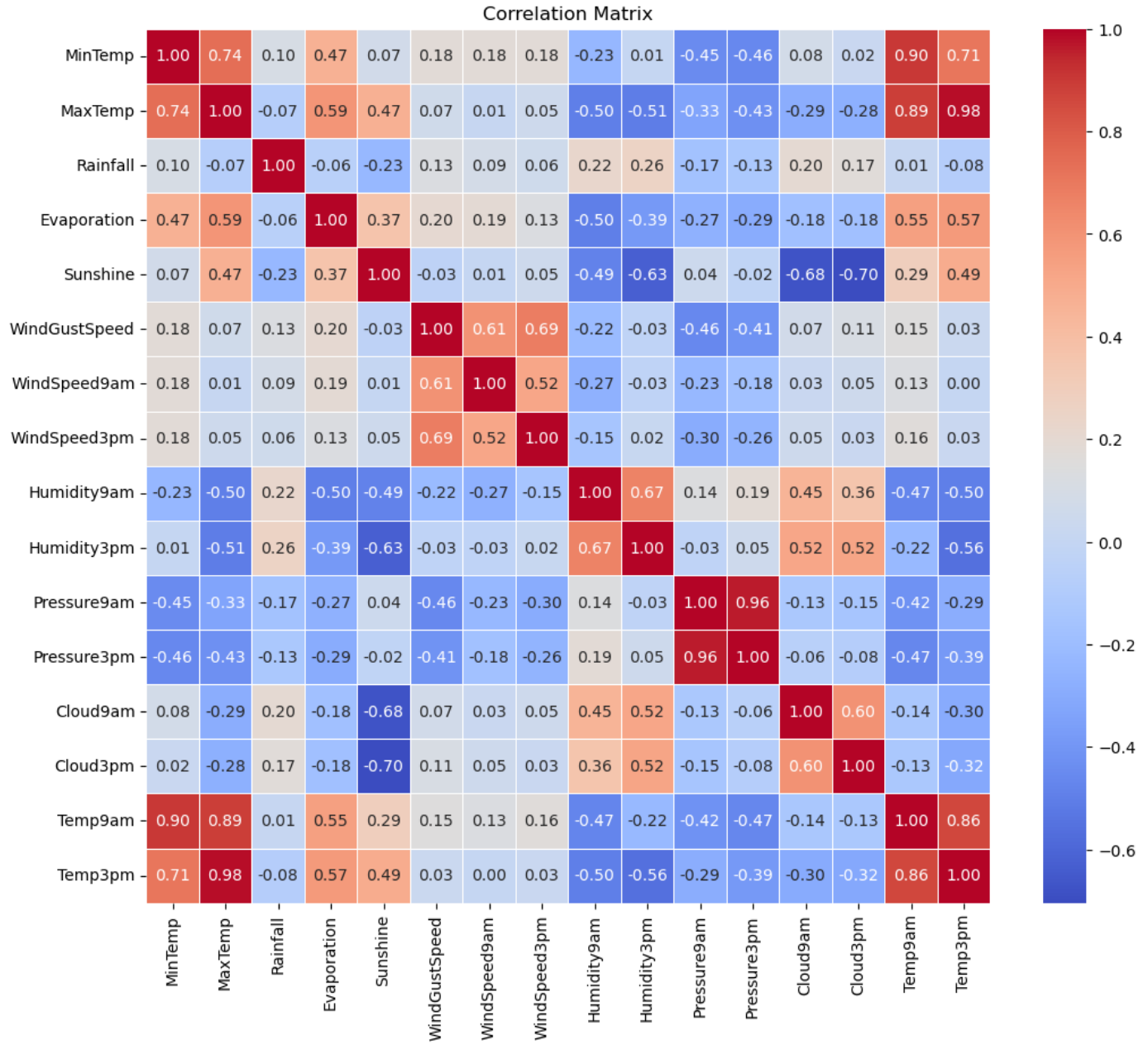
Out[6]:
```
Date                0
Location            0
MinTemp          1485
MaxTemp          1261
Rainfall         3261
Evaporation     62790
Sunshine        69835
WindGustDir     10326
WindGustSpeed   10263
WindDir9am      10566
WindDir3pm       4228
WindSpeed9am     1767
WindSpeed3pm     3062
Humidity9am      2654
Humidity3pm      4507
Pressure9am     15065
```

```
Pressure3pm      15028
Cloud9am         55888
Cloud3pm         59358
Temp9am           1767
Temp3pm           3609
RainToday         3261
RainTomorrow      3267
dtype: int64
```

In [7]:

```
correlation_matrix = df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

```
C:\Users\shmi\AppData\Local\Temp\ipykernel_12968\2405933513.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  correlation_matrix = df.corr()
```



In [8]:

```
from sklearn.impute import SimpleImputer

# Identify missing values in critical columns
critical_columns = ['RainTomorrow', 'Rainfall', 'Humidity9am', 'WindGustSpeed']
```

```
missing_data = df[critical_columns].isnull()


missing_counts = missing_data.sum()
print("Missing Value Counts:")
print(missing_counts)

# Impute RainTomorrow with mode
mode_imputer = SimpleImputer(strategy="most_frequent")
df['RainTomorrow'] = mode_imputer.fit_transform(df[['RainTomorrow']])

# Impute Rainfall, Humidity9am, WindGustSpeed with mean
mean_imputer = SimpleImputer(strategy="mean")
df[['Rainfall', 'Humidity9am', 'WindGustSpeed']] = mean_imputer.fit_transform(df[['Rainf


# Verify if there are any missing values left
print("\nMissing Values After Imputation:")
print(df[critical_columns].isnull().sum())
```

```
Missing Value Counts:
RainTomorrow      3267
Rainfall          3261
Humidity9am       2654
WindGustSpeed    10263
dtype: int64

Missing Values After Imputation:
RainTomorrow     0
Rainfall         0
Humidity9am      0
WindGustSpeed    0
dtype: int64
```

In [9]:

```
# Converting date column to datetime format
df['Date'] = pd.to_datetime(df['Date'])
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day


df.drop(columns=['Date'], inplace=True)

# Converting categorical variables to appropriate formats
categorical_columns = ['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday
df[categorical_columns] = df[categorical_columns].astype('category')


print(df.dtypes)
```

```
Location         category
MinTemp           float64
MaxTemp           float64
Rainfall          float64
Evaporation       float64
Sunshine          float64
WindGustDir      category
WindGustSpeed     float64
WindDir9am       category
WindDir3pm       category
WindSpeed9am      float64
WindSpeed3pm      float64
Humidity9am       float64
Humidity3pm       float64
Pressure9am       float64
Pressure3pm       float64
```

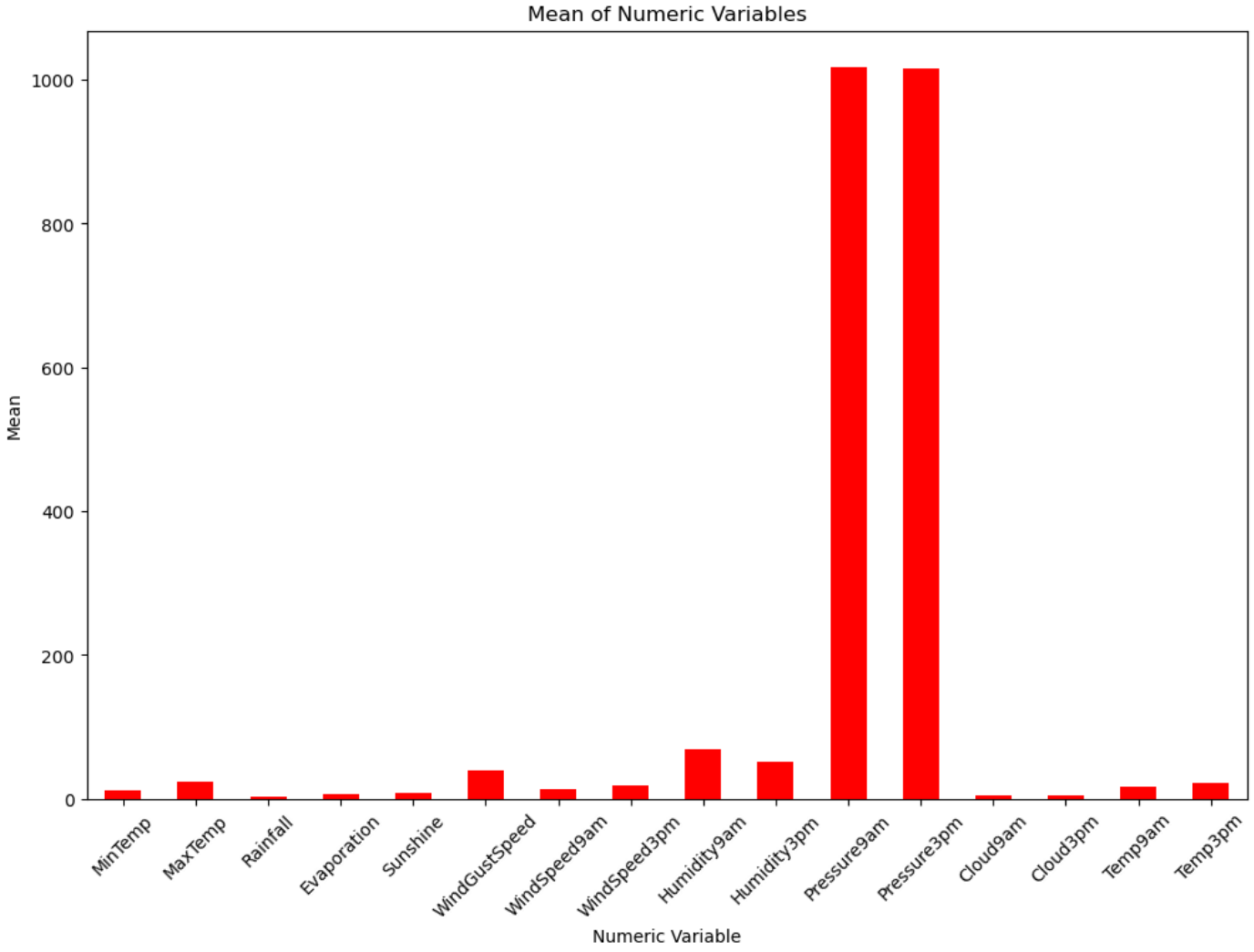
```
Cloud9am          float64
Cloud3pm          float64
Temp9am           float64
Temp3pm           float64
RainToday        category
RainTomorrow     category
Year                int64
Month               int64
Day                 int64
dtype: object
```

In [12]:

```
# Numeric columns for outlier detection
numeric_columns = ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine',
                   'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm',
                   'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm',
                   'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm']

# Calculate the mean of each numeric variable
mean_values = df[numeric_columns].mean()

# Plot the bar chart
plt.figure(figsize=(12, 8))
mean_values.plot(kind='bar', color='red')
plt.title('Mean of Numeric Variables')
plt.xlabel('Numeric Variable')
plt.ylabel('Mean')
plt.xticks(rotation=45)
plt.show()
```



In [13]:

```
# Statistical methods for outlier detection
# Let's use Z-score method for simplicity
```

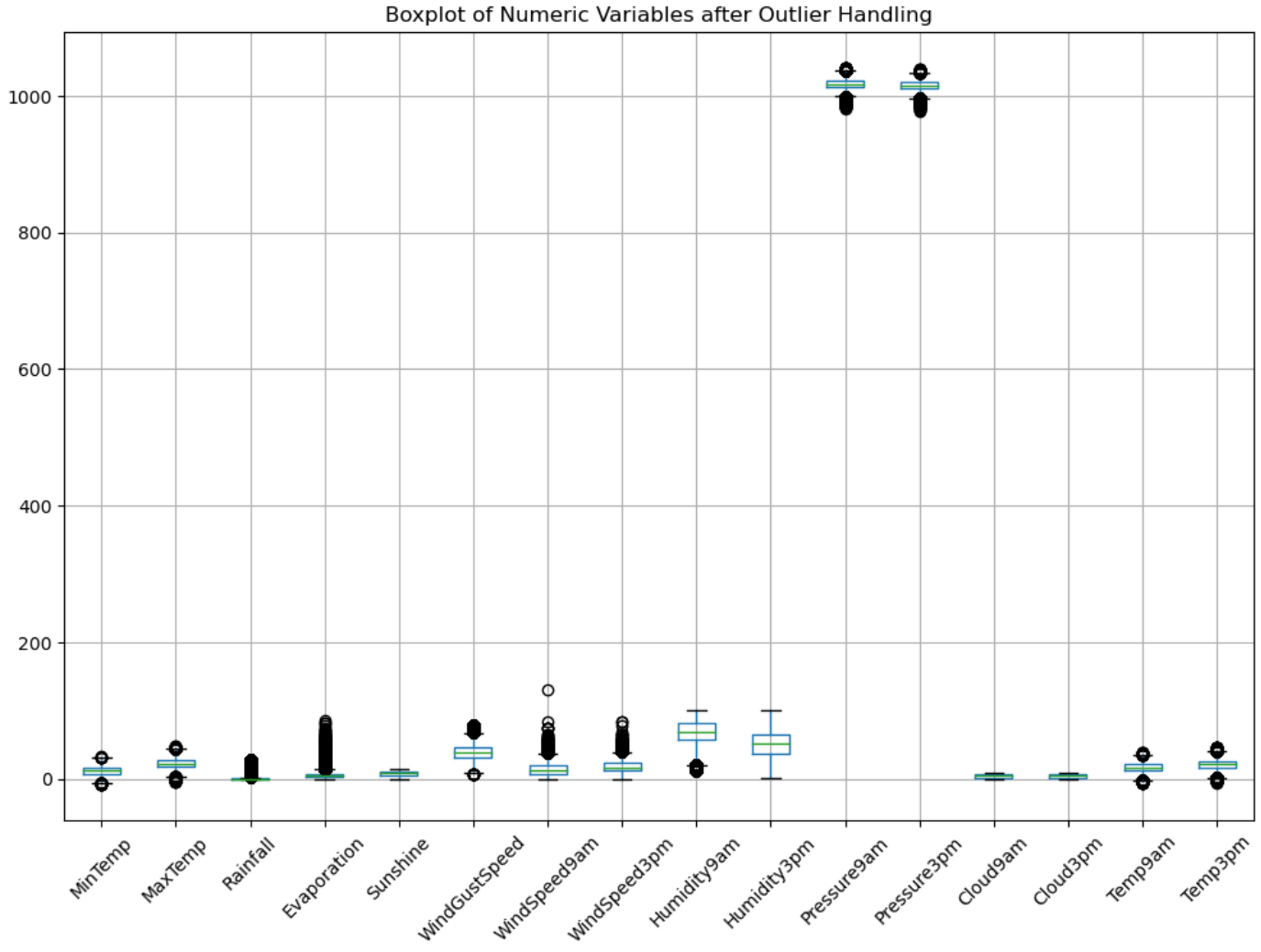
```
from scipy import stats
z_scores = stats.zscore(df[numeric_columns])
abs_z_scores = abs(z_scores)
outlier_rows = (abs_z_scores > 3).any(axis=1)  # Any variable having z-score > 3 is cons

# Print the number of outliers detected
print("Number of Outliers Detected:", outlier_rows.sum())

# Handle outliers
# For simplicity, let's replace outliers with NaN (you can choose a more appropriate met
df.loc[outlier_rows, numeric_columns] = None

# Visual inspection after handling outliers
plt.figure(figsize=(12, 8))
df[numeric_columns].boxplot()
plt.xticks(rotation=45)
plt.title('Boxplot of Numeric Variables after Outlier Handling')
plt.show()
```

```
Number of Outliers Detected: 4670
```



In [14]:

```
# Define numeric columns for outlier detection
numeric_columns = ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine',
                   'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm',
                   'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm',
                   'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm']

# Define a function to detect and handle outliers using IQR
def handle_outliers(df, columns):
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
```

```
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        # Replace outliers with NaN
        df[col] = df[col].mask((df[col] < lower_bound) | (df[col] > upper_bound))
    return df

# Handle outliers in numeric columns
df = handle_outliers(df, numeric_columns)

# Drop rows with missing values after outlier handling
df.dropna(inplace=True)

# Check the cleaned dataset
print(df.head())
```

```
     Location  MinTemp  MaxTemp  Rainfall  Evaporation  Sunshine WindGustDir  \
6049    Cobar     17.9     35.2       0.0         12.0      12.3         SSW
6052    Cobar     19.4     37.6       0.0         10.8      10.6         NNE
6053    Cobar     21.9     38.4       0.0         11.4      12.2         WNW
6055    Cobar     27.1     36.1       0.0         13.0       0.0           N
6056    Cobar     23.3     34.0       0.0          9.8      12.6         SSW

      WindGustSpeed WindDir9am WindDir3pm  ...  Pressure3pm  Cloud9am  \
6049           48.0        ENE         SW  ...       1004.4       2.0
6052           46.0        NNE        NNW  ...       1009.2       1.0
6053           31.0        WNW        WSW  ...       1009.1       1.0
6055           43.0          N        WNW  ...       1007.4       8.0
6056           41.0          S        SSE  ...       1009.9       3.0

      Cloud3pm  Temp9am  Temp3pm  RainToday  RainTomorrow  Year  Month  Day
6049       5.0     26.6     33.4         No            No  2009      1    1
6052       6.0     28.7     34.9         No            No  2009      1    4
6053       5.0     29.1     35.6         No            No  2009      1    5
6055       8.0     30.7     34.3         No            No  2009      1    7
6056       1.0     25.0     31.5         No            No  2009      1    8

[5 rows x 25 columns]
```

In [15]:
```
df['TempChange'] = df['Temp3pm'] - df['Temp9am']
print(df['TempChange'])
```

```
6049      6.8
6052      6.2
6053      6.5
6055      3.6
6056      6.5
         ...
142298    7.8
142299    7.2
142300    7.3
142301    4.4
142302    5.6
Name: TempChange, Length: 42392, dtype: float64
```

In [16]:
```
df['HumidityChange']= df['Humidity3pm'] - df['Humidity9am']
print(df['HumidityChange'])
```

```
6049      -7.0
6052     -20.0
6053     -15.0
6055      -7.0
6056     -18.0
          ...
142298   -31.0
142299   -28.0
142300   -23.0
```

```
142301    -4.0
142302   -41.0
Name: HumidityChange, Length: 42392, dtype: float64
```

In [17]:
```
df['PressureChange']= df['Pressure3pm'] - df['Pressure9am']
print(df['PressureChange'])
```

```
6049     -1.9
6052     -3.1
6053     -3.6
6055     -0.3
6056     -1.4
         ...
142298   -3.4
142299   -3.4
142300   -3.5
142301   -4.2
142302   -4.2
Name: PressureChange, Length: 42392, dtype: float64
```

In [18]:
```
df.head()
```

Out[18]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9an
6049	Cobar	17.9	35.2	0.0	12.0	12.3	SSW	48.0	EN
6052	Cobar	19.4	37.6	0.0	10.8	10.6	NNE	46.0	NN
6053	Cobar	21.9	38.4	0.0	11.4	12.2	WNW	31.0	WNW
6055	Cobar	27.1	36.1	0.0	13.0	0.0	N	43.0	N
6056	Cobar	23.3	34.0	0.0	9.8	12.6	SSW	41.0	S

5 rows × 28 columns

In [19]:
```
df.dropna(inplace=True)
```

In [20]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
X = df[['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine',
            'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm',
            'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm',
            'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm',
            'TempChange', 'HumidityChange', 'PressureChange']]
y=df['RainTomorrow']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
```

In [21]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8207335770727681
```

In [22]:
```
param_grid = {
    'max_depth': [3, 5, 7, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

In [23]:
```
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=5, scoring='accuracy
```

```
grid_search.fit(X, y)
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best Score: 0.8706361728340344
```

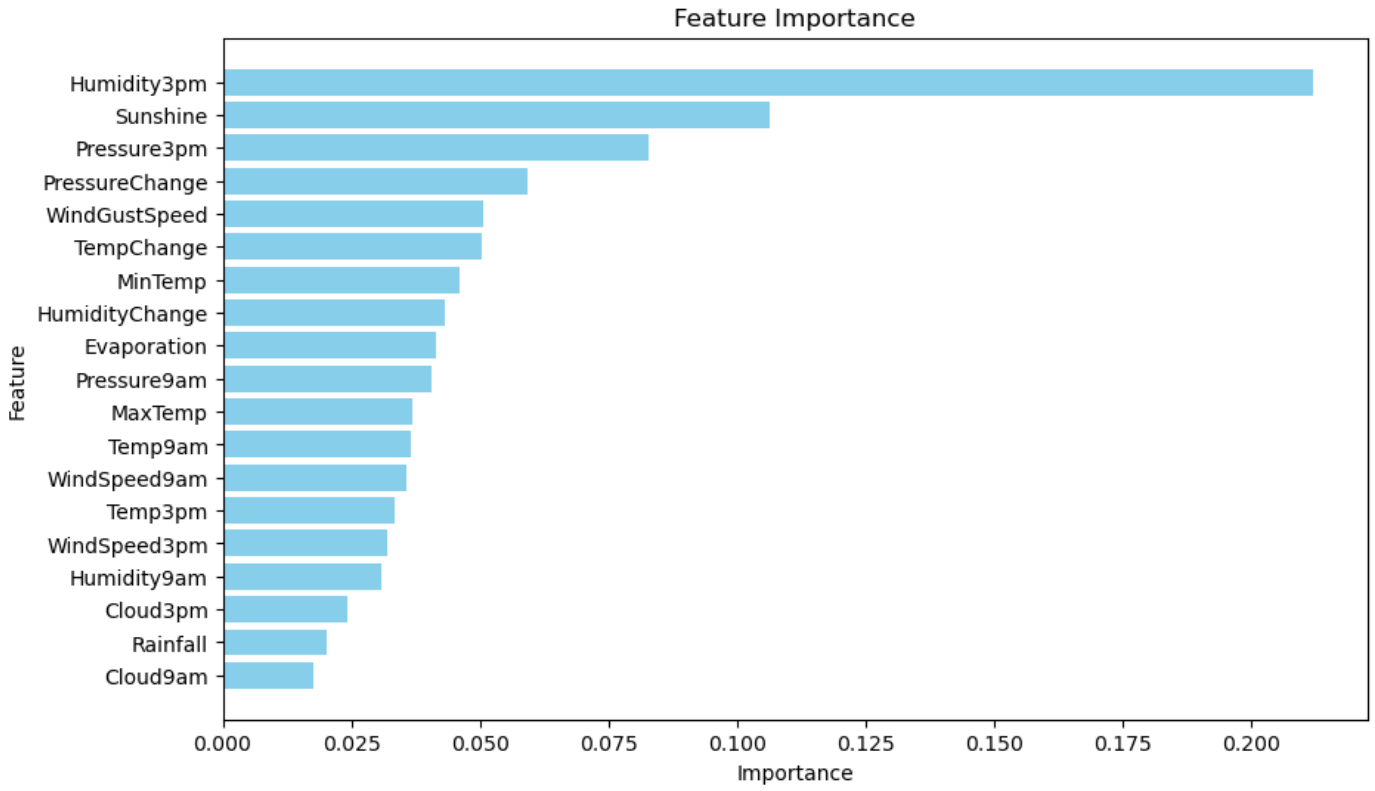
In [24]:
```
# Get feature importances
feature_importances = clf.feature_importances_

# Create a DataFrame to store feature names and importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_import

# Sort the DataFrame by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=Fal

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'], color='s
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance')
plt.gca().invert_yaxis()  # Invert y-axis to display the most important features at the
plt.show()

# Print the sorted feature importances
print(feature_importance_df)
```



	Feature	Importance
9	Humidity3pm	0.211831
4	Sunshine	0.106339
11	Pressure3pm	0.082852
18	PressureChange	0.059327
5	WindGustSpeed	0.050780
16	TempChange	0.050251
0	MinTemp	0.046057
17	HumidityChange	0.043095
3	Evaporation	0.041372
10	Pressure9am	0.040682
1	MaxTemp	0.036735
14	Temp9am	0.036612

```
6     WindSpeed9am    0.035854
15         Temp3pm    0.033511
7     WindSpeed3pm    0.032027
8      Humidity9am    0.030778
13        Cloud3pm    0.024198
2         Rainfall    0.020165
12        Cloud9am    0.017533
```

In [ ]: