Import excel files

In [51]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import *
from sklearn.linear_model import *
from sklearn.metrics import mean_absolute_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
import numpy as np
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score



# Load data into a pandas dataframe
df = pd.read_excel('EDA.xlsx')
```

In [2]:
```
df
```

Out[2]:

	created_at	entry_id	Temp	Humid
0	2023-01-07 13:51:53 UTC	13	29	56
1	2023-01-07 13:52:09 UTC	14	30	51
2	2023-01-07 13:52:24 UTC	15	30	50
3	2023-01-07 13:52:39 UTC	16	30	51
4	2023-01-07 13:52:55 UTC	17	52	14
...	...	...	...	...
95	2023-01-07 14:20:05 UTC	108	19	80
96	2023-01-07 14:20:20 UTC	109	19	75
97	2023-01-07 14:20:35 UTC	110	17	76
98	2023-01-07 14:20:50 UTC	111	16	81
99	2023-01-07 14:21:07 UTC	112	16	94

100 rows × 4 columns

Check for missing values

In [3]:
```
print(df.isnull().sum())
```

```
created_at    0
entry_id      0
Temp          0
Humid         0
dtype: int64
```

Descriptive statistics

In [4]:
```
print(df.describe())
```

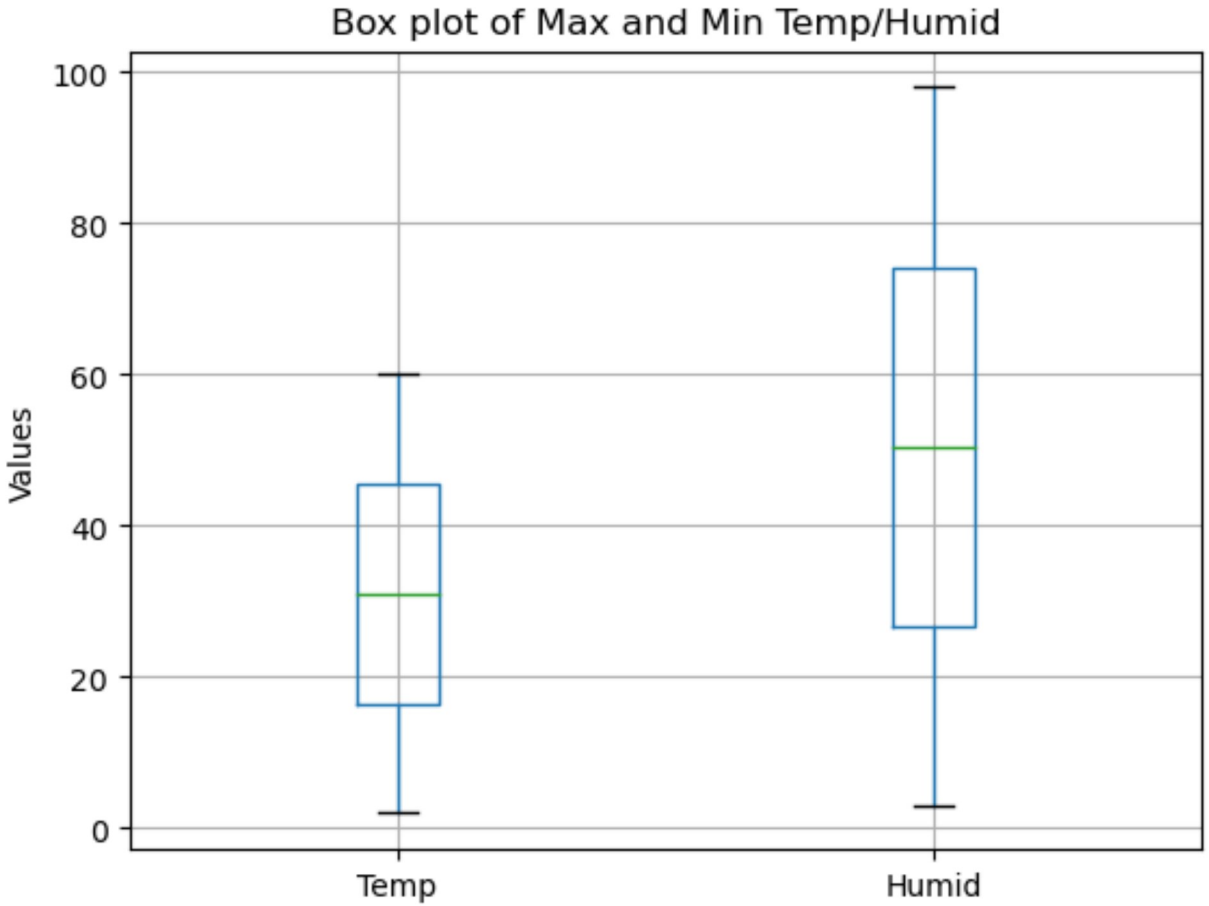
```
         entry_id        Temp       Humid
count  100.000000  100.000000  100.000000
mean    62.500000   33.280000   49.130000
std     29.011492   14.570678   28.640458
min     13.000000    2.000000    3.000000
25%     37.750000   24.000000   29.500000
50%     62.500000   30.000000   52.000000
75%     87.250000   42.250000   69.500000
max    112.000000   60.000000   98.000000
```

Max and Min

In [5]:
```
print('Max of enter_id -> ',max(df.entry_id))
print('Min of enter_id -> ',min(df.entry_id))

print('\nMax of Temp -> ',max(df.Temp))
print('Min of Temp -> ',min(df.Temp))

print('\nMax of Humid -> ',max(df.Humid))
print('Min of Humid -> ',min(df.Humid))
```

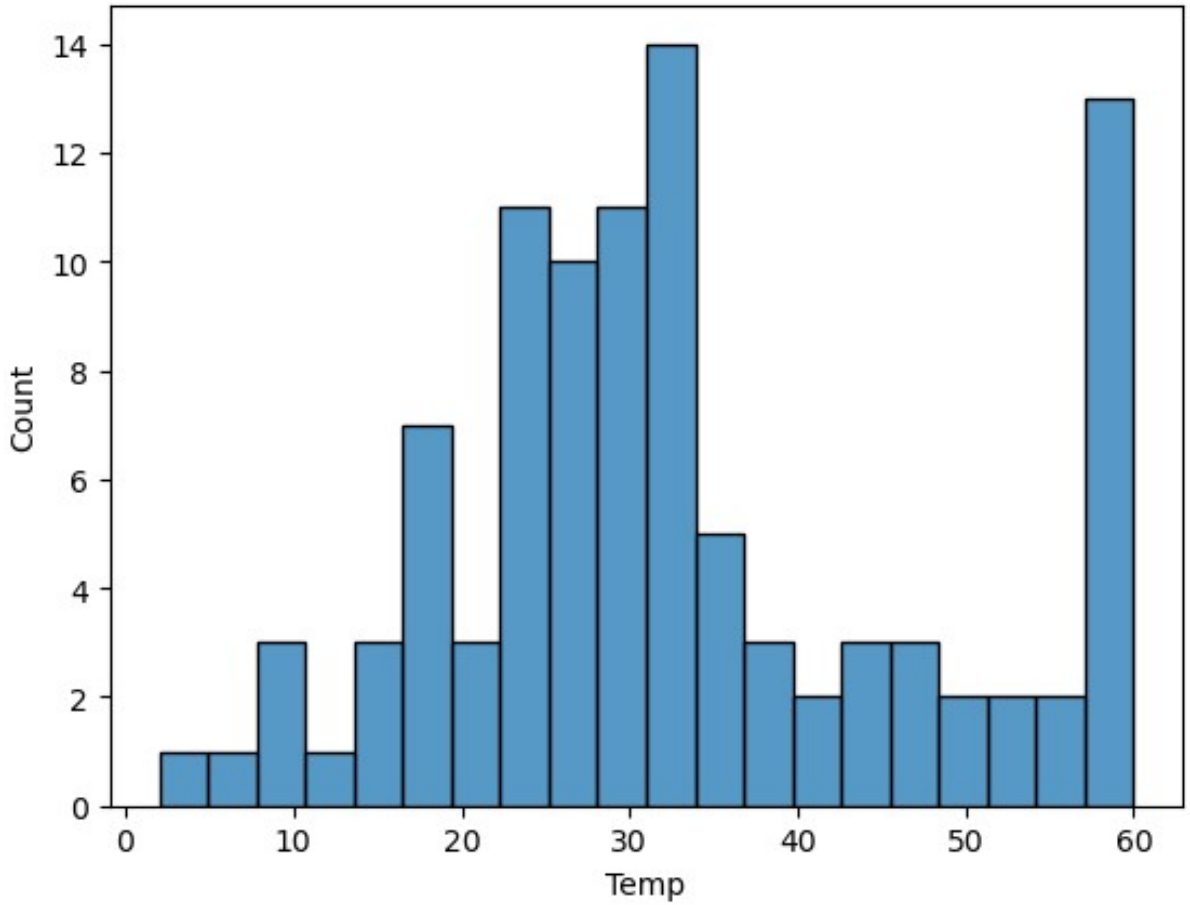
```
Max of enter_id ->  112
Min of enter_id ->  13

Max of Temp ->  60
Min of Temp ->  2

Max of Humid ->  98
Min of Humid ->  3
```

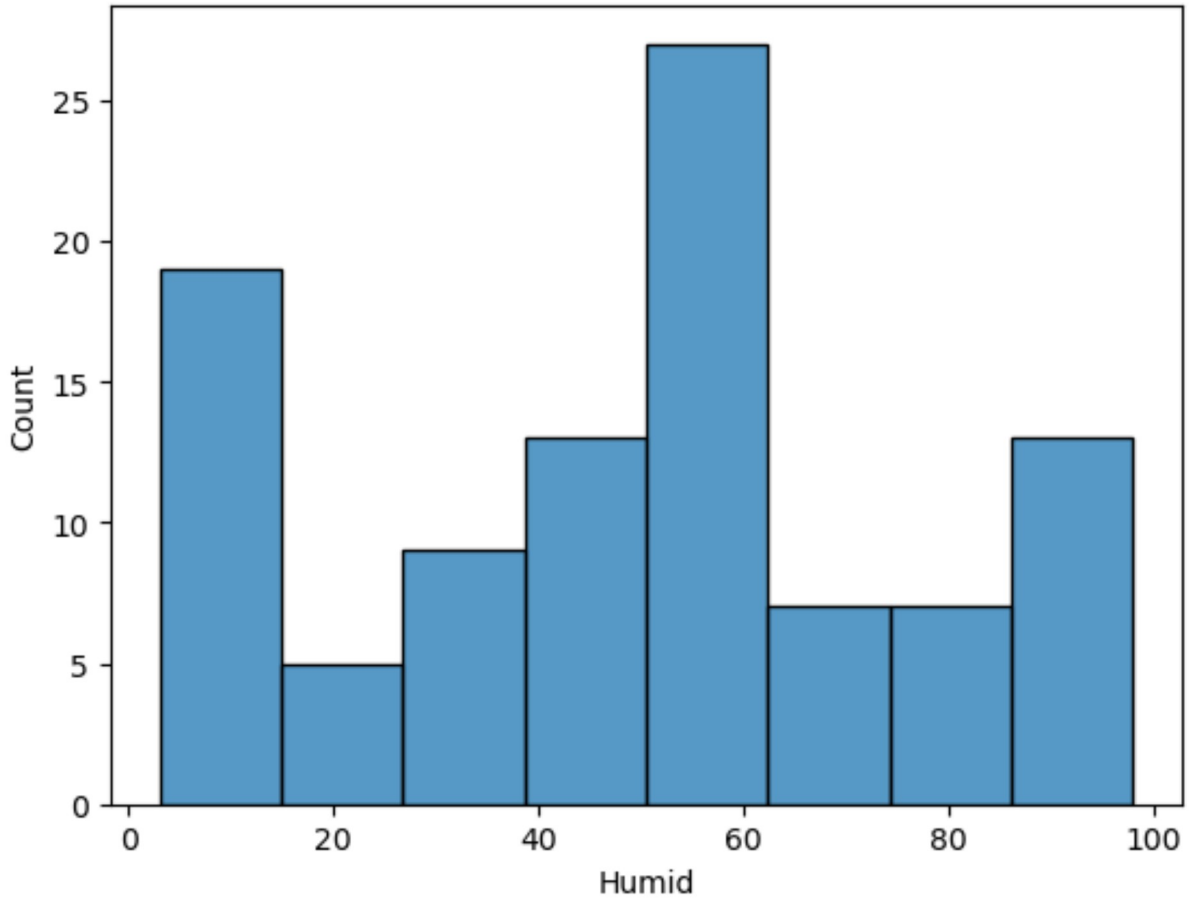
In [6]:
```
temp_max = max(df.Temp)
temp_min = min(df.Temp)
humid_max = max(df.Humid)
humid_min = min(df.Humid)

data = {'Temp': [temp_max, temp_min], 'Humid': [humid_max, humid_min]}

df_box = pd.DataFrame(data)

df_box.boxplot()
plt.title('Box plot of Max and Min Temp/Humid')
plt.ylabel('Values')
plt.show()
```



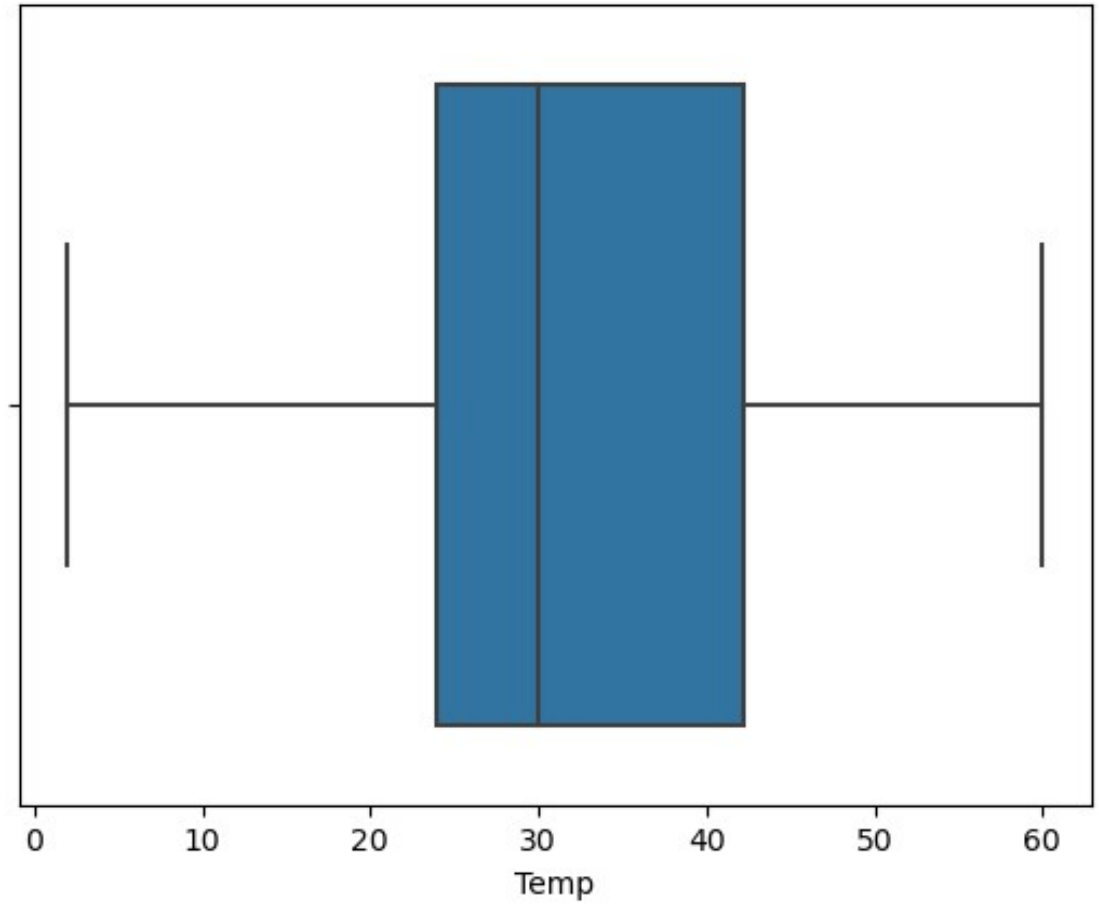
Univariate analysis

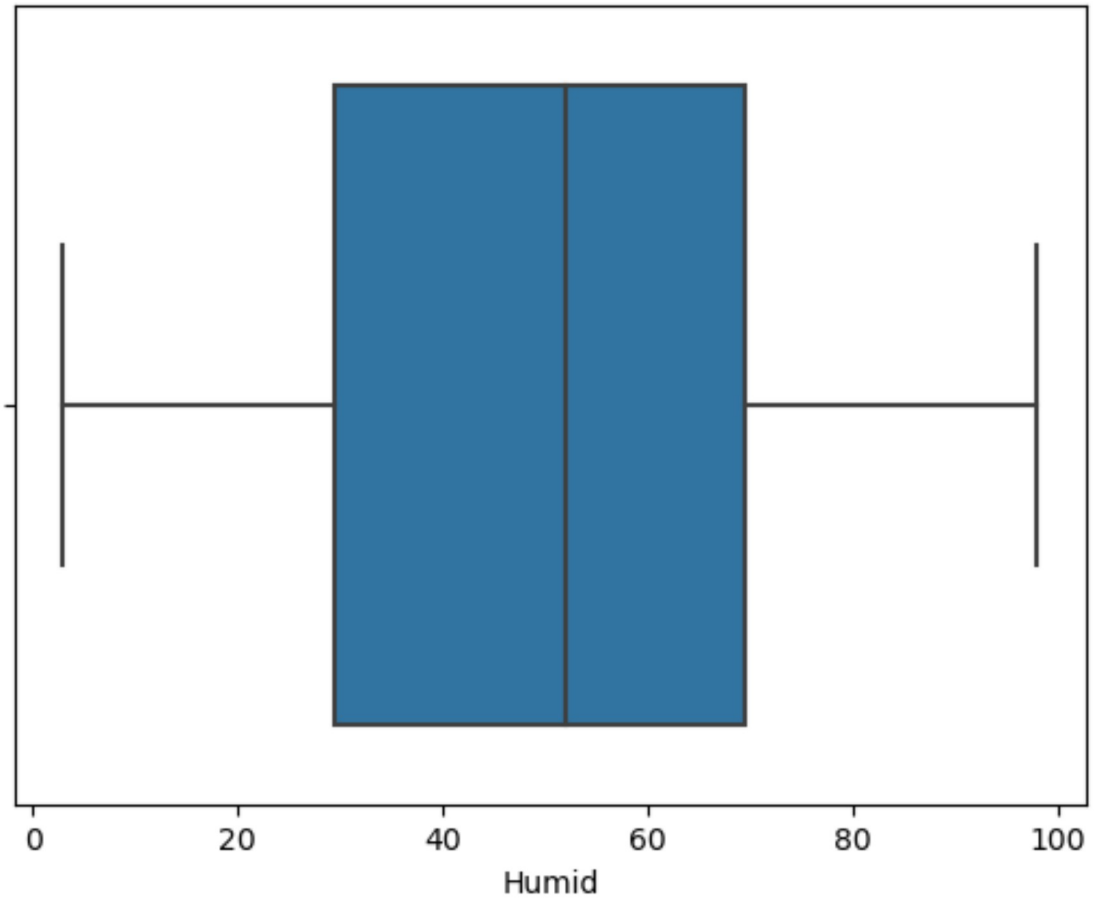
In [7]:
```
sns.histplot(data=df, x='Temp', bins=20)
plt.show()

sns.histplot(data=df, x='Humid')
plt.show()
```


Count
Temp
0
2
4
6
8
10
12
14
0
10
20
30
40
50
60



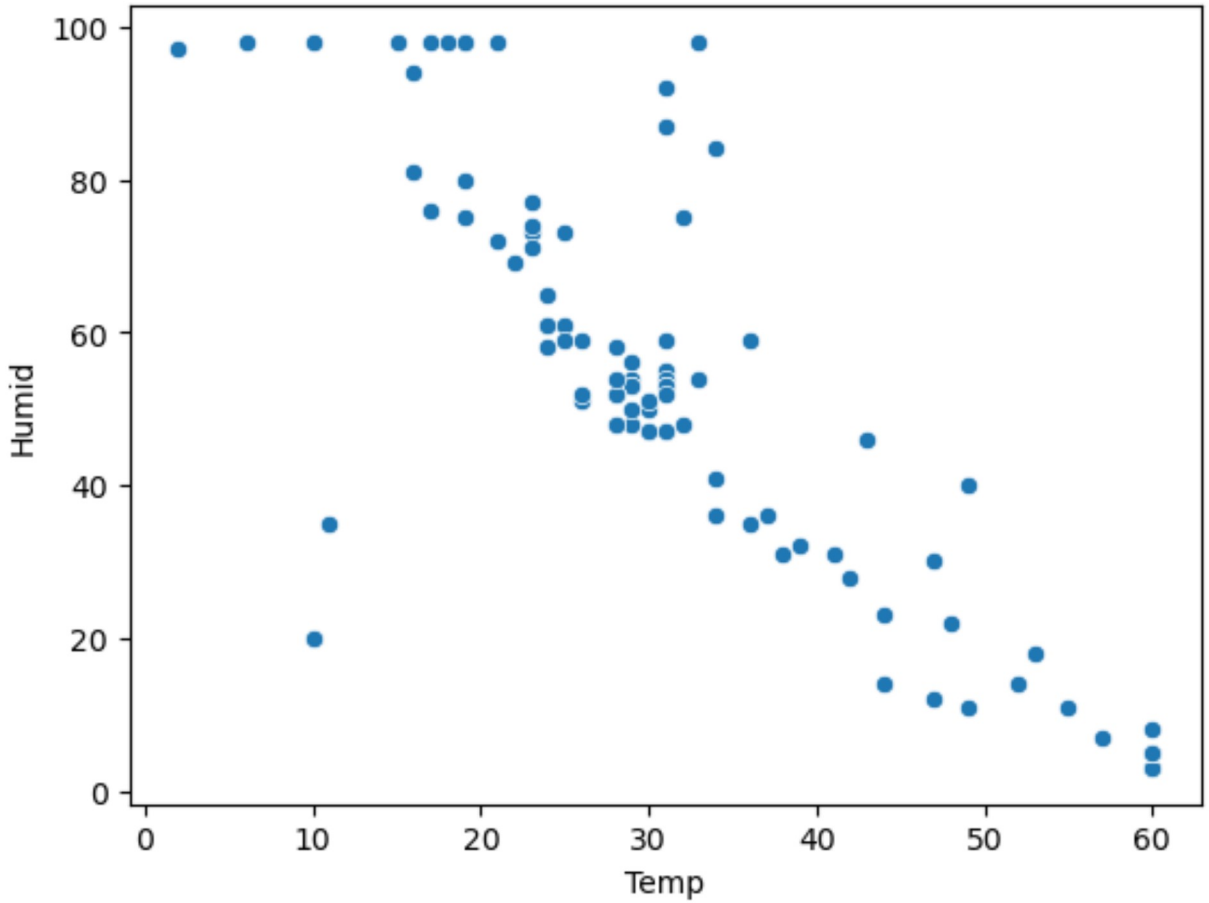
In [8]:
```
sns.boxplot(data=df, x='Temp')
plt.show()

sns.boxplot(data=df, x='Humid')
plt.show()
```


0
10
20
30
40
50
60
Temp

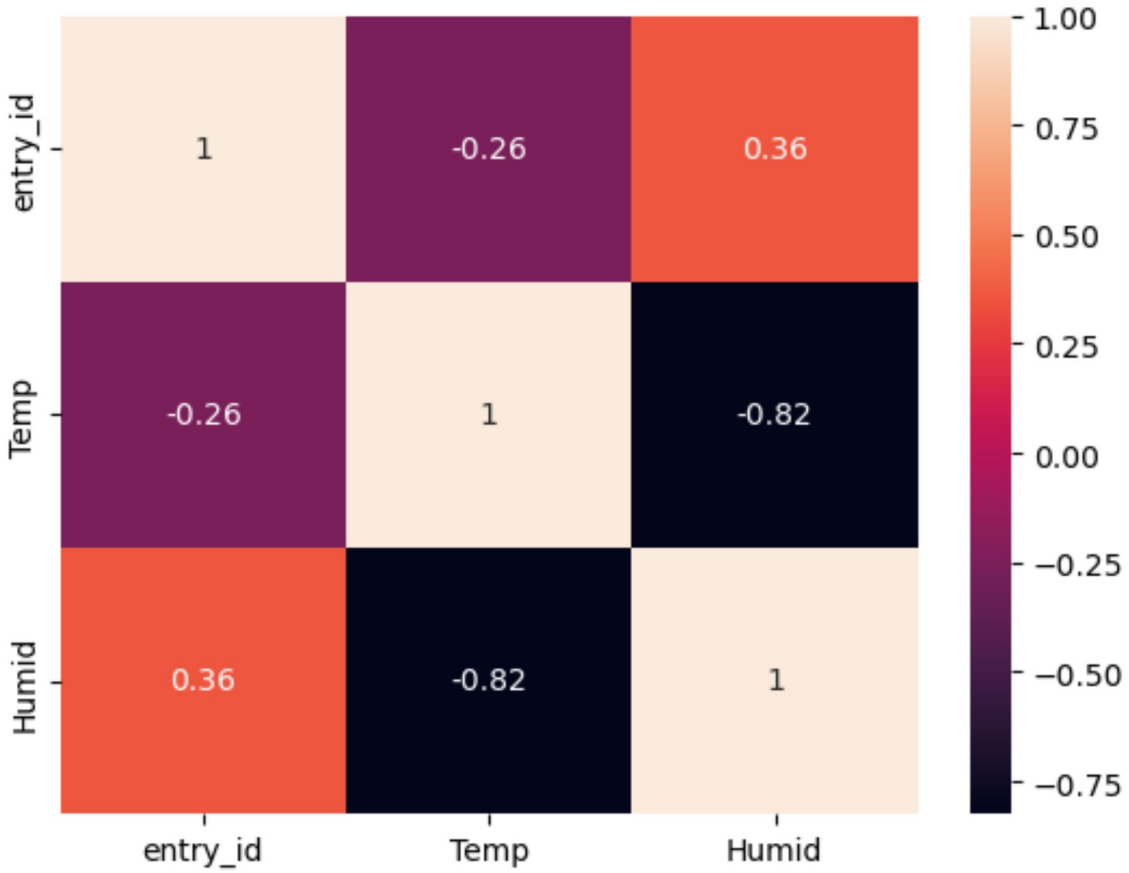


Bivariate analysis

In [9]:
```
sns.scatterplot(data=df, x='Temp', y='Humid')
plt.show()
```

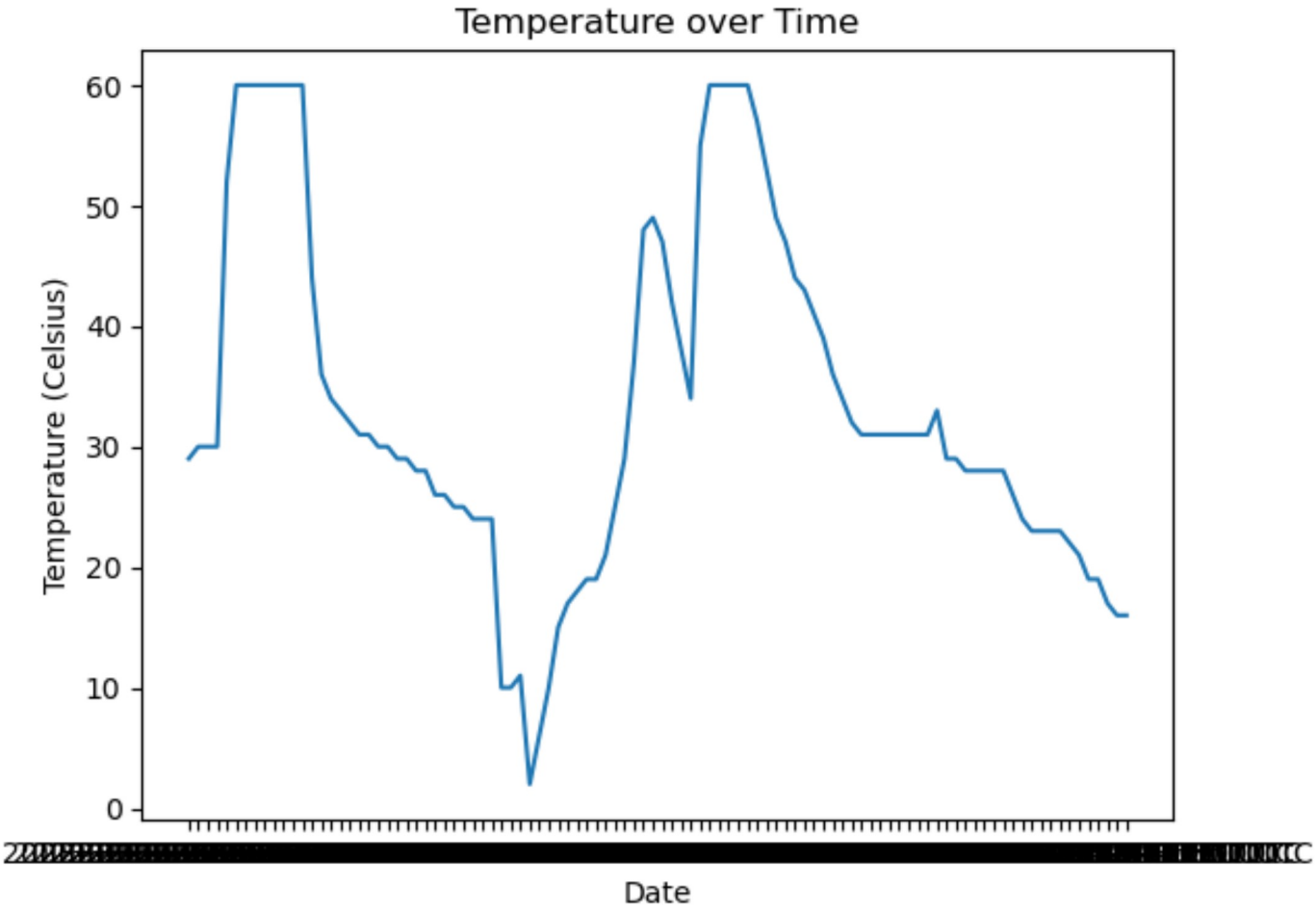


In [10]:
```
sns.heatmap(df.corr(), annot=True)
plt.show()
```



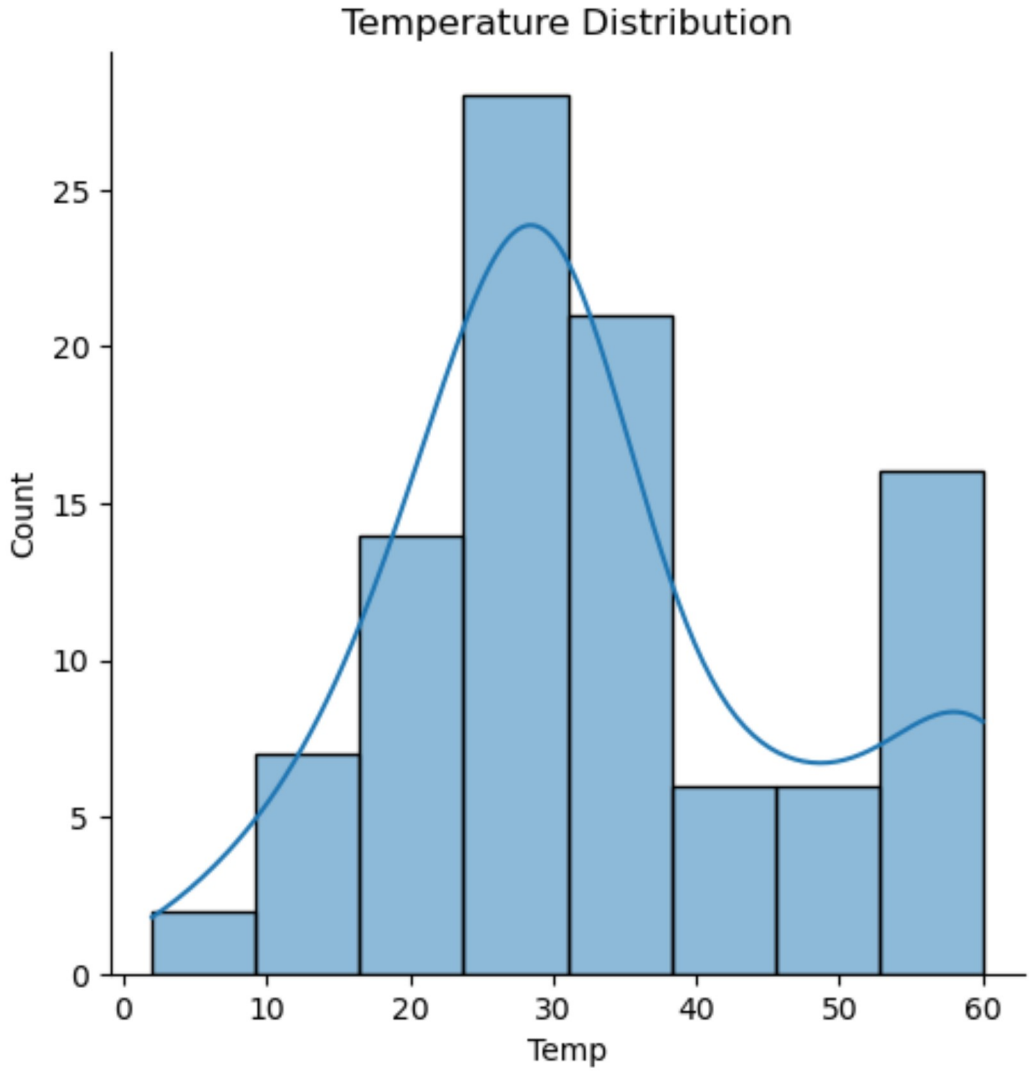
Time Series Analysis

In [11]:
```
plt.plot(df['created_at'], df['Temp'])
plt.title('Temperature over Time')
plt.xlabel('Date')
plt.ylabel('Temperature (Celsius)')
plt.show()
```

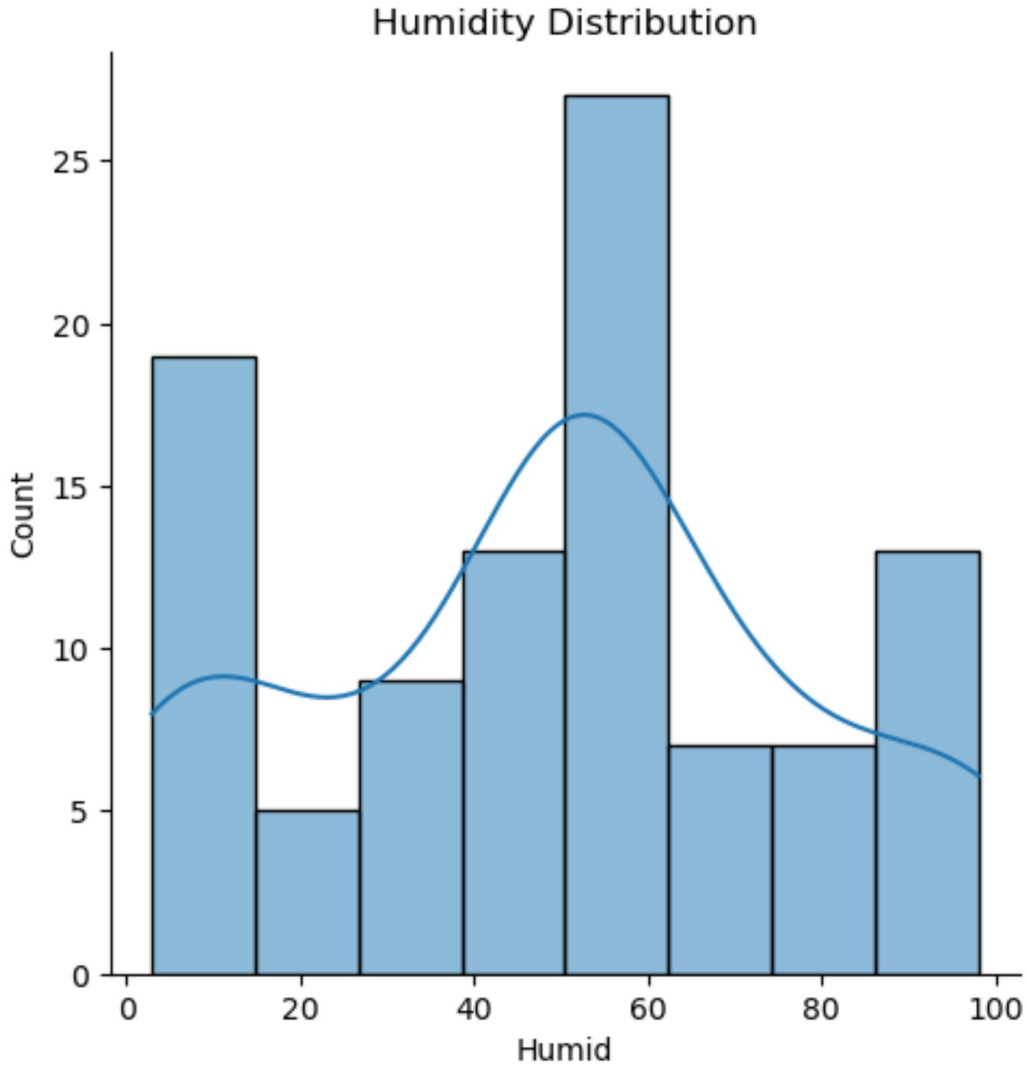


Distribution Plot

In [12]:
```
sns.displot(df, x='Temp', kde=True)
plt.title('Temperature Distribution')
plt.show()
```

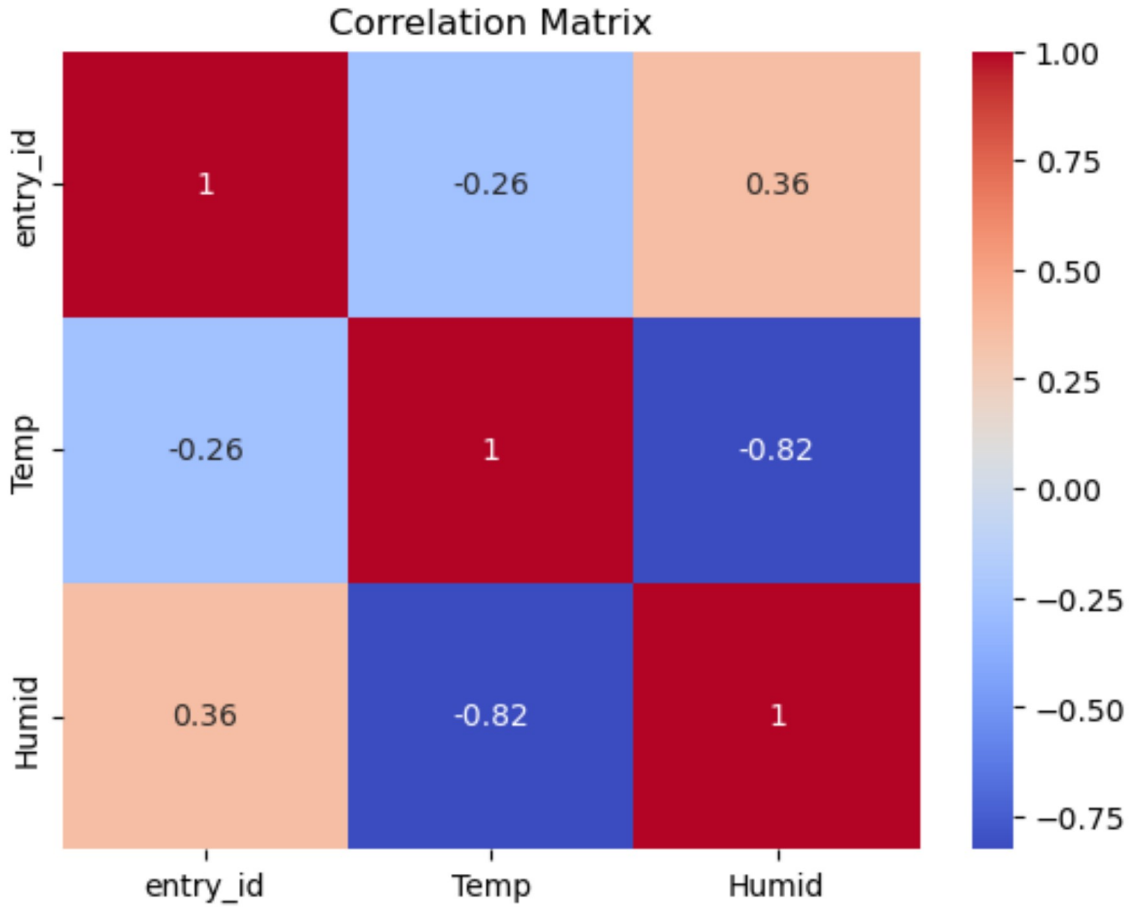


In [13]:
```
sns.displot(df, x='Humid', kde=True)
plt.title('Humidity Distribution')
plt.show()
```



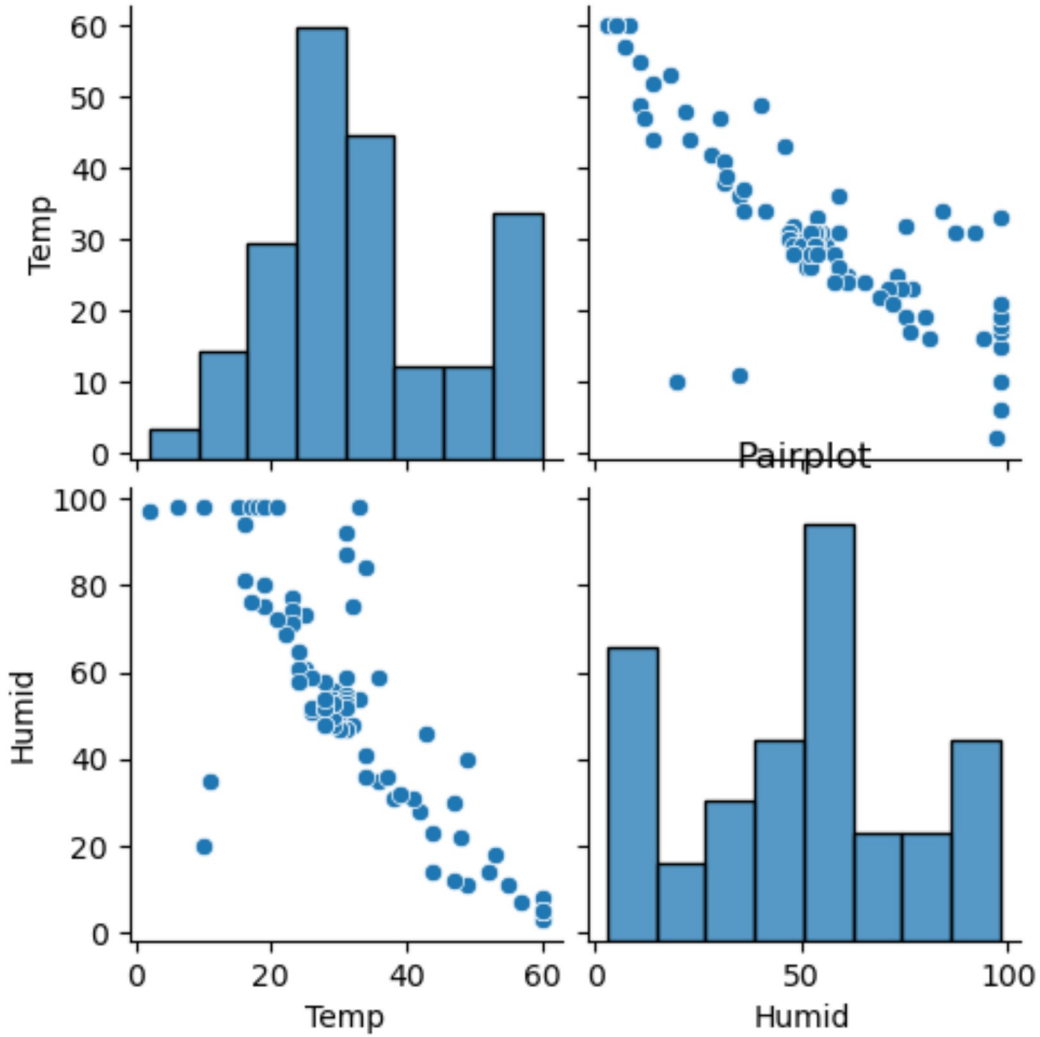
Correlation Analysis

In [14]:
```
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



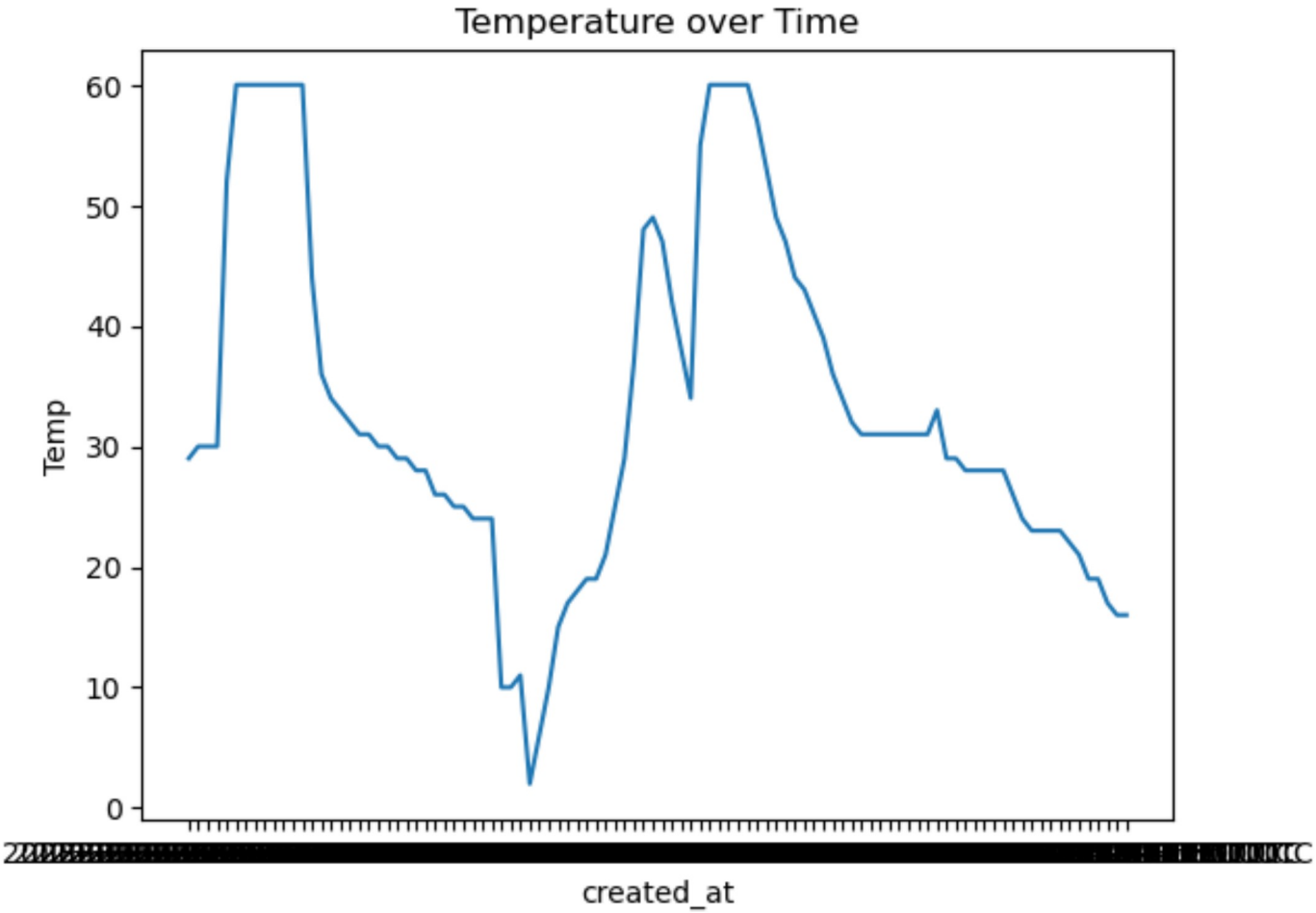
Pairplot

In [15]:
```
sns.pairplot(df, vars=['Temp', 'Humid'], kind='scatter')
plt.title('Pairplot')
plt.show()
```

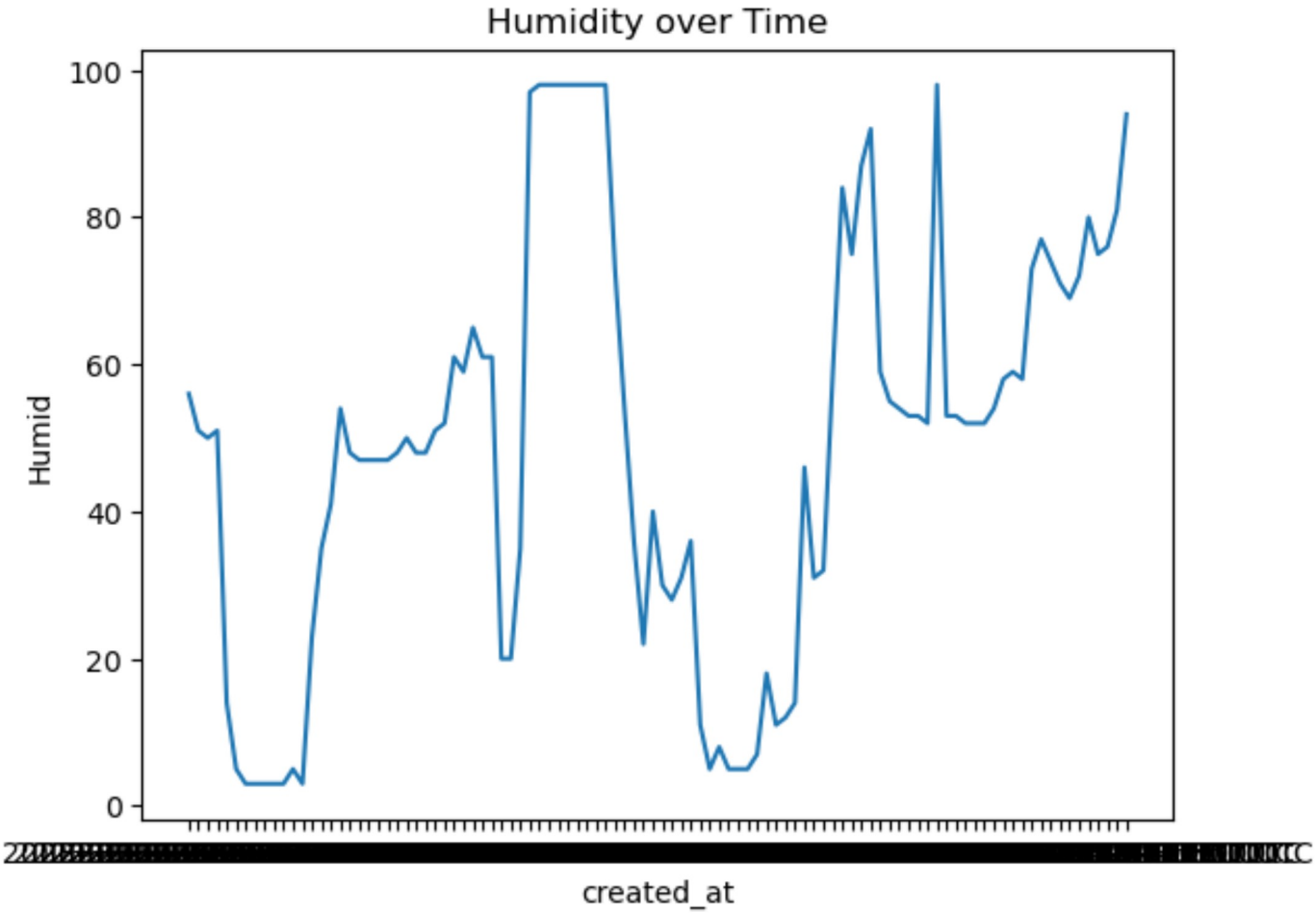


Lineplot

In [16]:
```
sns.lineplot(data=df, x='created_at', y='Temp')
plt.title('Temperature over Time')
plt.show()
```

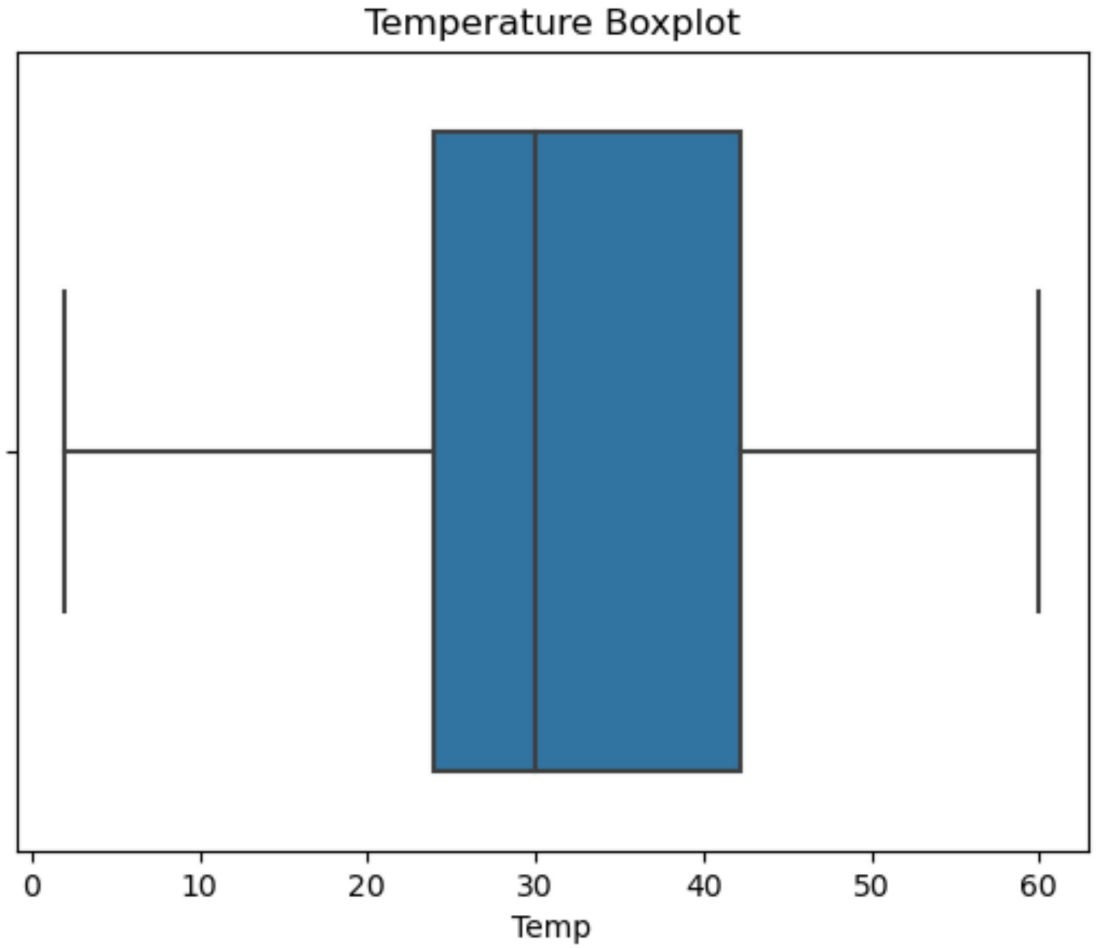


In [17]:
```
sns.lineplot(data=df, x='created_at', y='Humid')
plt.title('Humidity over Time')
plt.show()
```

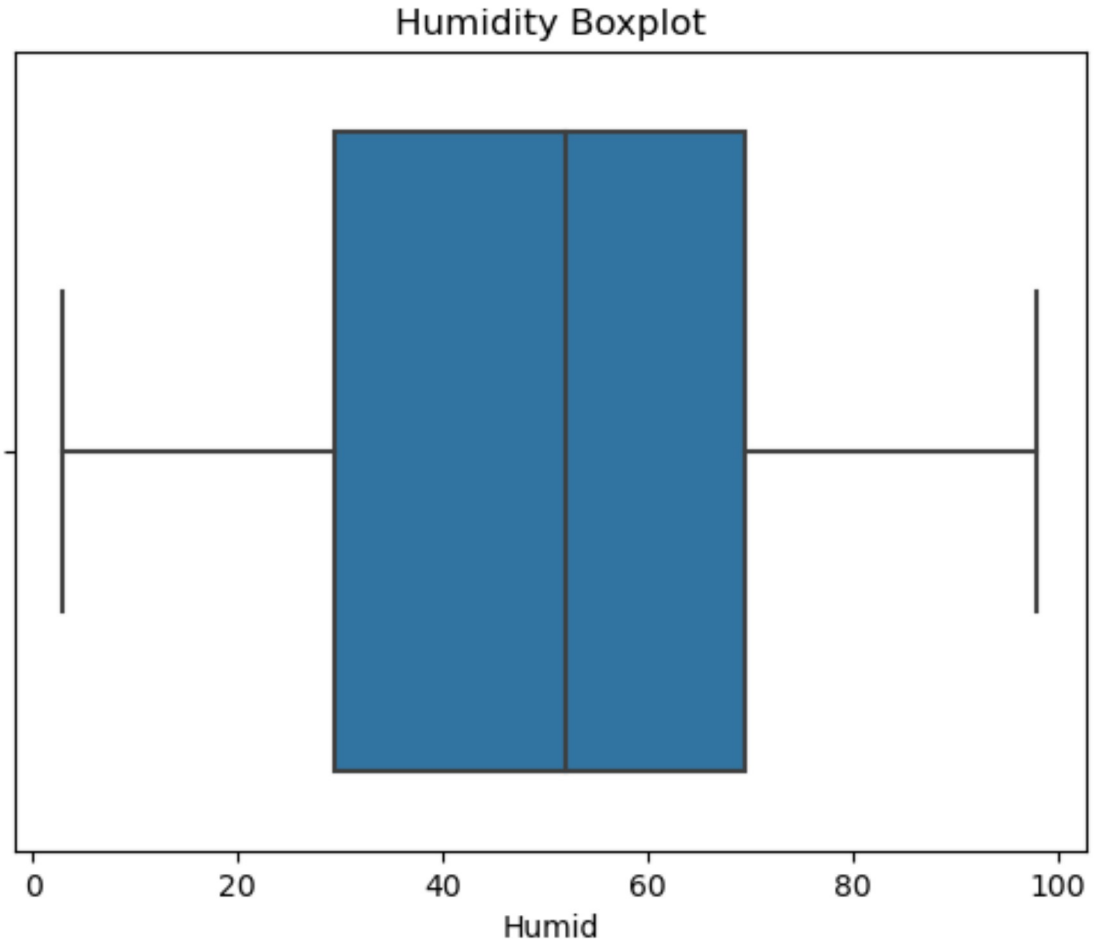


Boxplot

In [18]:
```
sns.boxplot(data=df, x='Temp')
plt.title('Temperature Boxplot')
plt.show()
```

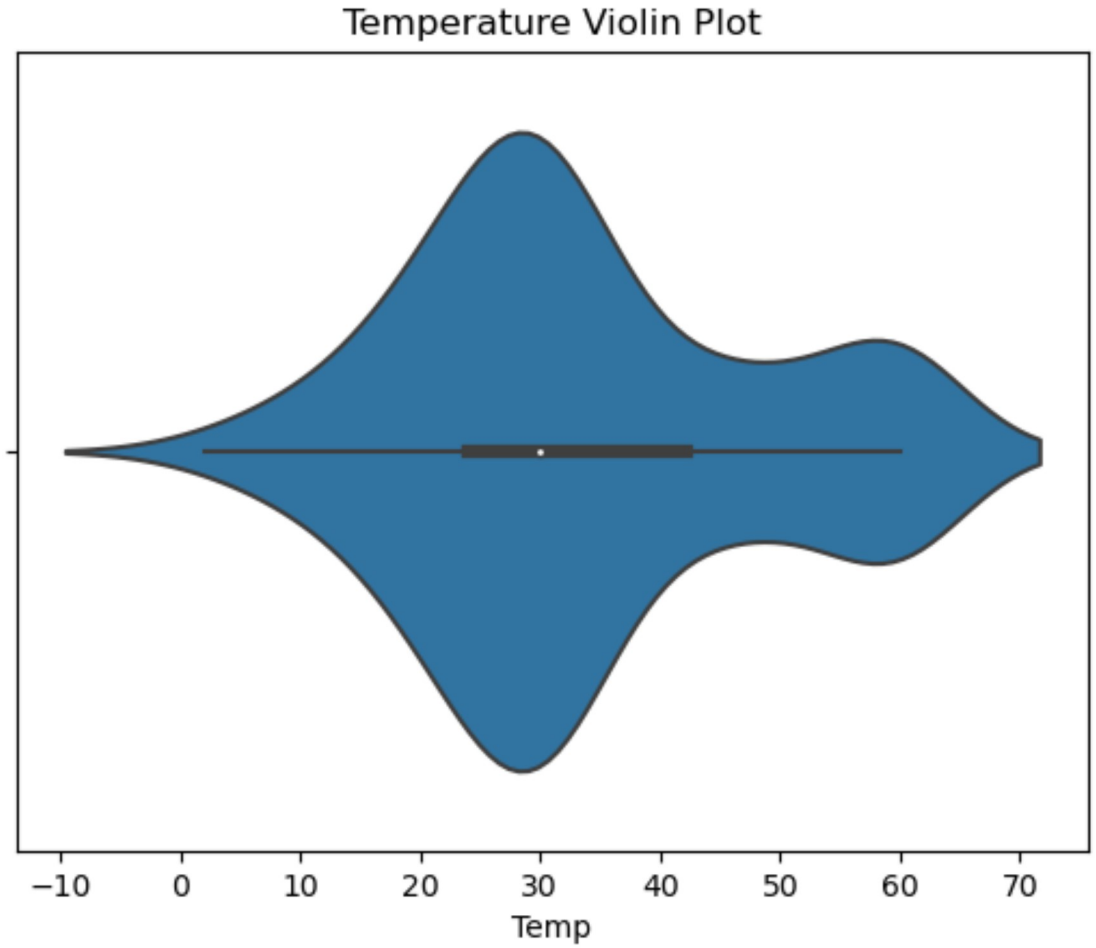


In [19]:
```
sns.boxplot(data=df, x='Humid')
plt.title('Humidity Boxplot')
plt.show()
```

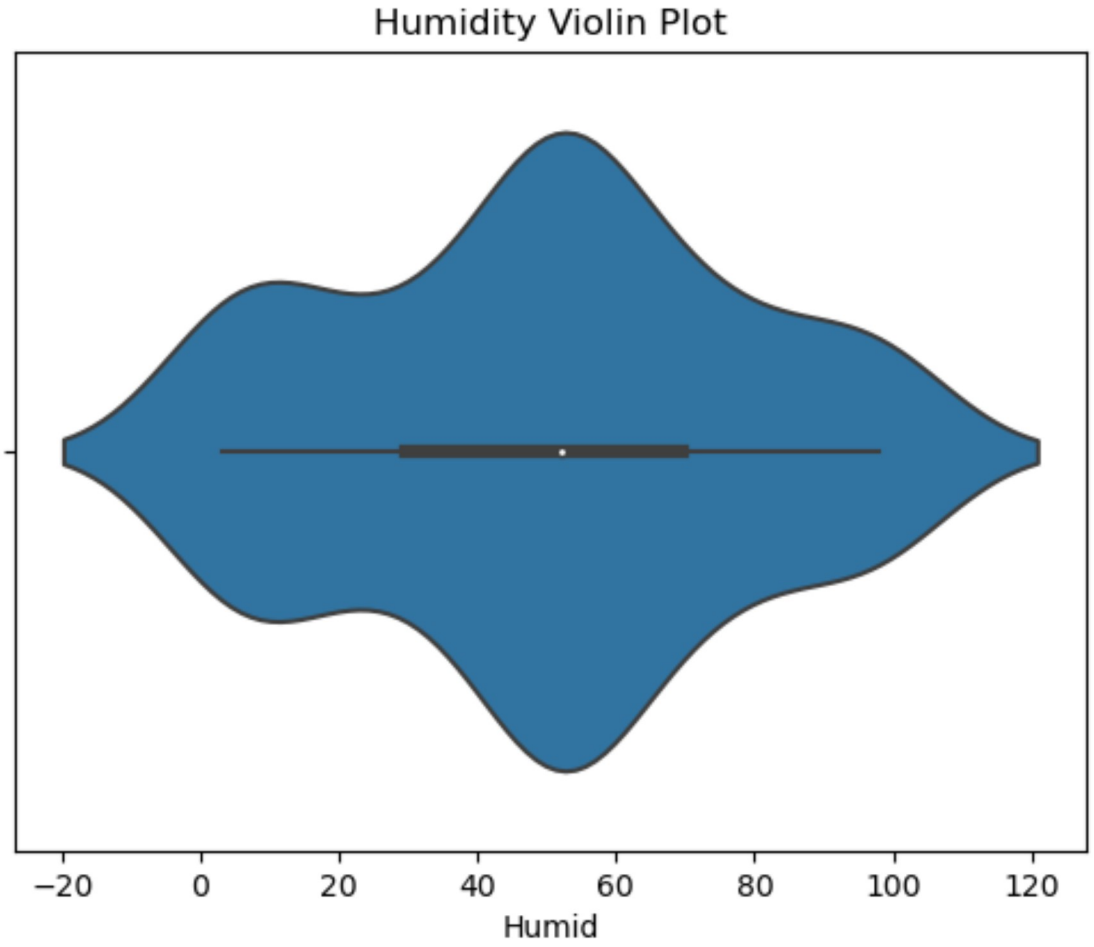


Violin Plot

In [20]:
```
sns.violinplot(data=df, x='Temp')
plt.title('Temperature Violin Plot')
plt.show()
```

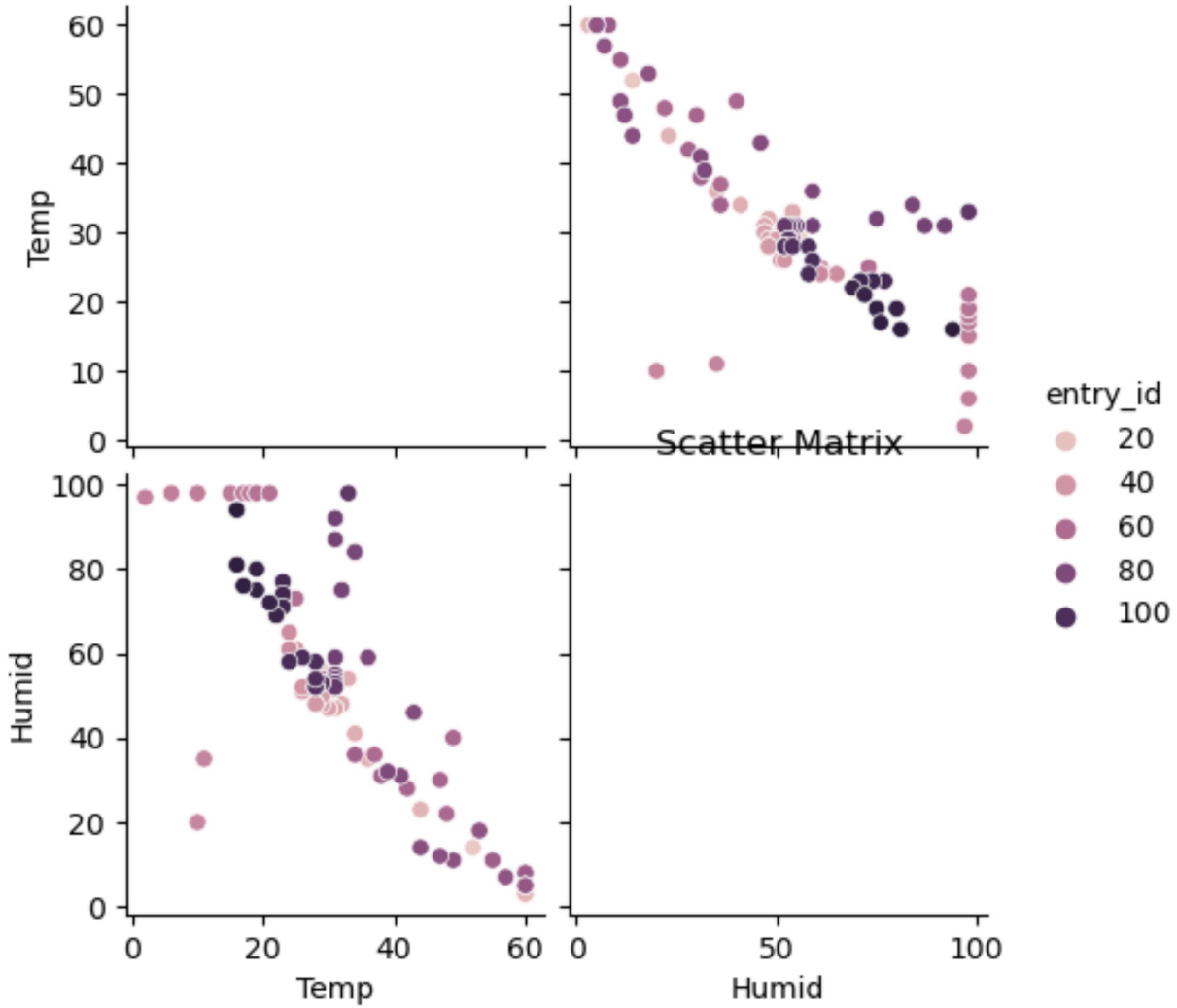


In [21]:
```
sns.violinplot(data=df, x='Humid')
plt.title('Humidity Violin Plot')
plt.show()
```



Scatter Matrix

In [22]:
```
sns.pairplot(df, vars=['Temp', 'Humid'], hue='entry_id', diag_kind='kde')
plt.title('Scatter Matrix')
plt.show()
```



What is the range of values for temperature and humidity in the dataset? Are there any extreme values or outliers?

In [23]:
```
# Get summary statistics for temperature and humidity
print(df[['Temp', 'Humid']].describe())

# Create boxplots to visualize the distribution of temperature and humidity
sns.boxplot(data=df[['Temp', 'Humid']])
```

```
              Temp       Humid
count  100.000000  100.000000
mean    33.280000   49.130000
std     14.570678   28.640458
min      2.000000    3.000000
25%     24.000000   29.500000
50%     30.000000   52.000000
75%     42.250000   69.500000
max     60.000000   98.000000
```

Out[23]: <AxesSubplot:>

How does the temperature and humidity vary over time? Are there any patterns or trends?

In [24]:
```
# Convert the 'created_at' column to a datetime format
df['created_at'] = pd.to_datetime(df['created_at'])

# Create line plots to visualize the temperature and humidity over time
sns.lineplot(data=df, x='created_at', y='Temp')
sns.lineplot(data=df, x='created_at', y='Humid')
```

Out[24]: <AxesSubplot:xlabel='created_at', ylabel='Temp'>

Is there a relationship between temperature and humidity? How strong is this relationship?

In [25]:
```
# Create a scatter plot to visualize the relationship between temperature and humidity
sns.scatterplot(data=df, x='Temp', y='Humid')
```

Out[25]: <AxesSubplot:xlabel='created_at', ylabel='Temp'>

Are there any differences in temperature and humidity between different entry IDs? Are there any entry IDs with significantly different values?

In [26]:
```
# Create boxplots to visualize the distribution of temperature and humidity for each entry ID
sns.boxplot(data=df, x='entry_id', y='Temp')
sns.boxplot(data=df, x='entry_id', y='Humid')

sns.barplot(x=df['entry_id'].value_counts().index, y=df['entry_id'].value_counts().values, palette='Set2')
```

Out[26]:
```
<AxesSubplot:xlabel='entry_id', ylabel='Humid'>
```

How many unique entry IDs are in the dataset? How frequently do different entry IDs appear?

In [27]:
```
# Print the number of unique entry IDs and their frequencies
print(df['entry_id'].value_counts())
```

```
13     1
76     1
86     1
85     1
84     1
      ..
43     1
42     1
41     1
40     1
112    1
Name: entry_id, Length: 100, dtype: int64
```

What is the overall distribution of temperature and humidity in the dataset? Are the distributions normal or skewed?

In [28]:
```
# Create histograms to visualize the distribution of temperature and humidity
sns.histplot(data=df, x='Temp', color='blue')
sns.histplot(data=df, x='Humid', color='green')
```

Out[28]:
```
<AxesSubplot:xlabel='entry_id', ylabel='Humid'>
```

Are there any missing values in the dataset? How many missing values are there, and how do they impact the analysis?

In [29]:
```
# Count the number of missing values for each variable
print(df.isnull().sum())
```

```
created_at    0
entry_id      0
Temp          0
Humid         0
dtype: int64
```

In [ ]:

In [ ]:

In [30]:
```
# Split the data into a 70-30 train-test split
train_df, test_df = train_test_split(df, test_size=0.3, random_state=42)

print('train df:\n',train_df)

print('\n\ntest df:\n',test_df)
```

train df:

	created_at	entry_id	Temp	Humid
11	2023-01-07 13:54:42+00:00	24	60	5
47	2023-01-07 14:07:29+00:00	60	37	36
85	2023-01-07 14:17:32+00:00	98	28	54
28	2023-01-07 13:59:31+00:00	41	25	61
93	2023-01-07 14:19:35+00:00	106	22	69
..	...	...	...	...
60	2023-01-07 14:10:47+00:00	73	57	7
71	2023-01-07 14:13:38+00:00	84	31	87
14	2023-01-07 13:55:27+00:00	27	36	35
92	2023-01-07 14:19:19+00:00	105	23	71
51	2023-01-07 14:08:29+00:00	64	42	28

[70 rows x 4 columns]

test df:

	created_at	entry_id	Temp	Humid
83	2023-01-07 14:16:59+00:00	96	28	52
53	2023-01-07 14:09:00+00:00	66	34	36
70	2023-01-07 14:13:21+00:00	83	32	75
45	2023-01-07 14:06:57+00:00	58	25	73
44	2023-01-07 14:06:42+00:00	57	21	98
39	2023-01-07 14:05:26+00:00	52	15	98
22	2023-01-07 13:57:30+00:00	35	29	48
80	2023-01-07 14:16:12+00:00	93	29	53
10	2023-01-07 13:54:26+00:00	23	60	3
0	2023-01-07 13:51:53+00:00	13	29	56
18	2023-01-07 13:56:29+00:00	31	31	47
30	2023-01-07 14:00:02+00:00	43	24	65
73	2023-01-07 14:14:16+00:00	86	31	59
33	2023-01-07 14:03:53+00:00	46	10	20
90	2023-01-07 14:18:49+00:00	103	23	77
4	2023-01-07 13:52:55+00:00	17	52	14
76	2023-01-07 14:15:12+00:00	89	31	53
77	2023-01-07 14:15:27+00:00	90	31	53
12	2023-01-07 13:54:57+00:00	25	60	3
31	2023-01-07 14:00:20+00:00	44	24	61
55	2023-01-07 14:09:30+00:00	68	60	5
88	2023-01-07 14:18:19+00:00	101	24	58
26	2023-01-07 13:59:00+00:00	39	26	51
42	2023-01-07 14:06:12+00:00	55	19	98
69	2023-01-07 14:13:05+00:00	82	34	84
15	2023-01-07 13:55:43+00:00	28	34	41
40	2023-01-07 14:05:41+00:00	53	17	98

```
96 2023-01-07 14:20:20+00:00       109    19     75
9  2023-01-07 13:54:11+00:00        22    60      3
72 2023-01-07 14:13:54+00:00        85    31     92
```

In [31]:
```
# Train a linear regression model on the training data
model = LinearRegression()
model.fit(train_df[['Humid']], train_df['Temp'])
```

Out[31]:
```
LinearRegression()
```

In [32]:
```
# Make predictions on the test data
y_pred = model.predict(test_df[['Humid']])

# Calculate the mean absolute error and the coefficient of determination
mae = mean_absolute_error(test_df['Temp'], y_pred)
r2 = r2_score(test_df['Temp'], y_pred)

# Print the results
print('Mean Absolute Error:', mae)
print('R²:', r2)
```

```
Mean Absolute Error: 6.251913051859811
R²: 0.5169168412538039
```

This code will print the MAE and R^2, which are both measures of how well your model is able to predict the temperature based on the humidity. The lower the MAE and the higher the R^2, the better the accuracy of your model.

Cross Validation

Rather than using a single train-test split, you can use cross-validation to evaluate your model's performance on multiple train-test splits. This can give you a more reliable estimate of your model's accuracy. Here's an example code for performing 10-fold cross-validation on your linear regression model:

In [33]:
```
# Perform 10-fold cross-validation on the linear regression model
scores = cross_val_score(model, df[['Humid']], df['Temp'], cv=10)

# Print the mean and standard deviation of the cross-validation scores
print('Cross-Validation Mean:', scores.mean())
print('Cross-Validation Std:', scores.std())
```

```
Cross-Validation Mean: -30.865217046866587
Cross-Validation Std: 87.94295746069663
```

Hyperparameter tuning

Some machine learning models have hyperparameters that you can tune to improve their accuracy. You can use techniques like grid search or randomized search to find the best hyperparameter values for your model. Here's an example code for performing a grid search to find the best value for the alpha hyperparameter of a Ridge regression model

In [34]:
```
# Define a Ridge regression model
model = Ridge()

# Define the hyperparameters to search over
param_grid = {'alpha': [0.1, 1.0, 10.0]}

# Perform a grid search to find the best hyperparameter values
grid_search = GridSearchCV(model, param_grid, cv=10)
grid_search.fit(df[['Humid']], df['Temp'])

# Print the best hyperparameter values and the corresponding cross-validation score
print('Best Hyperparameters:', grid_search.best_params_)
print('Best Cross-Validation Score:', grid_search.best_score_)
```

```
Best Hyperparameters: {'alpha': 10.0}
Best Cross-Validation Score: -30.856637051630862
```

Feature engineering

You can also try creating new features from your existing features to improve your model's accuracy. For example, you could create a new feature that represents the interaction between the temperature and humidity. Here's an example code for creating a new feature that represents the interaction between the temperature and humidity:

In [35]:
```
# Create a new feature that represents the interaction between the temperature and humidity
df['Temp*Humid'] = df['Temp'] * df['Humid']

# Train a linear regression model on the new feature and the original humidity feature
model = LinearRegression()
model.fit(df[['Humid', 'Temp']], df['Temp'])

# Make predictions on the test data and calculate the accuracy
y_pred = model.predict(test_df[['Humid', 'Temp']])
mae = mean_absolute_error(test_df['Temp'], y_pred)
r2 = r2_score(test_df['Temp'], y_pred)

# Print the results
print('Mean Absolute Error:', mae)
print('R²:', r2)
```

```
Mean Absolute Error: 6.631732200427602e-15
R²: 1.0
```

In [36]:
```
# Select the features
X = df[['Temp', 'Humid']]

# Select the target
y = df['entry_id']

# Add a constant term to the features
X = sm.add_constant(X)

# Create the model
model = sm.OLS(y, X)

# Fit the model
results = model.fit()

# Print the summary
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               entry_id   R-squared:                       0.132
Model:                            OLS   Adj. R-squared:                  0.114
Method:                 Least Squares   F-statistic:                     7.388
Date:                Fri, 31 Mar 2023   Prob (F-statistic):            0.00103
Time:                        15:00:36   Log-Likelihood:                -471.07
No. Observations:                 100   AIC:                             948.1
Df Residuals:                      97   BIC:                             956.0
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         31.9203     18.743      1.703      0.092      -5.279      69.120
Temp           0.2375      0.333      0.714      0.477      -0.423       0.898
Humid          0.4615      0.169      2.726      0.008       0.125       0.798
==============================================================================
Omnibus:                      213.507   Durbin-Watson:                   0.054
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                9.452
Skew:                          -0.074   Prob(JB):                      0.00886
Kurtosis:                       1.501   Cond. No.                         428.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [37]:
```
mlr = LinearRegression()
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=100)
mlr.fit(X_train,y_train)
y_pred_mlr= mlr.predict(X_test)
#Predicted values
print("Prediction for Temp set: {}".format(y_pred_mlr))
```

```
Prediction for Temp set: [81.84970773 48.08936346 62.90134319 84.48763802 51.65672522 61.85089728
 80.47369732 43.9863929  64.99388145 61.79385201 64.8310992  57.08701449
 72.17586701 81.05107393 73.28335819 64.49718113 84.36519391 47.52726431
 84.26781049 47.52726431 66.65511823 63.83769856 63.83769856 84.70746554
 67.64851886 62.07072481 63.06412545 75.15606893 66.81790048 61.79385201]
```

Accuracy test

In [82]:
```
X = df[['Temp', 'Humid']]
y = df['Temp']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)
```

In [90]:
```
svc = SVC(kernel='linear')
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)*100
print("SVC Accuracy:{:.2f} %".format(round(accuracy, 2)))
```

```
SVC Accuracy:63.33 %
```

In [ ]: