The folowoing code is an example of building a linear regression model to predict the temperature based on the apparent temperature using the weather history dataset. The dataset is first loaded and checked for any missing values, which are then dropped. The correlation matrix and a pair plot are then used to visualize the relationship between the variables.

In [1]:
```
# Import pandas library to read and manipulate datasets
import pandas as pd
```

In [2]:
```
# Read the CSV file into a pandas DataFrame
weather_data=pd.read_csv("weatherHistory.csv")

# Check for missing values in the DataFrame and calculate the sum of missing values for each column
weather_data.isnull().sum()
```

Out[2]:
```
Formatted Date                0
Summary                       0
Precip Type                 517
Temperature (C)               0
Apparent Temperature (C)      0
Humidity                      0
Wind Speed (km/h)             0
Wind Bearing (degrees)        0
Visibility (km)               0
Loud Cover                    0
Pressure (millibars)          0
Daily Summary                 0
dtype: int64
```

In [3]:
```
# Drop rows with missing values from the DataFrame

weather_data=weather_data.dropna()
```

In [4]:
```
# Check for missing values in the DataFrame after dropping rows with missing values and calculate the sum of missing values for each column

weather_data.isnull().sum()
```

Out[4]:
```
Formatted Date              0
Summary                     0
Precip Type                 0
Temperature (C)             0
Apparent Temperature (C)    0
Humidity                    0
Wind Speed (km/h)           0
Wind Bearing (degrees)      0
Visibility (km)             0
Loud Cover                  0
Pressure (millibars)        0
Daily Summary               0
dtype: int64
```

In [5]:
```
weather_data.head()
```

Out[5]:

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout the day.
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout the day.
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout the day.
4	2006-04-01 04:00:00.000 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51	Partly cloudy throughout the day.

The purpose of the code is to visualize the relationships between variables in a dataset using two types of plots: a correlation matrix and a scatterplot matrix.

The correlation matrix provides information on the strength and direction of the linear relationship between pairs of variables in the dataset. The values in the matrix range from -1 to 1, with a value of 1 indicating a perfect positive correlation, 0 indicating no correlation, and -1 indicating a perfect negative correlation.

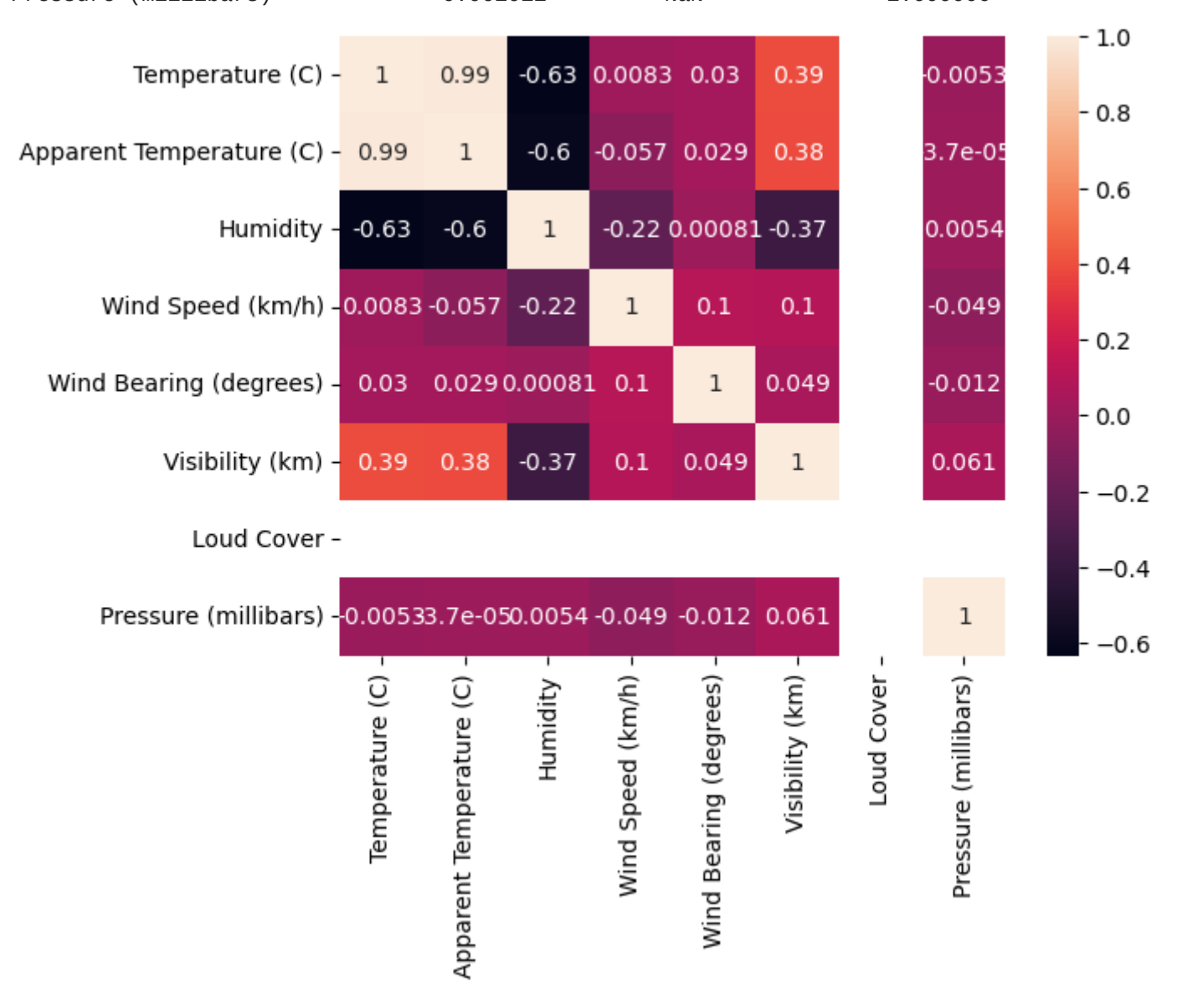
The scatterplot matrix, on the other hand, is a grid of scatterplots that shows the relationship between each pair of variables in the dataset. This plot allows us to visualize the nature of the relationship between variables, whether it is linear, curved, or non-existent, and to identify potential outliers or unusual patterns in the data.

By combining these two types of plots, we can gain a deeper understanding of the relationships between variables in a dataset, and use this information to inform our data analysis or modeling.

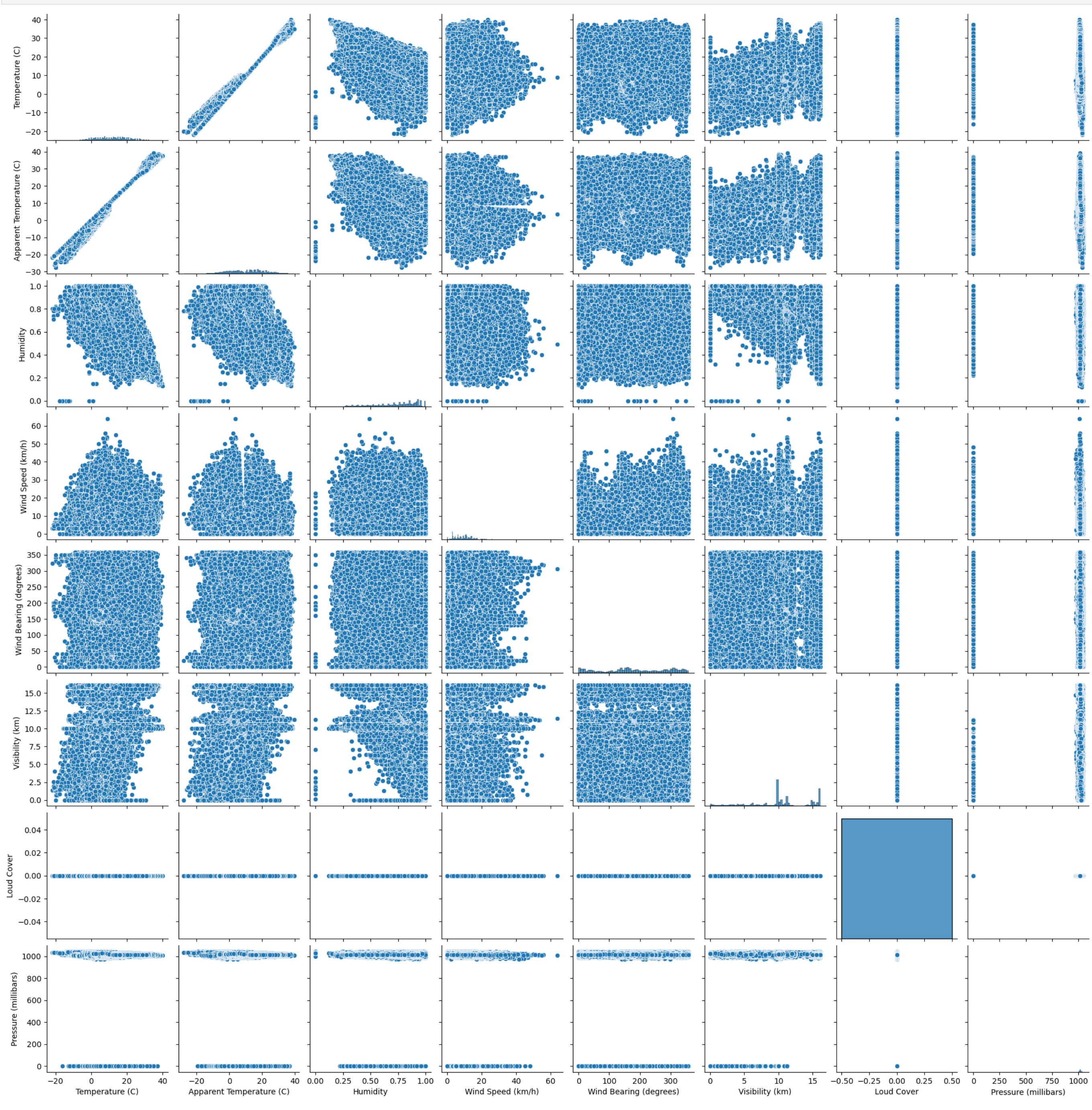
In [ ]:
```
# Import matplotlib and seaborn libraries for data visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

In [18]:
```
# Compute the correlation matrix for the numerical columns in the dataset and print it
print(weather_data.corr())

corr_matrix = weather_data.corr()
sns.heatmap(corr_matrix, annot=True)
plt.show()
```

```
                          Temperature (C)  Apparent Temperature (C)  Humidity  \
Temperature (C)                  1.000000                  0.992637 -0.632148
Apparent Temperature (C)         0.992637                  1.000000 -0.602390
Humidity                        -0.632148                 -0.602390  1.000000
Wind Speed (km/h)                0.008304                 -0.057387 -0.224569
Wind Bearing (degrees)           0.029937                  0.028951  0.000814
Visibility (km)                  0.393719                  0.382589 -0.370511
Loud Cover                            NaN                       NaN       NaN
Pressure (millibars)            -0.005253                 -0.000037  0.005375

                          Wind Speed (km/h)  Wind Bearing (degrees)  \
Temperature (C)                    0.008304                0.029937
Apparent Temperature (C)          -0.057387                0.028951
Humidity                          -0.224569                0.000814
Wind Speed (km/h)                  1.000000                0.104366
Wind Bearing (degrees)             0.104366                1.000000
Visibility (km)                    0.100801                0.048900
Loud Cover                              NaN                     NaN
Pressure (millibars)              -0.049453               -0.011629

                          Visibility (km)  Loud Cover  Pressure (millibars)
Temperature (C)                  0.393719         NaN             -0.005253
Apparent Temperature (C)         0.382589         NaN             -0.000037
Humidity                        -0.370511         NaN              0.005375
Wind Speed (km/h)                0.100801         NaN             -0.049453
Wind Bearing (degrees)           0.048900         NaN             -0.011629
Visibility (km)                  1.000000         NaN              0.061011
Loud Cover                            NaN         NaN                   NaN
Pressure (millibars)             0.061011         NaN              1.000000
```



In [19]:
```
# Create a scatterplot matrix to visualize the relationships between the variables in the dataset
sns.pairplot(weather_data, kind="scatter")
plt.show()
```



The dataset is split into training and testing sets, and the linear regression model is trained using the training set. The trained model is then used to make predictions on the testing set, and the actual and predicted values are plotted for both the training and testing sets.

In [7]:
```
# Select the 'Apparent Temperature (C)' column as input (X) and the 'Temperature (C)' column as output (y)

X=weather_data[['Apparent Temperature (C)']].values
y=weather_data[['Temperature (C)']].values
```

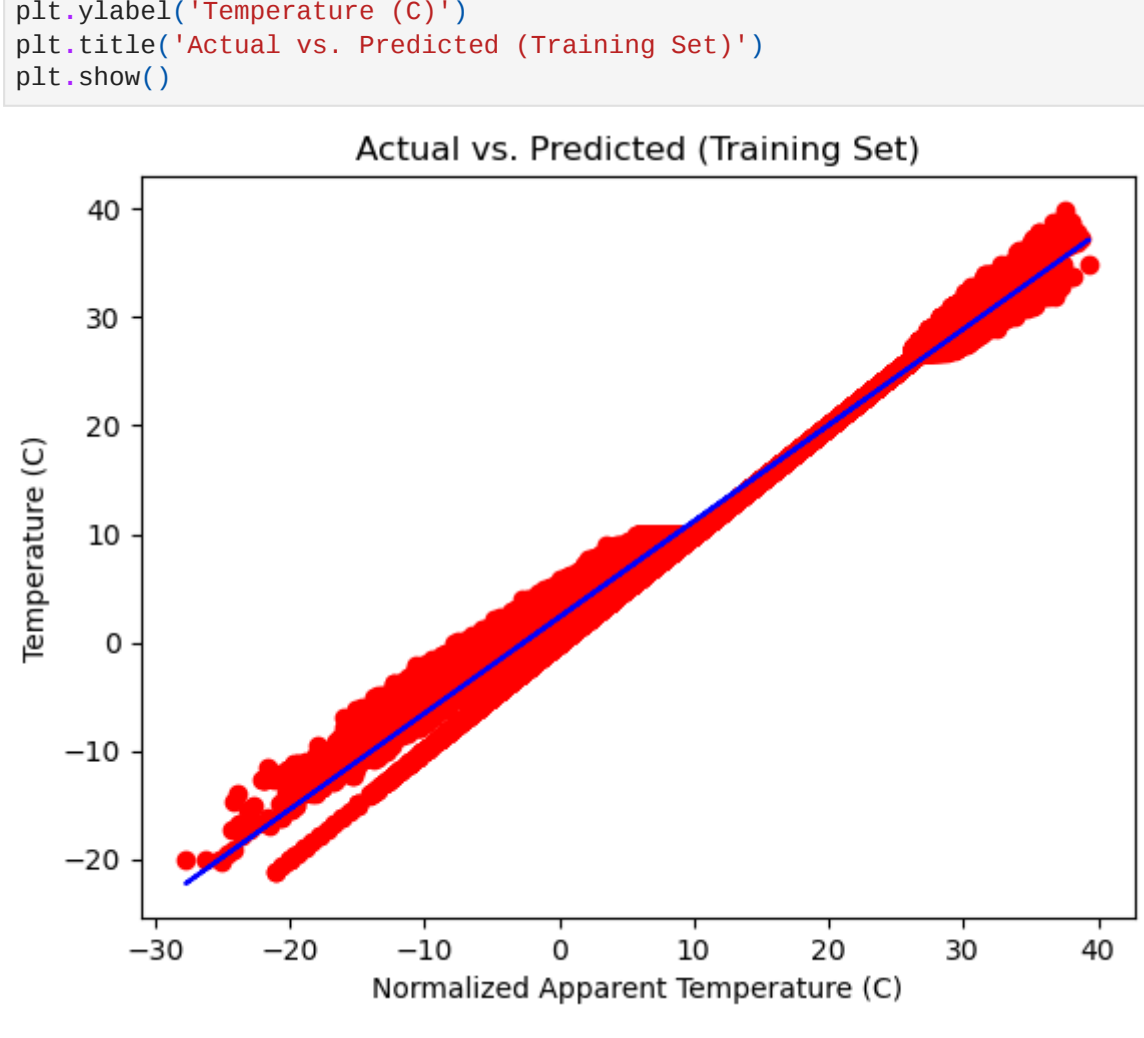
In [8]:
```
# Import train_test_split and LinearRegression classes from the sklearn library

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Split the dataset into training and testing sets, with a 70-30 ratio, and use a random_state of 0 for reproducibility
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Create a LinearRegression model
regressor = LinearRegression()

# Train the model using the training sets
regressor.fit(x_train, y_train)

# Make predictions using the testing set
y_pred = regressor.predict(x_test)
y_pred_train=regressor.predict(x_train)
```

The purpose of the first plot is to visualize the performance of a linear regression model on a training set of data.

The scatter plot shows the actual temperature values in the training set (y_train) against the corresponding normalized apparent temperature values (x_train), plotted as red dots.

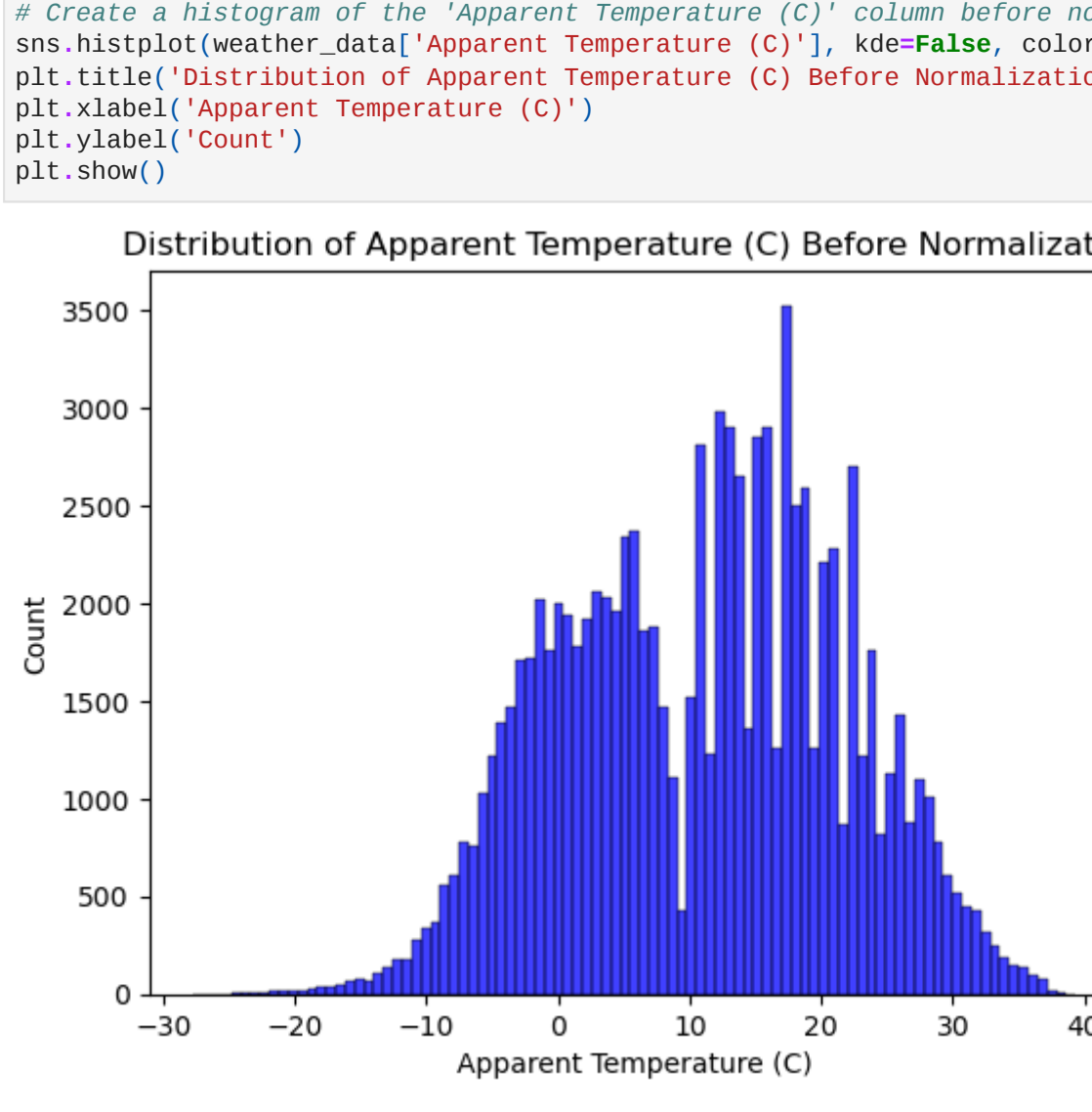
The blue line represents the predicted temperature values (y_pred_train) generated by the linear regression model. This plot allows us to visually assess how well the model fits the training data by comparing the predicted values with the actual values.

In [9]:
```
# Create a scatter plot of the training data with the model's predictions plotted on top
plt.scatter(x_train, y_train, color='red')
plt.plot(x_train, y_pred_train, color='blue')
plt.xlabel('Normalized Apparent Temperature (C)')
plt.ylabel('Temperature (C)')
plt.title('Actual vs. Predicted (Training Set)')
plt.show()
```



The second plot is a histogram that shows the distribution of the apparent temperature (C) variable in the dataset before normalization. The x-axis shows the range of apparent temperature values, and the y-axis shows the count of how many data points fall within each range. This plot allows us to visualize the shape of the distribution, whether it is skewed or symmetric, and identify any outliers or unusual patterns in the data.

In [10]:
```
# Create a histogram of the 'Apparent Temperature (C)' column before normalization
sns.histplot(weather_data['Apparent Temperature (C)'], kde=False, color='blue')
plt.title('Distribution of Apparent Temperature (C) Before Normalization')
plt.xlabel('Apparent Temperature (C)')
plt.ylabel('Count')
plt.show()
```



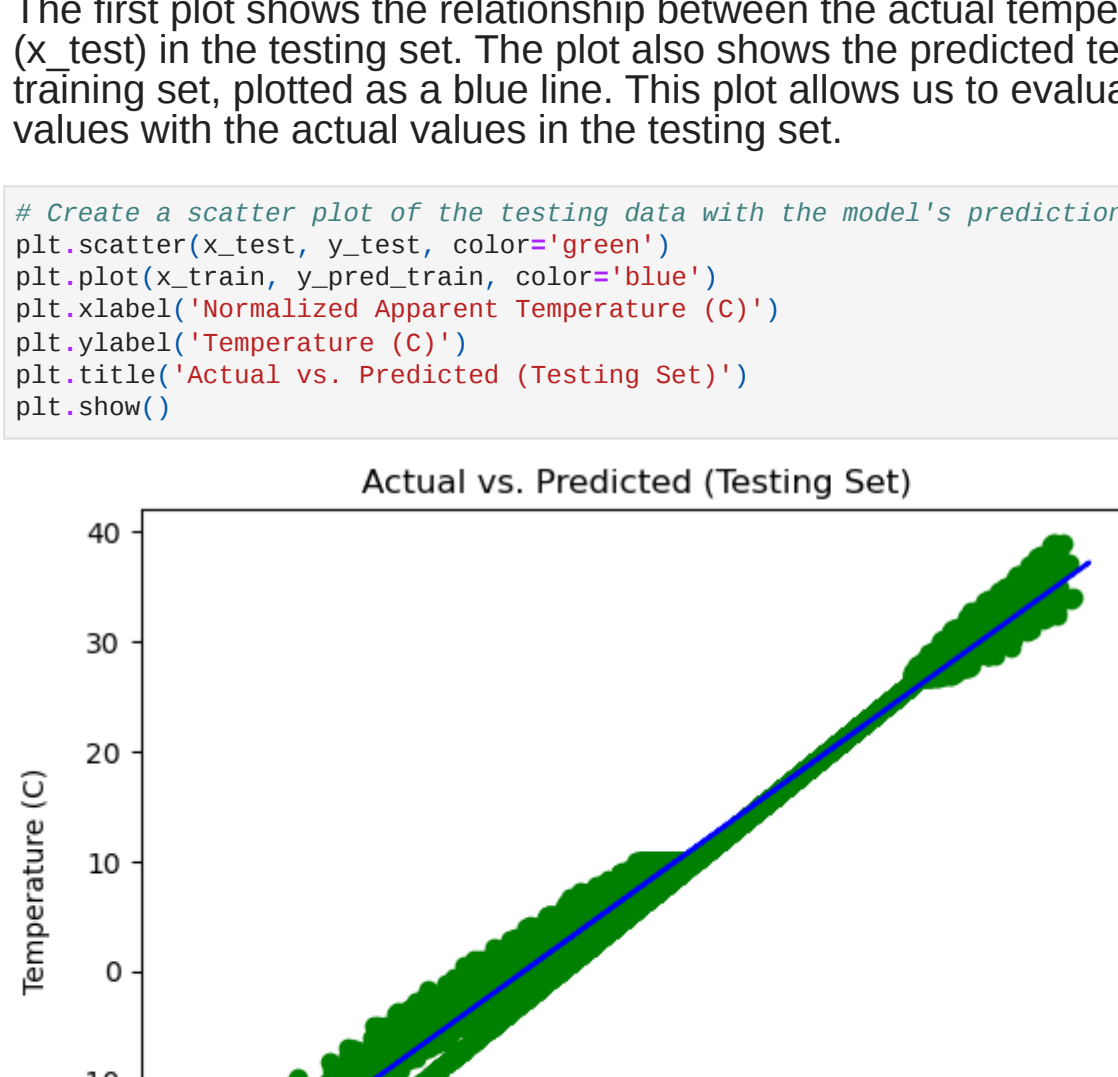
the apparent temperature feature is normalized using MinMaxScaler, and the model is trained and tested on the normalized dataset. The histograms of the apparent temperature before and after normalization are also plotted. Finally, the performance of the model is evaluated using the R-squared score and the root mean square error.

In [11]:
```
# Import StandardScaler and MinMaxScaler classes from the sklearn library
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Create a MinMaxScaler object to normalize the 'Apparent Temperature (C)' column
scaler = MinMaxScaler()
X_Normaliz = scaler.fit_transform(X)
```

In [12]:
```
# Split the normalized dataset into training and testing sets, with a 70-30 ratio, and use a random_state of 0 for reproducibility
x_train, x_test, y_train, y_test = train_test_split(X_Normaliz, y, test_size=0.3, random_state=0)

# Create a LinearRegression model
regressor = LinearRegression()

# Train the model using the training sets
regressor.fit(x_train, y_train)

# Make predictions using the testing set
y_pred = regressor.predict(x_test)
y_pred_train=regressor.predict(x_train)
```

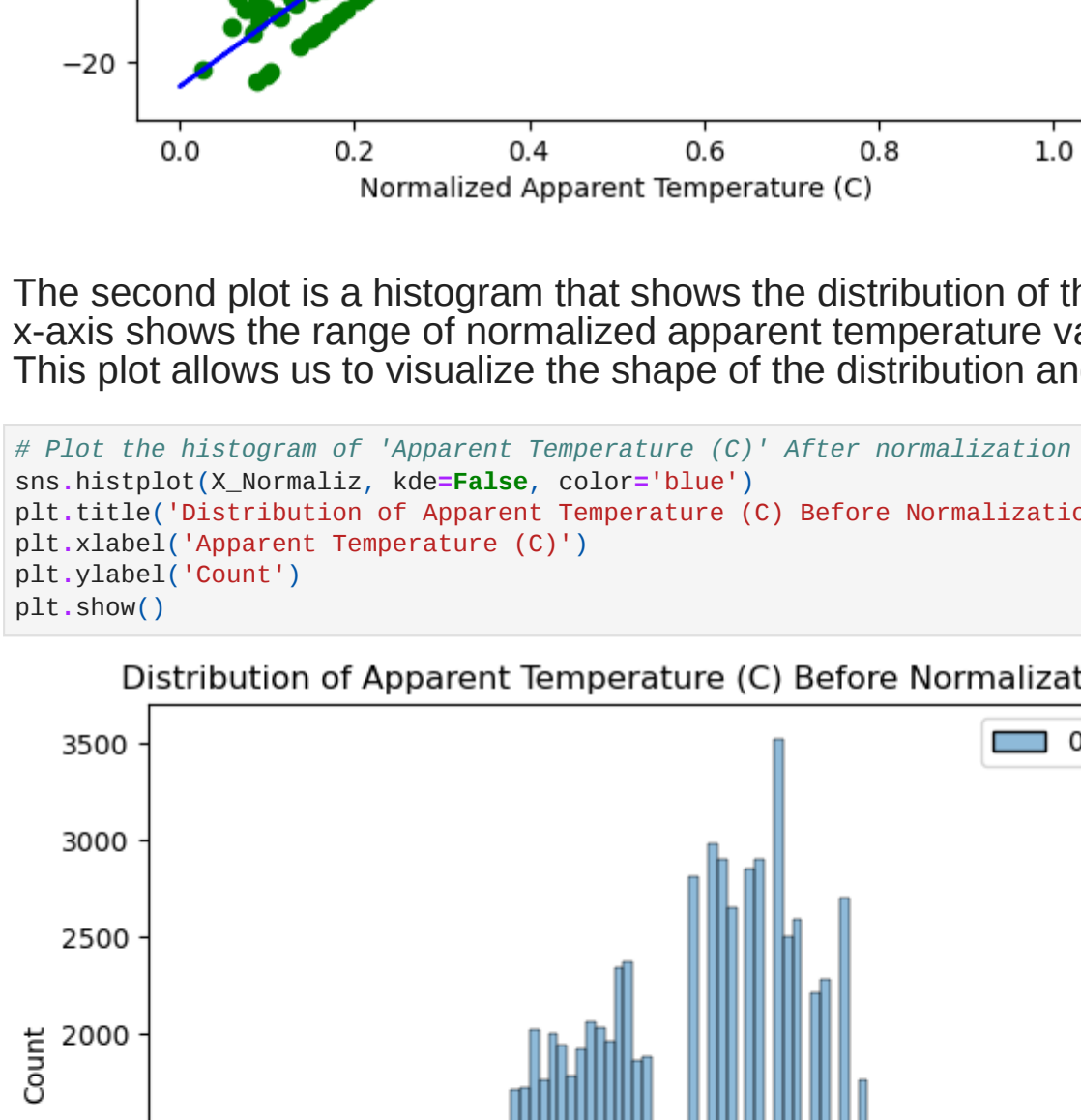
The first plot shows the relationship between the actual temperature values (y_test) and the corresponding normalized apparent temperature values (x_test) in the testing set. The plot also shows the predicted temperature values (y_pred_train) generated by the linear regression model on the training set, plotted as a blue line. This plot allows us to evaluate the model's ability to generalize to new, unseen data by comparing the predicted values with the actual values in the testing set.

In [13]:
```
# Create a scatter plot of the testing data with the model's predictions plotted on top
plt.scatter(x_test, y_test, color='green')
plt.plot(x_train, y_pred_train, color='blue')
plt.xlabel('Normalized Apparent Temperature (C)')
plt.ylabel('Temperature (C)')
plt.title('Actual vs. Predicted (Testing Set)')
plt.show()
```



The second plot is a histogram that shows the distribution of the normalized apparent temperature (C) variable in the dataset after normalization. The x-axis shows the range of normalized apparent temperature values, and the y-axis shows the count of how many data points fall within each range. This plot allows us to visualize the shape of the distribution and identify any outliers or unusual patterns in the data after normalization.

In [14]:
```
# Plot the histogram of 'Apparent Temperature (C)' After normalization
sns.histplot(X_Normaliz, kde=False, color='blue')
plt.title('Distribution of Apparent Temperature (C) Before Normalization')
plt.xlabel('Apparent Temperature (C)')
plt.ylabel('Count')
plt.show()
```



Overall, the previous plots help us understand the performance of the linear regression model on the testing set and the distribution of the data after normalization.

In [15]:
```
# Evaluate the performance of the model using R-squared and RMSE

from sklearn.metrics import r2_score
import numpy as np

# Evaluate the performance of the model using R-squared
r2 = r2_score(y_test, y_pred)
print("R-squared score:", r2)
root_mean_square_error=np.sqrt(np.mean(np.square(y_pred-y_test)))
print('Root Mean Square Error',root_mean_square_error)
```

```
R-squared score: 0.9853093837336178
Root Mean Square Error 1.1592596440552216
```

The conclusion is that the linear regression model performs well in predicting the temperature based on the apparent temperature, as indicated by the high R-squared score and the low root mean square error. Normalizing the apparent temperature feature does not significantly affect the model's performance. However, it is essential to note that this model is built based on a specific dataset, and the performance may differ for other datasets.