```
import os
import requests
import pandas as pd

directory = r"C:\Users\benla\Desktop\Grad_School\Classes\
GIS5571_SpatialDataScience\Labs\Lab3\Data"
os.chdir(directory)

print(os.getcwd())
```

```
C:\Users\benla\Desktop\Grad_School\Classes\GIS5571_SpatialDataScience\
Labs\Lab3\Data
```

```
## Step 1: Download NDAWN temperature data from all stations for last
30 days

# URL for NDAWN
NDAWN_url = "https://ndawn.ndsu.nodak.edu/table.csv?
station=78&station=111&station=98&station=162&station=174&station=142&
station=164&station=138&station=161&station=9&station=160&station=224&
station=159&station=10&station=229&station=118&station=56&station=165&
station=11&station=12&station=58&station=13&station=84&station=218&sta
tion=55&station=179&station=7&station=186&station=87&station=14&statio
n=15&station=96&station=191&station=16&station=210&station=201&station
=137&station=124&station=143&station=17&station=85&station=226&station
=140&station=134&station=18&station=136&station=219&station=65&station
=104&station=99&station=192&station=19&station=227&station=129&station
=20&station=101&station=166&station=178&station=81&station=21&station=
97&station=22&station=75&station=184&station=2&station=211&station=172
&station=139&station=158&station=23&station=157&station=220&station=62
&station=86&station=24&station=89&station=126&station=223&station=167&
station=93&station=183&station=90&station=25&station=205&station=83&st
ation=107&station=156&station=77&station=26&station=155&station=70&sta
tion=127&station=144&station=27&station=173&station=132&station=28&sta
tion=195&station=185&station=29&station=30&station=154&station=31&stat
ion=187&station=102&station=32&station=119&station=4&station=217&stati
on=80&station=33&station=59&station=153&station=105&station=82&station
=225&station=34&station=198&station=72&station=135&station=35&station=
76&station=120&station=209&station=141&station=109&station=36&station=
207&station=79&station=193&station=71&station=212&station=37&station=3
8&station=189&station=39&station=130&station=73&station=188&station=40
&station=41&station=54&station=228&station=69&station=194&station=145&
station=214&station=113&station=128&station=42&station=43&station=103&
station=171&station=116&station=196&station=88&station=114&station=3&s
tation=163&station=200&station=216&station=64&station=115&station=168&
station=67&station=175&station=146&station=170&station=197&station=44&
station=206&station=133&station=106&station=100&station=121&station=45
&station=46&station=61&station=66&station=181&station=74&station=213&s
tation=60&station=199&station=125&station=176&station=177&station=8&st
ation=180&station=204&station=47&station=221&station=122&station=108&s
```

```
tation=5&station=152&station=48&station=151&station=147&station=68&sta
tion=169&station=49&station=50&station=91&station=182&station=117&stat
ion=63&station=150&station=51&station=6&station=222&station=52&station
=92&station=112&station=131&station=123&station=95&station=53&station=
203&station=190&station=208&station=57&station=149&station=148&station
=202&station=215&station=110&variable=ddavt&year=2024&ttype=daily&quic
k_pick=30_d&begin_date=2024-10-27&end_date=2024-11-25"

# API Request with Pandas
NDAWN_pd = pd.read_csv(NDAWN_url, skiprows=[0, 1, 2, 4])

# Combine Year, Month, and Day into a single Date column in MM/DD/YYYY
format
NDAWN_pd['Date'] = NDAWN_pd.apply(lambda row:
f"{int(row['Month']):02}/{int(row['Day']):02}/{int(row['Year'])}",
axis=1)

print(NDAWN_pd)
```

```
     Station Name  Latitude  Longitude  Elevation  Year  Month  Day
Avg Temp  \
0             Ada  47.32119  -96.51406        910  2024     10   27
46.162
1             Ada  47.32119  -96.51406        910  2024     10   28
54.805
2             Ada  47.32119  -96.51406        910  2024     10   29
51.004
3             Ada  47.32119  -96.51406        910  2024     10   30
37.497
4             Ada  47.32119  -96.51406        910  2024     10   31
32.550
...           ...       ...        ...        ...   ...    ...  ...
...
6529      Zeeland  46.01351  -99.68768       2070  2024     11   21
15.338
6530      Zeeland  46.01351  -99.68768       2070  2024     11   22
22.719
6531      Zeeland  46.01351  -99.68768       2070  2024     11   23
24.432
6532      Zeeland  46.01351  -99.68768       2070  2024     11   24
19.212
6533      Zeeland  46.01351  -99.68768       2070  2024     11   25
11.397

     Avg Temp Flag        Date
0              NaN  10/27/2024
1              NaN  10/28/2024
2              NaN  10/29/2024
3              NaN  10/30/2024
4              NaN  10/31/2024
```

```
...             ...         ...
6529            NaN  11/21/2024
6530            NaN  11/22/2024
6531            NaN  11/23/2024
6532            NaN  11/24/2024
6533            NaN  11/25/2024

[6534 rows x 10 columns]
```

```
## Step 2: Create a new feature class for the station points
gdb_path = r"C:\Users\benla\Desktop\Grad_School\Classes\
GIS5571_SpatialDataScience\Labs\Lab3\Lab3_aprx\Lab3_aprx.gdb"
station_points = os.path.join(gdb_path, "StationPoints")

arcpy.management.CreateFeatureclass(gdb_path, "StationPoints",
"POINT", spatial_reference=arcpy.SpatialReference(4326))

## Step 4: Add fields to the feature class
arcpy.management.AddField(station_points, "Station_Name", "TEXT")
arcpy.management.AddField(station_points, "Avg_Temp", "DOUBLE")
arcpy.management.AddField(station_points, "Date", "TEXT")

## Step 5: Populate the feature class with station points and average
temperatures
with arcpy.da.InsertCursor(station_points, ['SHAPE@XY',
'Station_Name', 'Avg_Temp', 'Date']) as cursor:
    for index, row in NDAWN_pd.iterrows():
        point = (row['Longitude'], row['Latitude'])
        cursor.insertRow([point, row['Station Name'], row['Avg Temp'],
row['Date']])

print("Station points added successfully to the feature class")
```

```
Station points added successfully to the feature class
```

```
# Step 3: Create a new feature class with a single point for each
station, aggregating the data by Station Name and calculating mean,
high, and low temperatures
aggregated_data = NDAWN_pd.groupby('Station Name').agg({
    'Avg Temp': ['mean', 'max', 'min'],     # Mean, max, and min of
the temperature column
    'Latitude': 'first',                    # Latitude of the station
    'Longitude': 'first'                    # Longitude of the station
}).reset_index()

# Flatten the multi-level column headers from the aggregation
aggregated_data.columns = ['Station Name', 'Avg_Temp',
'High_Avg_Temp', 'Low_Avg_Temp', 'Latitude', 'Longitude']

# Step 2: Define the path for the new feature class
aggregated_fc_path = os.path.join(gdb_path, "AggStationPoints")
```

```
# Step 3: Create the new feature class
arcpy.management.CreateFeatureclass(gdb_path, "AggStationPoints",
"POINT", spatial_reference=arcpy.SpatialReference(4326))

# Step 4: Add fields to the new feature class
arcpy.management.AddField(aggregated_fc_path, "Station_Name", "TEXT")
arcpy.management.AddField(aggregated_fc_path, "Avg_Temp", "DOUBLE")
arcpy.management.AddField(aggregated_fc_path, "High_Avg_Temp",
"DOUBLE")
arcpy.management.AddField(aggregated_fc_path, "Low_Avg_Temp",
"DOUBLE")

# Step 5: Populate the new feature class with the aggregated data
with arcpy.da.InsertCursor(aggregated_fc_path, ['SHAPE@XY',
'Station_Name', 'Avg_Temp', 'High_Avg_Temp', 'Low_Avg_Temp']) as
cursor:
    for index, row in aggregated_data.iterrows():
        point = (row['Longitude'], row['Latitude'])
        cursor.insertRow([point, row['Station Name'], row['Avg_Temp'],
row['High_Avg_Temp'], row['Low_Avg_Temp']])

print("New feature class created with aggregated data including high
and low temperatures.")
```

```
New feature class created with aggregated data including high and low
temperatures.
```

```
## Step 4: Interpolate data - One Prediction per Location (IDW) - High
Temps

# Set the input feature class and output raster path for High Temp IDW
interpolation
input_fc = os.path.join(gdb_path, "AggStationPoints")
idw_output_raster_high_temp = os.path.join(gdb_path, "IDW_High_Temp")

# Perform IDW interpolation
idw_high_temp_result = arcpy.sa.Idw(input_fc, "High_Avg_Temp",
cell_size=0.01)
idw_high_temp_result.save(idw_output_raster_high_temp)

print("High Temp IDW interpolation added to the map")
```

```
High Temp IDW interpolation added to the map
```

```
## Step 5: Interpolate data - One Prediction per Location (IDW) - Low
Temps

# Set the input feature class and output raster path for Low Temp IDW
interpolation
idw_output_raster_low_temp = os.path.join(gdb_path, "IDW_Low_Temp")
```

```
# Perform IDW interpolation
idw_low_temp_result = arcpy.sa.Idw(input_fc, "Low_Avg_Temp",
cell_size=0.01)
idw_low_temp_result.save(idw_output_raster_low_temp)

print("Low Temp IDW interpolation added to the map")
```

```
Low Temp IDW interpolation added to the map
```

```
## Step 6: Interpolate data - Quantile Value (Kriging) - High Temps

# Set the output raster path for High Temp Kriging interpolation
kriging_output_raster_high_temp = os.path.join(gdb_path,
"Kriging_High_Temp")

# Perfrom the Kriging interpolation
kriging_high_temp_result = arcpy.sa.Kriging(
    input_fc,
    "High_Avg_Temp",
    "Spherical",
    0.01,
)

kriging_high_temp_result.save(kriging_output_raster_high_temp)

print("High Temp Kriging added to the map with default parameters.")
```

```
High Temp Kriging added to the map with default parameters.
```

```
## Step 7: Interpolate data - Quantile Value (Kriging) - Low Temps

# Set the output raster path for Low Temp Kriging interpolation
kriging_output_raster_low_temp = os.path.join(gdb_path,
"Kriging_Low_Temp")

# Perfrom the Kriging interpolation
kriging_low_temp_result = arcpy.sa.Kriging(
    input_fc,
    "Low_Avg_Temp",
    "Spherical",
    0.01,
)

kriging_low_temp_result.save(kriging_output_raster_low_temp)

print("Low Temp Kriging added to the map with default parameters.")
```

```
Low Temp Kriging added to the map with default parameters.
```

```
## Step 8: Interpolate data - Many Predicitions per Location
(Simulation) - High Temps

# Set the output path for High Temp Natural Neighbor interpolation
nat_neigh_output_raster_high_temp = os.path.join(gdb_path,
"NaturalNeighbor_High_Temp")

# Perform Natural Neighbor interpolation
nat_neigh_high_temp_result = arcpy.sa.NaturalNeighbor(input_fc,
"High_Avg_Temp", cell_size=0.01)
nat_neigh_high_temp_result.save(nat_neigh_output_raster_high_temp)

print("High Temp Natural Neighbor interpolation added to the map as a
substitute for simulation")
```

```
High Temp Natural Neighbor interpolation added to the map as a
substitute for simulation
```

```
## Step 9: Interpolate data - Many Predicitions per Location
(Simulation) - Low Temps

# Set the output path for Low Temp Natural Neighbor interpolation
nat_neigh_output_raster_low_temp = os.path.join(gdb_path,
"NaturalNeighbor_Low_Temp")

# Perform Natural Neighbor interpolation
nat_neigh_low_temp_result = arcpy.sa.NaturalNeighbor(input_fc,
"Low_Avg_Temp", cell_size=0.01)
nat_neigh_low_temp_result.save(nat_neigh_output_raster_low_temp)

print("Low Temp Natural Neighbor interpolation added to the map as a
substitute for simulation")
```

```
Low Temp Natural Neighbor interpolation added to the map as a
substitute for simulation
```