Prerequisite 1: Download and Store the CSV files

```
import requests
import os

# Define the base URLs
base_url_1 = "https://www.ncei.noaa.gov/data/global-summary-of-the-day/access/{}/99495199999.csv"
base_url_2 = "https://www.ncei.noaa.gov/data/global-summary-of-the-day/access/{}/72429793812.csv"

# Define the range of years
years = range(2021, 2023)

# Base directory to save the downloaded files
base_output_dir = "./weather_data/"

# Loop through each year and download the CSV files for both datasets
for year in years:
    # Create a directory for each year
    year_dir = os.path.join(base_output_dir, str(year))
    os.makedirs(year_dir, exist_ok=True)

    # Download each file (Florida and Cincinnati)
    for base_url, station_id in [(base_url_1, "99495199999"), (base_url_2, "72429793812")]:
        url = base_url.format(year)
        response = requests.get(url)

        # Check if the request was successful
        if response.status_code == 200:
            # Save the file in the appropriate year directory
            file_path = os.path.join(year_dir, f"{station_id}.csv")
            with open(file_path, "wb") as file:
                file.write(response.content)
            print(f"Downloaded: {file_path}")
        else:
            print(f"Failed to download {url}. Status code: {response.status_code}")
```

```
Downloaded: ./weather_data/2021/99495199999.csv
Downloaded: ./weather_data/2021/72429793812.csv
Downloaded: ./weather_data/2022/99495199999.csv
Downloaded: ./weather_data/2022/72429793812.csv
```

Prerequisite 2: Clean the data preserving original data

```
!pip install pandas
import os
import pandas as pd

# Define the base input and output directories
base_input_dir = "./weather_data/"
base_output_dir = "./cleaned_weather_data/"

# Define the invalid value representations
invalid_values = {
#     "TEMP": 9999.9,
#     "DEWP": 9999.9,
#     "SLP": 9999.9,
#     "STP": 9999.9,
#     "VISIB": 999.9,
#     "WDSP": 999.9,
    "MXSPD": 999.9,
#     "GUST": 999.9,
    "MAX": 9999.9,
#     "MIN": 9999.9,
#     "PRCP": 99.99,
#     "SNDP": 999.9
}

# Loop through each year directory
for year in range(2021, 2023):
    year_dir = os.path.join(base_input_dir, str(year))

    # Check if the year directory exists
    if os.path.exists(year_dir):
        # Loop through each file in the year directory
        for station_id in ["99495199999", "72429793812"]:
            file_path = os.path.join(year_dir, f"{station_id}.csv")

            # Check if the file exists
            if os.path.exists(file_path):
                # Read the CSV file into a DataFrame
                df = pd.read_csv(file_path)

                # Filter out rows with invalid values
                for column, invalid_value in invalid_values.items():
                    df = df[df[column] != invalid_value]

                # Create the output directory for the year if it doesn't exist
                output_year_dir = os.path.join(base_output_dir, str(year))
                os.makedirs(output_year_dir, exist_ok=True)

                # Save the cleaned DataFrame to the new directory
                cleaned_file_path = os.path.join(output_year_dir, f"{station_id}.csv")
                df.to_csv(cleaned_file_path, index=False)
                print(f"Cleaned data saved to: {cleaned_file_path}")
            else:
                print(f"File not found: {file_path}")
    else:
        print(f"Year directory not found: {year_dir}")
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in /home/pccoe/.local/lib/python3.10/site-packages (2.2.3)
Requirement already satisfied: pytz>=2020.1 in /usr/lib/python3/dist-packages (from pandas) (2022.1)
Requirement already satisfied: numpy>=1.22.4 in /home/pccoe/.local/lib/python3.10/site-packages (from p
Requirement already satisfied: tzdata>=2022.7 in /home/pccoe/.local/lib/python3.10/site-packages (from
Requirement already satisfied: python-dateutil>=2.8.2 in /home/pccoe/.local/lib/python3.10/site-package
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from python-dateutil>=2.8.2-
Cleaned data saved to: ./cleaned_weather_data/2021/99495199999.csv
Cleaned data saved to: ./cleaned_weather_data/2021/72429793812.csv
Cleaned data saved to: ./cleaned_weather_data/2022/99495199999.csv
Cleaned data saved to: ./cleaned_weather_data/2022/72429793812.csv
```

Question 2: Load the CSV files and display the count of each dataset.

Question 3: Find the hottest day (column MAX) for each year.

```
from pyspark.sql import functions as F
import os

# Base path to the weather data
base_path = "/home/pccoe/Downloads/cleaned_weather_data/"

# Initialize a dictionary to store the hottest days per year
hottest_days = {}

# Loop through the years to find the hottest day
for year in range(2021, 2023):
    year_dir = os.path.join(base_path, str(year))
    for filename in os.listdir(year_dir):
        if filename.endswith('.csv'):
            # Read the CSV file into a DataFrame
            df = spark.read.csv(os.path.join(year_dir, filename), header=True, inferSchema=True)

            # Check if the DataFrame is empty
            if df.isEmpty():
                continue  # Skip to the next file

            # Check if the "MAX" column exists
            if "MAX" not in df.columns:
                print(f"The 'MAX' column does not exist in {filename}.")
                continue  # Skip to the next file

            # Find the hottest day for the current DataFrame
            max_day = df.orderBy(F.desc("MAX")).first()

            # Check if max_day is None
            if max_day is not None:
                # Store the hottest day only if the year is not already recorded
                if year not in hottest_days:
                    hottest_days[year] = (max_day.STATION, max_day.NAME, max_day.DATE, max_day.MAX)

# Convert results to a DataFrame for display
if hottest_days:
    hottest_days_list = [(year, *data) for year, data in hottest_days.items()]
    hottest_days_df = spark.createDataFrame(hottest_days_list, ["YEAR", "STATION", "NAME", "DATE", "MAX"])
    hottest_days_df.show()
else:
    print("No hottest days found across the datasets.")
```

Question. 4: Find the coldest day (column MIN) for the month of March across all years (2015-2024).

```
from pyspark.sql import functions as F
import os

# Initialize an empty list to store results
march_data = []

# Initialize Spark session
spark = SparkSession.builder.appName("Coldest Day").getOrCreate()

# Base path to the weather data
base_path = "./cleaned_weather_data/"

# Loop through the years to collect March data
for year in range(2015, 2025):
    year_dir = os.path.join(base_path, str(year))
    for filename in os.listdir(year_dir):
        if filename.endswith('.csv'):
            df = spark.read.csv(os.path.join(year_dir, filename), header=True, inferSchema=True)

            # Filter for March data
            march_df = df.filter(df.DATE.contains('-03-'))

            if not march_df.isEmpty():
                # Get the coldest day for March in the current DataFrame
                coldest_day = march_df.orderBy(F.asc("MIN")).first()

                # Append results
                if coldest_day is not None:
                    march_data.append((coldest_day.STATION, coldest_day.NAME, coldest_day.DATE, coldest_day.

# Convert results to a DataFrame for display
if march_data:
    coldest_day_df = spark.createDataFrame(march_data, ["STATION", "NAME", "DATE", "MIN"])
    # Sort by MIN to get the overall coldest day in March
    overall_coldest_day = coldest_day_df.orderBy(F.asc("MIN")).first()
    overall_coldest_day_df = spark.createDataFrame([overall_coldest_day], ["STATION", "NAME", "DATE", "MIN"]
    overall_coldest_day_df.show()  # Display only the overall coldest day
else:
    print("No March data found across the datasets.")
```

```python
from pyspark.sql import functions as F
import os

# Initialize an empty list to store results
annual_precipitation = []

# Initialize Spark session
spark = SparkSession.builder.appName("Most Precipitation").getOrCreate()

# Base path to the cleaned weather data
base_path = "./cleaned_weather_data/"

# Loop through the years to calculate mean precipitation
for year in range(2015, 2025):
    year_dir = os.path.join(base_path, str(year))
    for filename in os.listdir(year_dir):
        if filename.endswith('.csv'):
            # Read the CSV file into a DataFrame
            df = spark.read.csv(os.path.join(year_dir, filename), header=True, inferSchema=True)

            # Check if the DataFrame is empty
            if df.isEmpty():
                continue  # Skip to the next file

            # Check if the DataFrame contains the 'PRCP' column
            if "PRCP" not in df.columns:
                print(f"'PRCP' column not found in {filename}")
                continue

            # Calculate mean of PRCP
            mean_prcp = df.agg(F.mean("PRCP").alias("Mean_PRCP")).first().Mean_PRCP

            # Get station info
            station_id = df.select("STATION").first().STATION
            station_name = df.select("NAME").first().NAME

            # Append results
            annual_precipitation.append((station_id, station_name, year, mean_prcp))

# Create a DataFrame from the results
annual_precipitation_df = spark.createDataFrame(annual_precipitation, ["STATION", "NAME", "YEAR", "Mean_PR

# Find the year with the most precipitation for each station
cincinnati_max_prcp = annual_precipitation_df.filter(annual_precipitation_df.STATION == "72429793812") \
    .orderBy(F.desc("Mean_PRCP")).first()

florida_max_prcp = annual_precipitation_df.filter(annual_precipitation_df.STATION == "99495199999") \
    .orderBy(F.desc("Mean_PRCP")).first()

# Display the results
if cincinnati_max_prcp:
    print(f"Cincinnati: STATION={cincinnati_max_prcp.STATION}, NAME={cincinnati_max_prcp.NAME}, YEAR={cinci

if florida_max_prcp:
    print(f"Florida: STATION={florida_max_prcp.STATION}, NAME={florida_max_prcp.NAME}, YEAR={florida_max_pr
```

Question 6: Count the percentage of missing values for wind gust (column GUST) for Cincinnati and Florida in the year 2024.

```
from pyspark.sql import SparkSession
import os

# Initialize Spark session
spark = SparkSession.builder.appName("Wind Gust Missing Values").getOrCreate()

# Base path to the cleaned weather data
base_path = "./cleaned_weather_data/2024/"

# Station codes for Florida and Cincinnati
station_codes = ['99495199999', '72429793812']  # Florida, Cincinnati
results = []

# Loop through each station code
for station_code in station_codes:
    file_path = os.path.join(base_path, f"{station_code}.csv")

    # Load the CSV file if it exists
    if os.path.exists(file_path):
        df = spark.read.csv(file_path, header=True, inferSchema=True)

        # Count total rows and missing values in the GUST column
        total_count = df.count()
        missing_count = df.filter(df.GUST == 999.9).count()

        # Calculate the percentage of missing values
        if total_count > 0:
            missing_percentage = (missing_count / total_count) * 100
        else:
            missing_percentage = 0.0

        # Store the result for this station
        results.append((station_code, missing_percentage))

# Display the results
for station_code, missing_percentage in results:
    print(f"Station Code: {station_code}, Missing GUST Percentage in the year 2024: {missing_percentage:.2f

# Stop the Spark session
spark.stop()
```

```
Station Code: 99495199999, Missing GUST Percentage in the year 2024: 100.00%
Station Code: 72429793812, Missing GUST Percentage in the year 2024: 40.00%
```

Question 7: Find the mean, median, mode, and standard deviation of the temperature (column TEMP) for Cincinnati in each month for the year 2020.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import mean, col, stddev, expr
from pyspark.sql import functions as F

# Initialize Spark session
```

```
spark = SparkSession.builder.appName("Temperature Analysis").getOrCreate()

# Load the data
df = spark.read.csv("./cleaned_weather_data/2020/72429793812.csv", header=True, inferSchema=True)

# Extract month from date (assuming there's a DATE column)
df_cincinnati = df.withColumn("MONTH", F.month(col("DATE")))

# Group by month and calculate statistics
result = df_cincinnati.groupBy("MONTH").agg(
    mean("TEMP").alias("Mean"),
    expr("percentile_approx(TEMP, 0.5)").alias("Median"),  # Median
    F.mode("TEMP").alias("Mode"),  # Mode
    stddev("TEMP").alias("Standard Deviation")
)

# Show results
result.orderBy("MONTH").show()
```

```
+-----+------------------+------+----+------------------+
|MONTH|              Mean|Median|Mode|Standard Deviation|
+-----+------------------+------+----+------------------+
|    1| 37.94516129032259|  37.7|43.1| 8.345810873712928|
|    2|  36.5896551724138|  36.0|25.9|  7.90159770587055|
|    3|  49.0741935483871|  47.8|39.6| 8.779406500135623|
|    4|51.779999999999994|  51.0|48.4| 7.313162436838541|
|    5| 60.89032258064518|  63.7|73.9| 9.314768017820217|
|    6| 72.54666666666667|  73.7|74.2| 4.899946047087439|
|    7|              77.6|  77.9|72.5|  2.33794781806609|
|    8| 73.34516129032258|  73.7|73.2| 3.487868375734898|
|    9|              66.1|  65.8|60.6| 7.118262089331474|
|   10|55.193548387096776|  54.0|51.1|  6.72869157582517|
|   11|48.003333333333345|  47.7|47.7| 6.825938527529321|
|   12| 35.99354838709677|  35.2|32.1| 6.642787340861814|
+-----+------------------+------+----+------------------+
```

Question 8: Find the top 10 days with the lowest Wind Chill for Cincinnati in 2017.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, expr, unix_timestamp, date_format

# Initialize Spark session
spark = SparkSession.builder.appName("Wind Chill Analysis").getOrCreate()

# Load the data
df = spark.read.csv("./cleaned_weather_data/2017/72429793812.csv", header=True, inferSchema=True)

# Filter for TEMP < 50°F, and WDSP > 3 mph
df_cincinnati = df.filter((col("TEMP") < 50) & (col("WDSP") > 3))

# Calculate Wind Chill using the given formula
df_cincinnati = df_cincinnati.withColumn(
    "Wind Chill",
    35.74 + (0.6215 * col("TEMP")) - (35.75 * (col("WDSP") ** 0.16)) + (0.4275 * col("TEMP") * (col("WDSP"
)

# Add a date column for sorting
# Assuming there's a DATE column, we format it to just keep the date part
df_cincinnati = df_cincinnati.withColumn("DATE", date_format("DATE", "yyyy-MM-dd"))

# Select relevant columns and sort by Wind Chill
```