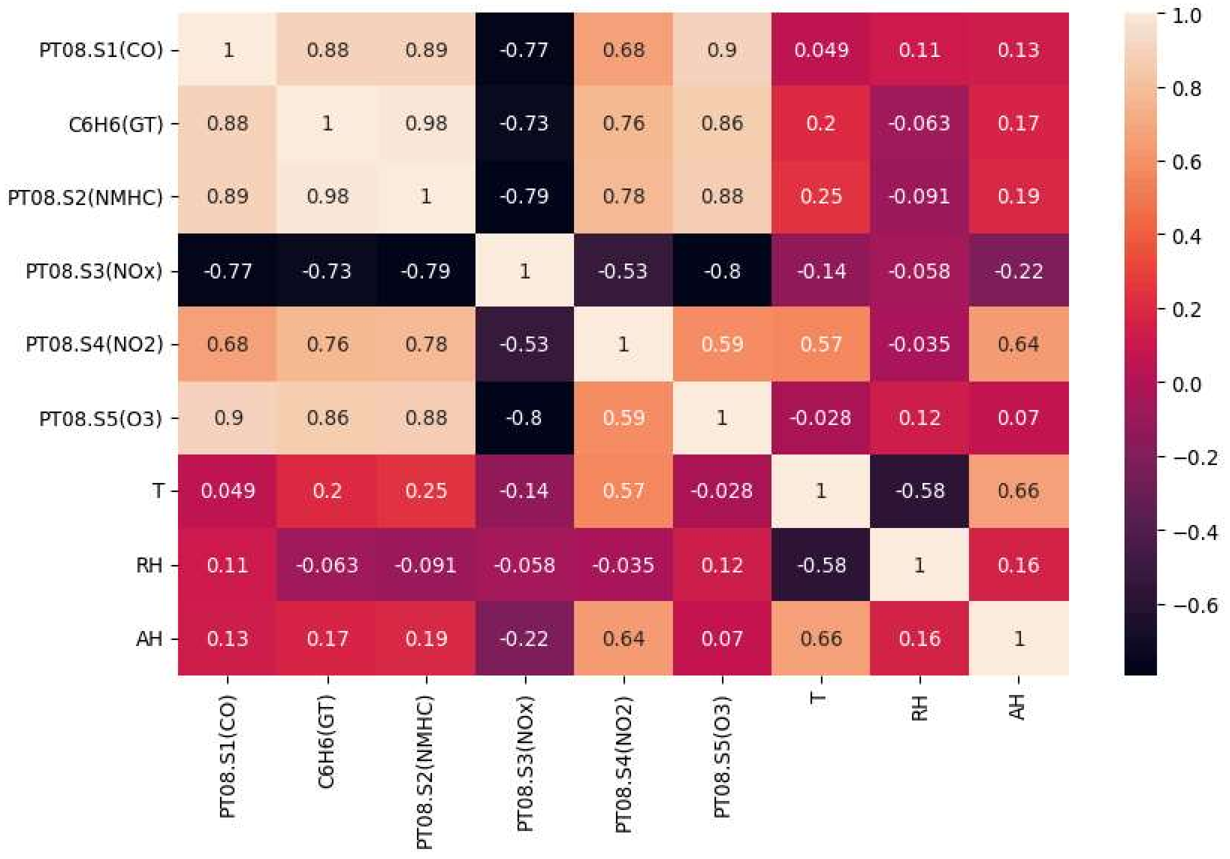
In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

# Loading the dataset.
csv_file = 'AirQualityUCI.csv'

df = pd.read_csv(csv_file, sep=';')

# Dropping the 'Unnamed: 15' & 'Unnamed: 16' columns.
df = df.drop(columns=['Unnamed: 15', 'Unnamed: 16'], axis=1)

# Dropping the null values.
df = df.dropna()

# Creating a Pandas series containing 'datetime' objects.
dt_series = pd.Series(data = [item.split("/")[2] + "-" + item.split("/")[1] + "-" + it
dt_series = pd.to_datetime(dt_series)

# Remove the Date & Time columns from the DataFrame and insert the 'dt_series' in it.
df = df.drop(columns=['Date', 'Time'], axis=1)
df.insert(loc=0, column='DateTime', value=dt_series)

# Get the Pandas series containing the year values as integers.
year_series = dt_series.dt.year

# Get the Pandas series containing the month values as integers.
month_series = dt_series.dt.month

# Get the Pandas series containing the day values as integers.
day_series = dt_series.dt.day

# Get the Pandas series containing the days of a week, i.e., Monday, Tuesday, Wednesda
day_name_series = dt_series.dt.day_name()

# Add the 'Year', 'Month', 'Day' and 'Day Name' columns to the DataFrame.
df['Year'] = year_series
df['Month'] = month_series
df['Day'] = day_series
df['Day Name'] = day_name_series

# Sort the DataFrame by the 'DateTime' values in the ascending order. Also, display th
df = df.sort_values(by='DateTime')

# Create a function to replace the commas with periods in a Pandas series.
def comma_to_period(series):
    new_series = pd.Series(data=[float(str(item).replace(',', '.')) for item in series
    return new_series

# Apply the 'comma_to_period()' function on the ''CO(GT)', 'C6H6(GT)', 'T', 'RH' and '
cols_to_correct = ['CO(GT)', 'C6H6(GT)', 'T', 'RH', 'AH'] # Create a list of column na
for col in cols_to_correct: # Iterate through each column
    df[col] = comma_to_period(df[col]) # Replace the original column with the new seri
```

```
# Remove all the columns from the 'df' DataFrame containing more than 10% garbage valu
df = df.drop(columns=['NMHC(GT)', 'CO(GT)', 'NOx(GT)', 'NO2(GT)'], axis=1)

# Create a new DataFrame containing records for the years 2004 and 2005.
aq_2004_df = df[df['Year'] == 2004]
aq_2005_df = df[df['Year'] == 2005]

# Replace the -200 value with the median values for each column having indices between
for col in aq_2004_df.columns[1:-4]:
  median = aq_2004_df.loc[aq_2004_df[col] != -200, col].median()
  aq_2004_df[col] = aq_2004_df[col].replace(to_replace=-200, value=median)

# Repeat the same exercise for the 2005 year DataFrame.
for col in aq_2005_df.columns[1:-4]:
  median = aq_2005_df.loc[aq_2005_df[col] != -200, col].median()
  aq_2005_df[col] = aq_2005_df[col].replace(to_replace=-200, value=median)

# Group the DataFrames about the 'Month' column.
group_2004_month = aq_2004_df.groupby(by='Month')
group_2005_month = aq_2005_df.groupby(by='Month')

# Concatenate the two DataFrames for 2004 and 2005 to obtain one DataFrame.
df = pd.concat([aq_2004_df, aq_2005_df])

# Information of the DataFrame.
df.info()



# Heatmap to pinpoint the columns in the 'df' DataFrame exhibiting high correlation.
corr_df = df.iloc[:, 1:-4].corr()
plt.figure(figsize = (10, 6), dpi = 96)
sns.heatmap(data = corr_df, annot = True) # 'annot=True' fills the R values in the hea
plt.show()

"""---

#### Train-Test Split
"""

# Splitting the DataFrame into the train and test sets.
from sklearn.model_selection import train_test_split

X = df['T'] # Pandas DataFrame containing only feature variables
y = df['RH'] # Pandas Series containing the target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_sta

# Create the 'errors_product()' and 'squared_errors()' function.
def errors_product():
  prod = (X_train - X_train.mean()) * (y_train - y_train.mean())
  return prod

def squared_errors():
  sq_errors = (X_train - X_train.mean()) ** 2
  return sq_errors

# Calculate the slope and intercept values for the best fit line.
slope = errors_product().sum()/ squared_errors().sum()
intercept = y_train.mean() - slope * X_train.mean()
```

```
print(f"Slope: {slope} \nIntercept: {intercept}")

"""Equation of the best fit line is
$$y = -1.1120x + 69.6911$$

or

$$\text{relative humidity} = -1.1120 \times \text{temperature} + 69.6911$$

which is also the regression line.
"""

# Plot the regression line in the scatter plot between relative humidity and temperatu
plt.figure(figsize = (12, 5), dpi = 96)
plt.title("Regression Line", fontsize = 16)
plt.scatter(df['T'], df['RH'])
plt.plot(df['T'], slope * df['T'] + intercept, color = 'r', linewidth = 2, label = '$y
plt.xlabel("Temperature")
plt.ylabel("Relative Humidity")
plt.legend()
plt.show()

"""---

#### Simple Linear Regression Using `sklearn` Module^^
"""

# Deploy linear regression model using the 'sklearn.linear_model' module.
from sklearn.linear_model import LinearRegression

X_train_reshaped = X_train.values.reshape(-1, 1)
y_train_reshaped = y_train.values.reshape(-1, 1)
X_test_reshaped = X_test.values.reshape(-1, 1)
y_test_reshaped = y_test.values.reshape(-1, 1)


lin_reg = LinearRegression()
lin_reg.fit(X_train_reshaped, y_train_reshaped)

print("Coefficient of $x$ (or slope) ==>", lin_reg.coef_)
print("Intercept ==>", lin_reg.intercept_)

# Evaluate the linear regression model using the 'r2_score', 'mean_squared_error' & 'm
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

y_train_pred = lin_reg.predict(X_train_reshaped)
y_test_pred = lin_reg.predict(X_test_reshaped)

print(f"Train Set\n{'-' * 50}")
print(f"R-squared: {r2_score(y_train_reshaped, y_train_pred):.3f}")
print(f"Mean Squared Error: {mean_squared_error(y_train_reshaped, y_train_pred):.3f}")
print(f"Root Mean Squared Error: {np.sqrt(mean_squared_error(y_train_reshaped, y_trair
print(f"Mean Absolute Error: {mean_absolute_error(y_train_reshaped, y_train_pred):.3f}

print(f"\n\nTest Set\n{'-' * 50}")
print(f"R-squared: {r2_score(y_test_reshaped, y_test_pred):.3f}")
print(f"Mean Squared Error: {mean_squared_error(y_test_reshaped, y_test_pred):.3f}")
print(f"Root Mean Squared Error: {np.sqrt(mean_squared_error(y_test_reshaped, y_test_p
print(f"Mean Absolute Error: {mean_absolute_error(y_test_reshaped, y_test_pred):.3f}")
```

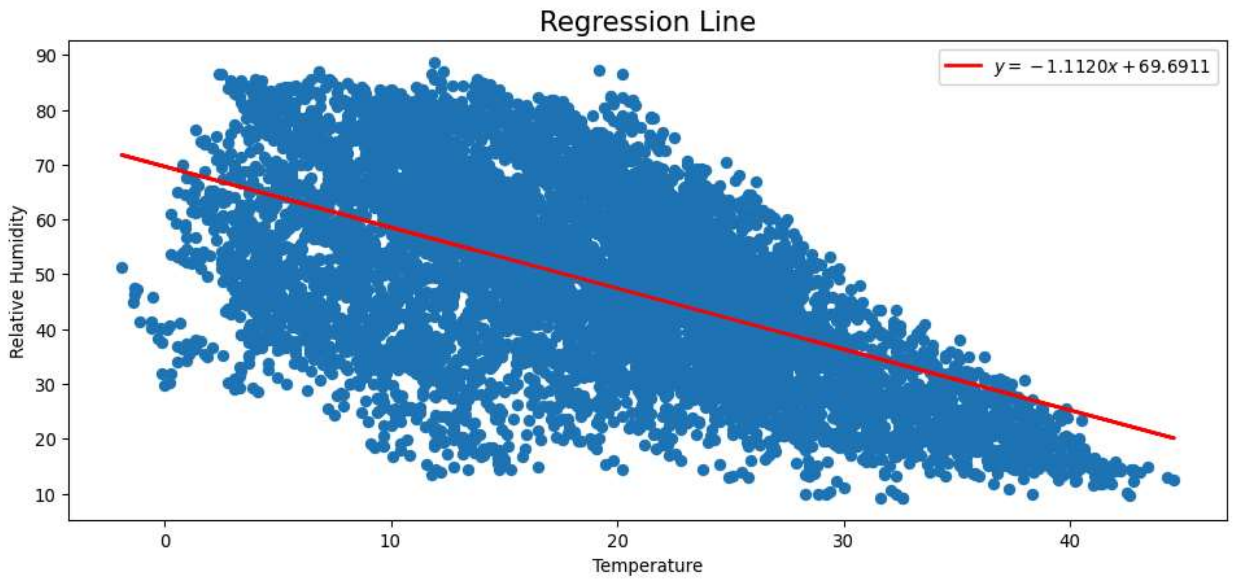
```
"""For a highly accurate regression model:

- The $R^2$ squared value should be close to 1.

- The MSE, RMSE and MAE values should be close to zero. However, in the case of fracti

---
"""
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9357 entries, 0 to 9356
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   DateTime       9357 non-null   datetime64[ns]
 1   PT08.S1(CO)    9357 non-null   float64       
 2   C6H6(GT)       9357 non-null   float64       
 3   PT08.S2(NMHC)  9357 non-null   float64       
 4   PT08.S3(NOx)   9357 non-null   float64       
 5   PT08.S4(NO2)   9357 non-null   float64       
 6   PT08.S5(O3)    9357 non-null   float64       
 7   T              9357 non-null   float64       
 8   RH             9357 non-null   float64       
 9   AH             9357 non-null   float64       
 10  Year           9357 non-null   int64         
 11  Month          9357 non-null   int64         
 12  Day            9357 non-null   int64         
 13  Day Name       9357 non-null   object        
dtypes: datetime64[ns](1), float64(9), int64(3), object(1)
memory usage: 1.1+ MB
```



	PT08.S1(CO)	C6H6(GT)	PT08.S2(NMHC)	PT08.S3(NOx)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	AH
PT08.S1(CO)	1	0.88	0.89	-0.77	0.68	0.9	0.049	0.11	0.13
C6H6(GT)	0.88	1	0.98	-0.73	0.76	0.86	0.2	-0.063	0.17
PT08.S2(NMHC)	0.89	0.98	1	-0.79	0.78	0.88	0.25	-0.091	0.19
PT08.S3(NOx)	-0.77	-0.73	-0.79	1	-0.53	-0.8	-0.14	-0.058	-0.22
PT08.S4(NO2)	0.68	0.76	0.78	-0.53	1	0.59	0.57	-0.035	0.64
PT08.S5(O3)	0.9	0.86	0.88	-0.8	0.59	1	-0.028	0.12	0.07
T	0.049	0.2	0.25	-0.14	0.57	-0.028	1	-0.58	0.66
RH	0.11	-0.063	-0.091	-0.058	-0.035	0.12	-0.58	1	0.16
AH	0.13	0.17	0.19	-0.22	0.64	0.07	0.66	0.16	1

```
Slope: -1.112053910794772
Intercept: 69.69110324644876
```



```
Coefficient of $x$ (or slope) ==> [[-1.11205391]]
Intercept ==> [69.69110325]
Train Set
--------------------------------------------------
R-squared: 0.325
Mean Squared Error: 195.281
Root Mean Squared Error: 13.974
Mean Absolute Error: 11.289


Test Set
--------------------------------------------------
R-squared: 0.346
Mean Squared Error: 187.026
Root Mean Squared Error: 13.676
Mean Absolute Error: 11.150
```

```
Out[3]: 'For a highly accurate regression model:\n\n- The $R^2$ squared value should be close
to 1.\n\n- The MSE, RMSE and MAE values should be close to zero. However, in the case
of fractional values (or values between 0 and 1), MAE is a better metric to evaluate
the accuracy of a regression-based prediction model.\n\n---\n'
```

```
In [ ]:
```

```
In [ ]:
```