ASSIGNMENT NO 4

Title: Weather Data Analysis and Prediction

1. Introduction

- **Overview:**
    - Briefly describe the purpose of the project: to download, clean, analyze, and predict weather data.
    - Mention the use of Python libraries and PySpark for data processing.
- **Objectives:**
    - List the main objectives of the code, such as:
        - Downloading weather data from online sources.
        - Cleaning the data by handling invalid values.
        - Performing exploratory data analysis (EDA) to find insights.
        - Predicting future weather patterns using historical data

.

2. Data Description

- **Dataset Source:**
    - Specify the source of the weather data (e.g., National Centers for Environmental Information (NCEI)).
    - Provide the base URLs used for downloading the data.
- **Data Characteristics:**
    - Describe the type of data (e.g., daily weather summaries).
    - List the key columns used in the analysis (e.g., TEMP, DEWP, SLP, DATE, GUST, PRCP, FRSHTT).
    - Mention the time period covered by the data (2021-2023).
- **Data Cleaning:**
    - Explain how invalid values were handled (e.g., filtering out rows with specific values).

3. Methodology

- **Data Acquisition:**

    - Describe the process of downloading data using the requests library.
    - Explain how the code iterates through years and station IDs to download data for different locations (e.g., Florida and Cincinnati).
- **Data Cleaning:**
    - Detail the use of pandas for reading and cleaning the data.
    - Explain the process of identifying and removing invalid values from the data.
- **Data Analysis:**
    - Explain the use of PySpark for various data analysis tasks.
    - Describe the specific analyses performed, including:
        - Finding hottest and coldest days.
        - Analyzing precipitation.
        - Calculating descriptive statistics (mean, median, mode, standard deviation) for temperature.
        - Analyzing wind chill.
        - Counting days with extreme weather conditions.
- **Weather Prediction:**
    - Explain the use of Linear Regression for predicting maximum temperatures.
    - Describe how the training data was prepared and how the model was trained.
    - Explain the process of generating predictions for future months.

4. Tools and Libraries Used

- List the Python libraries used in the project and their purposes:
    - requests: For downloading data.
    - os: For interacting with the operating system.
    - pandas: For data cleaning and manipulation.
    - pyspark: For distributed data processing and analysis.
    - pyspark.sql.functions: For various DataFrame operations.
    - pyspark.ml.regression.LinearRegression: For building the prediction model.
    - pyspark.ml.feature.VectorAssembler: To assemble features for the model.

5. Results

- Summarize the key findings from the data analysis:
    - Examples:
        - Hottest days for each year.
        - Coldest day in March.
        - Years with most precipitation for Cincinnati and Florida.
        - Percentage of missing wind gust values.
        - Temperature statistics for Cincinnati.
        - Top 10 days with lowest wind chill.
        - Number of days with extreme weather conditions.
        - Maximum predicted temperatures for November and December 2024.
- Present any relevant tables or visualizations (if applicable).

6. Conclusion

We have successfully implement the weather prediction on pyspark using a large historical data and have tried to understand every aspect of the data.

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}")
```

```
Requirement already satisfied: pandas in
/usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.23.2 in
/usr/local/lib/python3.11/dist-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2-
>pandas) (1.17.0)
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.

```
!pip install pyspark
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
import os

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

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

# Initialize Spark session
spark = SparkSession.builder.appName("Hottest Day").getOrCreate() #
Creating a spark session and assigning it to the variable spark.

# 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.")
```

```
Requirement already satisfied: pyspark in
/usr/local/lib/python3.11/dist-packages (3.5.5)
Requirement already satisfied: py4j==0.10.9.7 in
/usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7)
+----+-----------+--------------------+----------+----+
|YEAR|    STATION|                NAME|      DATE| MAX|
+----+-----------+--------------------+----------+----+
|2021|72429793812|CINCINNATI MUNICI...|2021-08-12|95.0|
|2022|72429793812|CINCINNATI MUNICI...|2022-06-14|96.1|
+----+-----------+--------------------+----------+----+
```

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 = "/content/cleaned_weather_data"

# Loop through the years to collect March data
# Updated the range to (2021, 2023) to match available data
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'):
            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.MIN))

# 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.")
```

```
+-----------+--------------------+----------+----+
|    STATION|                NAME|      DATE| MIN|
```

```
+-----------+--------------------+----------+----+
|72429793812|CINCINNATI MUNICI...|2022-03-13|18.0|
+-----------+--------------------+----------+----+
```

Question 5: Find the year with the most precipitation for Cincinnati and Florida.

```
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
# Changed the range to (2021, 2023) to match available data
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 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_PRCP"])

# 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={cincinnati_max_prcp.YEAR}, Mean
of PRCP={cincinnati_max_prcp.Mean_PRCP}")

if florida_max_prcp:
    print(f"Florida: STATION={florida_max_prcp.STATION},
NAME={florida_max_prcp.NAME}, YEAR={florida_max_prcp.YEAR}, Mean of
PRCP={florida_max_prcp.Mean_PRCP}")
```

```
Cincinnati: STATION=72429793812, NAME=CINCINNATI MUNICIPAL AIRPORT
LUNKEN FIELD, OH US, YEAR=2022, Mean of PRCP=0.39241758241758234
```

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/2021/"  # Changed to 2021

# 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()
        # Assume that if GUST is 999.9, it is a missing value
        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))
    else:
        print(f"File not found for station code: {station_code}")

# Display the results
for station_code, missing_percentage in results:
    station_name = "Florida" if station_code == '99495199999' else
"Cincinnati"  # Get station name
    print(f"{station_name}: Missing GUST Percentage in the year 2021:
{missing_percentage:.2f}%")

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

```
Florida: Missing GUST Percentage in the year 2021: 0.00%
Cincinnati: Missing GUST Percentage in the year 2021: 52.05%
```

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
import os

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

# Construct the correct file path
file_path = os.path.join("./cleaned_weather_data", "2021",
"72429793812.csv")  # Changed 2020 to 2021
```

```
# Check if the file exists before attempting to load it
if os.path.exists(file_path):
    # Load the data
    df = spark.read.csv(file_path, 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()
else:
    print(f"Error: File not found at {file_path}")
```

```
+-----+------------------+------+----+------------------+
|MONTH|              Mean|Median|Mode|Standard Deviation|
+-----+------------------+------+----+------------------+
|    1|  33.9516129032258|  34.8|33.1| 4.899583041289802|
|    2| 29.95357142857143|  26.8|21.5| 9.450592070139862|
|    3|  47.8258064516129|  47.7|48.2| 7.747987598593389|
|    4| 52.87666666666666|  53.1|53.1| 8.798341667152622|
|    5|60.858064516129026|  58.5|70.9| 8.565931519437857|
|    6| 72.87666666666668|  72.9|63.5| 4.985128458437801|
|    7| 74.77419354838709|  75.2|73.9|  3.45571678449257|
|    8| 75.52258064516128|  75.6|75.5| 3.920349446789945|
|    9| 68.51333333333335|  69.7|74.9| 5.616400450732527|
|   10| 61.18387096774194|  61.4|60.0| 7.731541319995368|
|   11| 41.54666666666666|  39.1|37.7| 6.540838991493651|
|   12| 43.85161290322581|  46.1|32.7| 9.586896298863428|
+-----+------------------+------+----+------------------+
```

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
import os

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

```
# Update the file path to point to an existing file within the
'cleaned_weather_data' folder
file_path = "./cleaned_weather_data/2021/72429793812.csv"  # Using
2021 data as a proxy

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

    # Filter for Cincinnati data (station ID: 72429793812)
    cincinnati_df = df.filter(col("STATION") == "72429793812")

    # Calculate Wind Chill if the column doesn't exist (using a common
formula)
    # You might need to adjust the formula based on your data's units
    if "WND" not in cincinnati_df.columns: # Assuming WND is Wind
Speed and TEMP is Temperature in Celsius
        cincinnati_df = cincinnati_df.withColumn(
            "WND",
            expr(
                "13.12 + 0.6215 * TEMP - 11.37 * power(WDSP, 0.16) +
0.3965 * TEMP * power(WDSP, 0.16)"
            ),
        )

    # Order by Wind Chill (WND) in ascending order (lowest first) and
select the top 10
    top_10_lowest_wind_chill =
cincinnati_df.orderBy(col("WND").asc()).limit(10)

    # Format the DATE column for better readability (optional)
    # top_10_lowest_wind_chill = top_10_lowest_wind_chill.withColumn(
    #     "DATE", date_format(unix_timestamp("DATE", "yyyy-MM-
dd").cast("timestamp"), "yyyy-MM-dd")
    # )

    # Show the results
    top_10_lowest_wind_chill.select("DATE", "WND").show()  # Display
DATE and WND columns
else:
    print(f"Error: File not found at {file_path}")

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

```
+----------+------------------+
|      DATE|               WND|
+----------+------------------+
```

```
|2021-02-17|14.956525032438046|
|2021-02-16| 18.80671772412256|
|2021-02-15|22.382884855614343|
|2021-02-20|23.042083812001643|
|2021-02-11|23.712152333721775|
|2021-02-10| 24.34904551808595|
|2021-02-07| 24.82705649693917|
|2021-02-21|25.241016038086144|
|2021-02-08| 26.02325153920556|
|2021-02-18| 26.46568143429768|
+----------+------------------+
```

Question 9: Investigate how many days had extreme weather conditions for Florida (fog, rain, snow, etc.) using the FRSHTT column.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
import os

# Initialize a Spark session
spark = SparkSession.builder \
    .appName("Extreme Weather Analysis for Florida") \
    .getOrCreate()

# Define the directory containing cleaned weather data
base_directory = './cleaned_weather_data/'
file_paths = []

# Collect all relevant file paths for Florida
for year in range(2015, 2025):  # Adjust the range as necessary
    file_path = os.path.join(base_directory, str(year),
'99495199999.csv')
    if os.path.exists(file_path):
        file_paths.append(file_path)

# Load all the CSV files into a single DataFrame
df = spark.read.csv(file_paths, header=True, inferSchema=True)
# Count the number of days with extreme weather conditions
extreme_weather_count = df.filter(col("FRSHTT") != 0).count()

# Show the result
print(f"Number of days with extreme weather conditions in Florida:
{extreme_weather_count}")

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

```
Number of days with extreme weather conditions in Florida: 0
```

Question 10: Predict the maximum Temperature for Cincinnati for November and December 2024, based on the previous 2 years of weather data.

```
import os
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, dayofyear, month, max as
spark_max, when
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler

# Initialize Spark session
spark = SparkSession.builder.appName("Weather Data
Prediction").getOrCreate()

# Define base directory for your CSV files
base_directory = './cleaned_weather_data'
file_paths = []

# Collect file paths for relevant years (2022, 2023) for station
"72429793812"
for year in [2022, 2023]:
    file_path = os.path.join(base_directory, str(year),
'72429793812.csv')
    if os.path.exists(file_path):
        file_paths.append(file_path)

# Load all the CSV files into a single DataFrame
historical_data = spark.read.csv(file_paths, header=True,
inferSchema=True)

# Filter data for November and December (months 11 and 12) and for
station "72429793812"
historical_df = historical_data.filter(
    (col("STATION") == "72429793812") & (month("DATE").isin([11, 12]))
)

# Prepare the training data by adding the day of the year
training_data = historical_df.withColumn("DAY_OF_YEAR",
dayofyear("DATE"))

# Assemble the features
assembler = VectorAssembler(inputCols=["DAY_OF_YEAR"],
outputCol="features")
train_data = assembler.transform(training_data).select("features",
col("MAX").alias("label"))

# Train the Linear Regression model
lr = LinearRegression()
lr_model = lr.fit(train_data)
```

```
# Prepare data for predicting for each day in November and December
2024 (days 305 to 365 of the year)
predictions_df = spark.createDataFrame(
    [(day,) for day in range(305, 366)], ["DAY_OF_YEAR"]
)

# Transform the prediction data with the same assembler
predictions = assembler.transform(predictions_df)

# Generate predictions using the trained model
predicted_temps = lr_model.transform(predictions)

# Identify the maximum predicted temperature for November and December
max_predictions = predicted_temps.withColumn(
    "MONTH", when(col("DAY_OF_YEAR") < 335, 11).otherwise(12)
).groupBy("MONTH").agg(spark_max("prediction").alias("Max Predicted
Temp"))

# Show the maximum temperature predictions for November and December
2024
max_predictions.show()

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

```
+-----+------------------+
|MONTH|Max Predicted Temp|
+-----+------------------+
|   11| 65.22373881977461|
|   12| 53.08328332169012|
+-----+------------------+
```

```
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
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, max

# Initialize Spark session
spark = SparkSession.builder.appName("Max Snowfall Finder").getOrCreate()

base_input_dir = "./weather_data/"
year = "2022"
year_dir = base_input_dir + year

# Load all CSV files for the selected year
df = spark.read.option("header", "true").csv(year_dir + "/*.csv")
df = df.select(col("DATE"), col("STATION"), col("SNDP").cast("float"))
# Filter out null or missing snowfall values
df = df.filter(col("SNDP").isNotNull())

# Find the maximum snowfall value
max_snowfall = df.agg(max("SNDP")).collect()[0][0]

# Find the records with the maximum snowfall
max_snowfall_df = df.filter(col("SNDP") == max_snowfall)

# Show the result
max_snowfall_df.show()
spark.stop()
```

```
+----------+-----------+-----+
|      DATE|    STATION| SNDP|
+----------+-----------+-----+
|2022-01-01|72429793812|999.9|
|2022-01-02|72429793812|999.9|
|2022-01-03|72429793812|999.9|
|2022-01-04|72429793812|999.9|
|2022-01-05|72429793812|999.9|
|2022-01-06|72429793812|999.9|
|2022-01-07|72429793812|999.9|
|2022-01-08|72429793812|999.9|
|2022-01-09|72429793812|999.9|
|2022-01-10|72429793812|999.9|
|2022-01-11|72429793812|999.9|
|2022-01-12|72429793812|999.9|
|2022-01-13|72429793812|999.9|
```

```
|2022-01-14|72429793812|999.9|
|2022-01-15|72429793812|999.9|
|2022-01-16|72429793812|999.9|
|2022-01-17|72429793812|999.9|
|2022-01-18|72429793812|999.9|
|2022-01-19|72429793812|999.9|
|2022-01-20|72429793812|999.9|
+----------+-----------+-----+
only showing top 20 rows
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, max

# Initialize Spark session
spark = SparkSession.builder.appName("Max Snowfall Finder").getOrCreate()

base_input_dir = "./weather_data/"
year = "2021"
year_dir = base_input_dir + year

# Load all CSV files for the selected year
df = spark.read.option("header", "true").csv(year_dir + "/*.csv")
df = df.select(col("DATE"), col("STATION"), col("SNDP").cast("float"))
# Filter out null or missing snowfall values
df = df.filter(col("SNDP").isNotNull())

# Find the maximum snowfall value
max_snowfall = df.agg(max("SNDP")).collect()[0][0]

# Find the records with the maximum snowfall
max_snowfall_df = df.filter(col("SNDP") == max_snowfall)

# Show the result
max_snowfall_df.show()
spark.stop()
```

```
+----------+-----------+-----+
|      DATE|    STATION| SNDP|
+----------+-----------+-----+
|2021-01-01|72429793812|999.9|
|2021-01-02|72429793812|999.9|
|2021-01-03|72429793812|999.9|
|2021-01-04|72429793812|999.9|
|2021-01-05|72429793812|999.9|
|2021-01-06|72429793812|999.9|
|2021-01-07|72429793812|999.9|
|2021-01-08|72429793812|999.9|
|2021-01-09|72429793812|999.9|
|2021-01-10|72429793812|999.9|
|2021-01-11|72429793812|999.9|
|2021-01-12|72429793812|999.9|
|2021-01-13|72429793812|999.9|
|2021-01-14|72429793812|999.9|
|2021-01-15|72429793812|999.9|
|2021-01-16|72429793812|999.9|
|2021-01-17|72429793812|999.9|
|2021-01-18|72429793812|999.9|
|2021-01-19|72429793812|999.9|
|2021-01-20|72429793812|999.9|
+----------+-----------+-----+
only showing top 20 rows
```