

New York City Daily Temperature Data Processing and Analysis

Selen Ozdogan

July 31, 2025

Purpose

This document aims to provide a comprehensive guide to downloading and cleaning daily temperature and precipitation data for New York City between 2015-2022. The workflow integrates two primary data sources:

1. **Global Historical Climatology Network daily (GHCNd):** Daily climate summaries from the U.S. National Centers for Environmental Information's (NCEI) global network of weather stations.
2. **ERA5-Land Reanalysis:** High-resolution hourly estimates of atmospheric and land-surface conditions from the European Union's Copernicus Project.

The final output is an analysis-ready data set of daily temperature and precipitation measures, including calculated wet bulb temperature. To facilitate time-series analysis, the script also generates various temporal exposure lags. You can easily change the spatial extent and time period in this script for other applications.

Setup

We will use RStudio Projects and the {here} package to ensure the code is portable. This practice avoids hard-coding file paths, a common source of errors when sharing code.

As a first step, create an RStudio Project (.Rproj file) in your desired project directory.

Part 1: Load and Process GHCHd Weather Station Data

This section downloads data from 3 individual GHCNd weather stations and aggregates it to generate a single set of daily, city-wide metrics for New York City.

1.1: Load Libraries

This code chunk loads the necessary R packages, installing them automatically if they are not already present using the {pacman} package.

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tidyverse, lubridate, magrittr, here)
```

1.2: Identify and Read NYC Weather Station Data

First, we define the identifiers for the three main GHCNd weather stations in New York City. A full list of all stations is available from the NCEI ([here](#)). We then use these identifiers to load the daily summary data directly from the NCEI website.

```

# Define the station IDs for NYC (Central Park, JFK, LaGuardia)
station_ids <- c("USW00094728", "USW00094789", "USW00014732")

# Define the base URL for data access
base_url <- "https://www.ncei.noaa.gov/data/daily-summaries/access/"

# Loop over the stations, read the csv file from the URL, and combine all data
all_station_data <- station_ids %>%
  map_dfr(~read_csv(
    paste0(base_url, .x, ".csv"),
    show_col_types = FALSE
  ))

# Check out the first few rows of the data
print(head(all_station_data))

## # A tibble: 6 x 136
##   STATION    DATE      LATITUDE LONGITUDE ELEVATION NAME   PRCP PRCP_ATTRIBUTES
##   <chr>      <date>      <dbl>     <dbl>     <dbl> <chr> <dbl> <chr>
## 1 USW000947~ 1869-01-01    40.8      -74.0      42.7 NY C~   191 ,,Z,
## 2 USW000947~ 1869-01-02    40.8      -74.0      42.7 NY C~    8 ,,Z,null
## 3 USW000947~ 1869-01-03    40.8      -74.0      42.7 NY C~    0 T,,Z,null
## 4 USW000947~ 1869-01-04    40.8      -74.0      42.7 NY C~   46 ,,Z,null
## 5 USW000947~ 1869-01-05    40.8      -74.0      42.7 NY C~   13 ,,Z,null
## 6 USW000947~ 1869-01-06    40.8      -74.0      42.7 NY C~    0 ,,Z,null
## # i 128 more variables: SNOW <dbl>, SNOW_ATTRIBUTES <chr>, SNWD <dbl>,
## #   SNWD_ATTRIBUTES <chr>, TMAX <dbl>, TMAX_ATTRIBUTES <chr>, TMIN <dbl>,
## #   TMIN_ATTRIBUTES <chr>, ACMH <dbl>, ACMH_ATTRIBUTES <chr>, ACSH <dbl>,
## #   ACSH_ATTRIBUTES <chr>, ADPT <dbl>, ADPT_ATTRIBUTES <chr>, ASLP <dbl>,
## #   ASLP_ATTRIBUTES <chr>, ASTP <dbl>, ASTP_ATTRIBUTES <chr>, AWBT <dbl>,
## #   AWBT_ATTRIBUTES <chr>, AWND <dbl>, AWND_ATTRIBUTES <chr>, DAEV <lgl>,
## #   DAEV_ATTRIBUTES <lgl>, DASF <lgl>, DASF_ATTRIBUTES <lgl>, DAWM <lgl>, ...

```

1.3: Clean and Process GHCHd Data

Next, we clean the data by dropping dates and variables we don't need, changing the units of the variables. We then aggregate the data from the three stations to city-wide daily measures.

```

ghchd_clean <- all_station_data %>%

  # Create a clean date variable and separate variables
  mutate(date = ymd(DATE),
    year = year(date),
    month = month(date),
    day_of_year = yday(date)) %>%

  # Drop the original date variable
  dplyr::select(-DATE) %>%

  # Keep only 2015-2022, our period of interest
  filter(year <= 2022 & year >= 2015) %>%

  # Keep daily max and min temperature, and precipitation variables

```

```

dplyr::select(date, TMAX, TMIN, PRCP) %>%

# Group by date (to aggregate variables citywide using the three stations)
group_by(date) %>%

# Calculate mean maximum and minimum temperatures (tmax, tmin) across stations
# Data in tenths of degrees Celsius, so divide by 10 to convert to Celsius
# Get average precip across stations and convert from tenths of mm to inches
summarise(tmax_c = mean(TMAX / 10, na.rm = TRUE),
          tmin_c = mean(TMIN / 10, na.rm = TRUE),
          precip_in = mean(PRCP / 254, na.rm = TRUE)) %>%

# Create temperatures in Fahrenheit
mutate(tmax_f = (tmax_c * 9/5) + 32,
       tmin_f = (tmin_c * 9/5) + 32)

# Check out the first and last few rows of the cleaned data
print(head(ghchd_clean))

## # A tibble: 6 x 6
##   date      tmax_c tmin_c precip_in tmax_f tmin_f
##   <date>    <dbl> <dbl>    <dbl> <dbl> <dbl>
## 1 2015-01-01  4.07 -2.5      0      39.3  27.5
## 2 2015-01-02  6.13  1.7      0      43.0  35.1
## 3 2015-01-03  6.3  -0.3     0.773  43.3  31.5
## 4 2015-01-04 13.3   5.37    0.360  56    41.7
## 5 2015-01-05 10     -5.47    0      50    22.2
## 6 2015-01-06 -5.3  -7.3     0.0577 22.5  18.9

print(tail(ghchd_clean))

## # A tibble: 6 x 6
##   date      tmax_c tmin_c precip_in tmax_f tmin_f
##   <date>    <dbl> <dbl>    <dbl> <dbl> <dbl>
## 1 2022-12-26 -1.4  -7.5      0      29.5  18.5
## 2 2022-12-27  1.7  -1.77     0      35.1  28.8
## 3 2022-12-28  7.93 -0.5      0      46.3  31.1
## 4 2022-12-29  9.67  2.57     0      49.4  36.6
## 5 2022-12-30 14.3   4.07     0      57.7  39.3
## 6 2022-12-31 12.1   6.87    0.287  53.7  44.4

# Create a directory to save the clean data
if (!dir.exists("data/clean")) {
  dir.create("data/clean")
}

# Save as a csv file
write_csv(ghchd_clean, "data/clean/nyc_2015_2022_ghchd.csv")

```

Part 2: Download and Process ERA5 Raster Data

This section outlines the process for downloading hourly ERA5 reanalysis data and processing it into daily wet bulb temperatures.

2.1: Load Libraries

This code chunk loads additional packages required for spatial data processing and statistical analysis.

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tidyverse, lubridate, magrittr, here, sf, raster, exactextractr, openxlsx, fixest, slides)
```

2.2: Download ERA5 Data using CDS API and Python

ERA5 data can be downloaded using a Python script that calls the Climate Data Store (CDS) API. This requires a one-time setup of the CDS API on your machine, with instructions available [here](#).

Once your API is configured, run the accompanying Python script, `install_era5.py`, to download the data into a local directory. This script and the following R functions are adapted from the [replication package](#) for LoPalo (2023).

To customize your download for different variables or time periods, you can use the online download tool ([here](#)) to automatically generate a new Python script.

2.3: Unzip the Downloaded ERA5 Data

Due to large file sizes, the ERA5 data for each year is downloaded in three 4-month chunks. This R code prepares the downloaded data by iterating through each .zip archive and unzipping the files into a specified directory. The code also renames the unzipped files.

```
# Define years and chunks
years_to_unzip <- 2015:2022
chunks <- 1:3

# Define the directory
climate_path <- here("data", "era5")

# Unzip and rename files
expand_grid(year = years_to_unzip, chunk = chunks) %>%
  pwalk(~{
    zip_file <- here(climate_path, paste0("era5_", .x, "_", .y, ".zip"))

    unzip(zipfile = zip_file, exdir = climate_path)

    current_file <- here(climate_path, "data_0.nc")
    new_file      <- here(climate_path, paste0("era5_", .x, "_", .y, ".nc"))

    file.rename(from = current_file, to = new_file)
  })
```

2.4: Download NYC Boundary Shapefile

To aggregate the ERA5 raster data to a city-wide value, we first need a polygon defining the geographic area of New York City. This R code chunk loads the “Borough Boundaries” shapefile from the NYC Planning website, re-projects it to an equal-area projection (CRS: 2163) for accurate calculations, and unifies the five borough polygons into a single feature.

```

# Create a directory to store the NYC boundary data
boundary_dir <- "data/boundaries"

if (!dir.exists(boundary_dir)) {
  dir.create(boundary_dir, recursive = TRUE, showWarnings = FALSE)
}

# Define the full path to the shapefile
shp_path <- file.path(boundary_dir, "nybb_25b/nybb.shp")

# Define the URL and the local destination for the zip file
nyc_url <- "https://s-media.nyc.gov/agencies/dcp/assets/files/zip/data-tools/bytes/borough-boundaries/nybb.zip"
zip_dest <- file.path(boundary_dir, "nybb.zip")

# Download the zipped data
download.file(nyc_url, destfile = zip_dest)

# Unzip the downloaded file into our local directory
unzip(zip_dest, exdir = boundary_dir)

# Read in the shapefile
nyc_boundary <- st_read(shp_path) %>%

# Project to US National Atlas Equal Area for accurate area calculations
st_transform(crs = st_crs(2163)) %>%

# Merge all borough polygons into a single feature
st_union()

## Reading layer `nybb' from data source
##   `/Users/selenozd/Library/CloudStorage/Dropbox/hartisland_weather/cache_nyc_temp/data/boundaries/nybb.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 5 features and 4 fields
## Geometry type: MULTIPOLYGON
## Dimension:      XY
## Bounding box:   xmin: 913175.1 ymin: 120128.4 xmax: 1067383 ymax: 272844.3
## Projected CRS: NAD83 / New York Long Island (ftUS)

# Plot to confirm the boundary was loaded correctly
plot(st_geometry(nyc_boundary), main = "NYC Boundary")

```

NYC Boundary



2.5: Define Functions to Calculate Relative Humidity and Wet Bulb Temperature

This section defines two R functions to derive relative humidity (RH) and wet bulb temperature from the original ERA5 variables.

The first function, `calc_rh`, computes RH using air temperature and dewpoint temperature. The second function, `calc_wetbtemp`, uses the resulting RH and air temperature to estimate the wet bulb temperature.

```
# Define a function to calculate relative humidity (RH)
calc_rh <- function(temp_k, dewp_k) {

  # The original ERA5 data is in Kelvin. The calculations require Celsius.
  # Convert Kelvin to Celsius
  temp_c <- temp_k - 273.15
  dewp_c <- dewp_k - 273.15

  # Define constants for the formula
  a <- 17.625
  b <- 243.04

  # Calculate numerator and denominator of the RH formula
  numer <- exp((a * dewp_c) / (b + dewp_c))
  denom <- exp((a * temp_c) / (b + temp_c))

  # Calculate relative humidity
  rh <- 100 * (numer / denom)
  return(rh)
}

# Define a function to calculate wet bulb temperature, using relative humidity
calc_wetbtemp <- function(rh, temp_c) {
  # define constants for the formula
  k <- 0.151977
  l <- 8.313659
  m <- 1.676331
```

```

n <- 0.00391838
p <- 0.023101
s <- 4.686035

# calculate wet bulb temperature
wetbtemp <- temp_c * atan((k*(rh + 1))^(1/2)) +
  atan(temp_c + rh) - atan(rh - m) +
  (n*(rh^(3/2))) * atan(p*rh) - s

return(wetbtemp)
}

```

2.6: Define a Function to Process Data for a Single Year

To make the code repeatable and organized, we'll wrap the workflow for a single year into a function. This function takes the year, file path, and NYC boundary as inputs and returns a clean data frame of daily climate summaries.

```

process_yearly_data <- function(year, climate_path, nyc_boundary) {

  # 1 # Read Raster Data

  # Define the variables to extract from the raw files
  vars <- list("d2m", "t2m") # d2m: dew point, t2m: air temp

  # Loop through the year chunks & variables, stack, and reproject data
  for (i in 1:3) {
    for (v in vars) {
      name <- brick(file.path(climate_path,
                              paste0("era5_", year, "_", i, ".nc")),
                    varname = v) %>%
        projectRaster(crs = crs(nyc_boundary))
      assign(paste0(v, i), name)
    }
  }
  rm(name)

  #2# Append chunks and get one brick for each year and variable
  d2m <- stack(d2m1, d2m2, d2m3)
  t2m <- stack(t2m1, t2m2, t2m3)
  rm(d2m1, d2m2, d2m3, t2m1, t2m2, t2m3)

  # 2 # Calculate Wet Bulb Temperature

  # Calculate relative humidity
  rh <- calc_rh(t2m, d2m)

  # Convert air temperature to Celsius and calculate wet bulb temperature
  t2m_c <- t2m - 273.15
  wetbtemp_c <- calc_wetbtemp(rh, t2m_c)

  # Convert final wet bulb temperature from Celsius to Fahrenheit
  wetbtemp_f <- (wetbtemp_c * 1.8) + 32

```

```

# 3 # Aggregate Hourly Data to Daily Summaries

# Get the number of layers in the wet bulb temperature brick
num_lyrs <- nlayers(wetbtemp_f)

# Initialize empty lists to store the new day layers
daily_lyrs_mean <- list()
daily_lyrs_min <- list()
daily_lyrs_max <- list()

# Initialize iteration
iter <- 0

# Loop through the layers in groups of 24 (hours of the day)
for (i in seq(1, num_lyrs, by = 24)) {

  # Subset the brick to get the hours of a particular day
  subset <- wetbtemp_f[[i:(i + 24 - 1)]]

  # Calculate the daily summaries
  daily_mean <- calc(subset, mean)
  daily_min <- calc(subset, min)
  daily_max <- calc(subset, max)

  # Increase the iteration
  iter <- iter + 1

  # Add the day layer to the list
  daily_lyrs_mean[[iter]] <- daily_mean
  daily_lyrs_min[[iter]] <- daily_min
  daily_lyrs_max[[iter]] <- daily_max
}

# Create a brick from the list of layers
daily_wetb_mean <- brick(daily_lyrs_mean)
daily_wetb_min <- brick(daily_lyrs_min)
daily_wetb_max <- brick(daily_lyrs_max)

# 4 # Spatially Aggregate to NYC Boundary

# Get spatial max and weighted mean of maximum temperature
df_max <- exact_extract(daily_wetb_max, nyc_boundary,
                        c("max", "weighted_mean"),
                        weights = "area",
                        force_df = TRUE,
                        stack_apply = TRUE)

# Get spatial min and weighted mean of minimum temperature
df_min <- exact_extract(daily_wetb_min, nyc_boundary,
                        fun = c("min", "weighted_mean"),
                        weights = "area",
                        stack_apply = TRUE,
                        force_df = TRUE)

```



```

# Get spatial weighted mean of mean wet bulb temperature
df_mean <- exact_extract(daily_wetb_mean, nyc_boundary,
                        fun = "weighted_mean",
                        weights = "area",
                        stack_apply = TRUE,
                        force_df = TRUE)

# 5 # Clean and Combine Data Frames
df_max %<>%
  pivot_longer(everything(),
               names_to = c(".value", "day"),
               names_pattern = "(\\w+)\\.layer\\.((\\d+))" %>%
  rename(wetb_max_citymax = max, wetb_max_citymean = weighted_mean)

df_min %<>%
  pivot_longer(everything(),
               names_to = c(".value", "day"),
               names_pattern = "(\\w+)\\.layer\\.((\\d+))" %>%
  rename(wetb_min_citymin = min, wetb_min_citymean = weighted_mean)

df_mean %<>%
  pivot_longer(everything(),
               names_to = c(".value", "day"),
               names_pattern = "(\\w+)\\.layer\\.((\\d+))" %>%
  rename(wetb_mean_citymean = weighted_mean)

# Join all data frames into one, create a date variable
yearly_df <- list(df_max, df_min, df_mean) %>%
  reduce(full_join, by = "day") %>%
  mutate(
    year = year,
    day = as.integer(day),
    date = make_date(year, 1, 1) + days(day - 1)
  ) %>%
  dplyr::select(date, year, day, everything())

return(yearly_df)
}

```

2.7: Process All Years and Combine Data

With the main processing function defined, we can now iterate over the entire study period (2015-2022). We use the `map_dfr()` function to apply the processing function created earlier to each year in our list. This function efficiently executes the workflow for every year and row-binds the results into a single data frame. Note that processing eight years of hourly raster data is computationally intensive and may take some time to complete.

```

# Define the full range of years we want to process
years_to_process <- 2015:2022

# Use map_dfr() to iterate through each year and combine the results
era5_clean <- map_dfr(years_to_process, ~process_yearly_data(
  year = .x,

```

```

climate_path = here("data", "era5"),
nyc_boundary = nyc_boundary
))

# Arrange the final data frame by date
era5_clean %<>%
  arrange(date)

# Let's inspect the first and last few rows of our complete dataset
print(head(era5_clean))

## # A tibble: 6 x 8
##   date      year  day wetb_max_citymax wetb_max_citymean wetb_min_citymin
##   <date>    <int> <int>          <dbl>          <dbl>          <dbl>
## 1 2015-01-01  2015     1            32.0            30.0            15.3
## 2 2015-01-02  2015     2            37.5            36.8            24.9
## 3 2015-01-03  2015     3            44.1            40.9            21.6
## 4 2015-01-04  2015     4            65.7            63.2            37.2
## 5 2015-01-05  2015     5            64.7            60.4            20.1
## 6 2015-01-06  2015     6            21.5            20.3             8.24
## # i 2 more variables: wetb_min_citymean <dbl>, wetb_mean_citymean <dbl>
print(tail(era5_clean))

## # A tibble: 6 x 8
##   date      year  day wetb_max_citymax wetb_max_citymean wetb_min_citymin
##   <date>    <int> <int>          <dbl>          <dbl>          <dbl>
## 1 2022-12-26  2022   360            22.1            21.9             7.75
## 2 2022-12-27  2022   361            30.8            30.0            15.1
## 3 2022-12-28  2022   362            42.7            41.8            19.0
## 4 2022-12-29  2022   363            47.3            46.5            24.2
## 5 2022-12-30  2022   364            57.2            56.1            31.1
## 6 2022-12-31  2022   365            60.3            59.0            36.7
## # i 2 more variables: wetb_min_citymean <dbl>, wetb_mean_citymean <dbl>

# Create a directory to store the clean data
clean_dir <- "data/clean"

if (!dir.exists(clean_dir)) {
  dir.create(clean_dir, recursive = TRUE, showWarnings = FALSE)
}

# Save the data as a csv file
write.csv(era5_clean, "data/clean/nyc_2015_2022_era5.csv")

```

Part 3: Merge GHCDd (Air Temp) and ERA5 (Wet Bulb Temp) Data and Create Lagged Measures

The final stage of the workflow involves integrating the processed GHCNd and ERA5 data sets and then generating lagged variables suitable for time-series analysis. The goal is to produce a single, analysis-ready data frame containing all relevant climate measures.

3.1: Merge GHCDd and ERA5 Data

This step joins the daily GHCNd data (which includes air temperature and precipitation) with the daily ERA5 data (containing the calculated wet bulb temperature).

```
# Merge the GHCDd and ERA5 summaries by date
daily_climate_data <- full_join(ghchd_clean, era5_clean, by = "date") %>%

# Sort by date
arrange(date) %>%

# Drop variable we don't need
dplyr::select(-c("tmax_c", "tmin_c"))

# Take a look at the data
glimpse(daily_climate_data)

## Rows: 2,922
## Columns: 11
## $ date                <date> 2015-01-01, 2015-01-02, 2015-01-03, 2015-01-04, 20~
## $ precip_in           <dbl> 0.000000000, 0.000000000, 0.77296588, 0.35958005, 0.0~
## $ tmax_f              <dbl> 39.32, 43.04, 43.34, 56.00, 50.00, 22.46, 24.50, 21~
## $ tmin_f              <dbl> 27.50, 35.06, 31.46, 41.66, 22.16, 18.86, 9.86, 8.2~
## $ year                <int> 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 201~
## $ day                 <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ~
## $ wetb_max_citymax    <dbl> 31.96522, 37.52550, 44.08417, 65.66180, 64.68448, 2~
## $ wetb_max_citymean   <dbl> 30.02267, 36.84089, 40.85118, 63.18095, 60.35521, 2~
## $ wetb_min_citymin    <dbl> 15.325259, 24.854813, 21.634148, 37.216013, 20.0777~
## $ wetb_min_citymean   <dbl> 16.413284, 26.049728, 23.060247, 42.711773, 21.8284~
## $ wetb_mean_citymean  <dbl> 21.197086, 29.491789, 29.356157, 52.972916, 35.7814~
```

3.2: Generate Lagged Temperature Exposure Variables

In this final step, we generate lagged variables, which are often required for time-series models. More specifically, this code chunk calculates rolling rolling maximums, minimums, and means over 3-day and 7-day windows for air and wet bulb temperature variables.

```
# Define the variables to process
vars_for_max <- c("tmax_f", "wetb_max_citymax", "wetb_max_citymean")
vars_for_min <- c("tmin_f", "wetb_min_citymin", "wetb_min_citymean")
vars_for_mean <- c("wetb_mean_citymean")
all_vars_for_lags <- c(vars_for_max, vars_for_min)

# Create rolling statistics for temperatures
daily_climate_data <- daily_climate_data %>%
  mutate(
    across(
      .cols = all_of(vars_for_max),
      .fns = list(
        "3d" = ~ slide_dbl(.x, max, .before = 2, .complete = FALSE),
        "7d" = ~ slide_dbl(.x, max, .before = 6, .complete = FALSE)
      ),
      .names = "{.col}_{.fn}"
    )
  )
```

```

),

across(
  .cols = all_of(vars_for_min),
  .fns = list(
    "3d" = ~ slide_dbl(.x, min, .before = 2, .complete = FALSE),
    "7d" = ~ slide_dbl(.x, min, .before = 6, .complete = FALSE)
  ),
  .names = "{.col}_{.fn}"
),

across(
  .cols = all_of(vars_for_mean),
  .fns = list(
    "3d" = ~ slide_dbl(.x, mean, .before = 2, .complete = FALSE),
    "7d" = ~ slide_dbl(.x, mean, .before = 6, .complete = FALSE)
  ),
  .names = "{.col}_{.fn}"
),

)

# View the resulting dataset
glimpse(daily_climate_data)

## Rows: 2,922
## Columns: 25
## $ date                <date> 2015-01-01, 2015-01-02, 2015-01-03, 2015-01-04, ~
## $ precip_in           <dbl> 0.000000000, 0.000000000, 0.77296588, 0.35958005, ~
## $ tmax_f              <dbl> 39.32, 43.04, 43.34, 56.00, 50.00, 22.46, 24.50, ~
## $ tmin_f              <dbl> 27.50, 35.06, 31.46, 41.66, 22.16, 18.86, 9.86, ~
## $ year                <int> 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, ~
## $ day                 <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1~
## $ wetb_max_citymax    <dbl> 31.96522, 37.52550, 44.08417, 65.66180, 64.68448~
## $ wetb_max_citymean   <dbl> 30.02267, 36.84089, 40.85118, 63.18095, 60.35521~
## $ wetb_min_citymin    <dbl> 15.325259, 24.854813, 21.634148, 37.216013, 20.0~
## $ wetb_min_citymean   <dbl> 16.413284, 26.049728, 23.060247, 42.711773, 21.8~
## $ wetb_mean_citymean  <dbl> 21.197086, 29.491789, 29.356157, 52.972916, 35.7~
## $ tmax_f_3d           <dbl> 39.32, 43.04, 43.34, 56.00, 56.00, 56.00, 50.00, ~
## $ tmax_f_7d           <dbl> 39.32, 43.04, 43.34, 56.00, 56.00, 56.00, 56.00, ~
## $ wetb_max_citymax_3d <dbl> 31.96522, 37.52550, 44.08417, 65.66180, 65.66180~
## $ wetb_max_citymax_7d <dbl> 31.96522, 37.52550, 44.08417, 65.66180, 65.66180~
## $ wetb_max_citymean_3d <dbl> 30.02267, 36.84089, 40.85118, 63.18095, 63.18095~
## $ wetb_max_citymean_7d <dbl> 30.02267, 36.84089, 40.85118, 63.18095, 63.18095~
## $ tmin_f_3d           <dbl> 27.50, 27.50, 27.50, 31.46, 22.16, 18.86, 9.86, ~
## $ tmin_f_7d           <dbl> 27.50, 27.50, 27.50, 27.50, 22.16, 18.86, 9.86, ~
## $ wetb_min_citymin_3d <dbl> 15.325259, 15.325259, 15.325259, 21.634148, 20.0~
## $ wetb_min_citymin_7d <dbl> 15.325259, 15.325259, 15.325259, 15.325259, 15.3~
## $ wetb_min_citymean_3d <dbl> 16.413284, 16.413284, 16.413284, 23.060247, 21.8~
## $ wetb_min_citymean_7d <dbl> 16.413284, 16.413284, 16.413284, 16.413284, 16.4~
## $ wetb_mean_citymean_3d <dbl> 21.197086, 25.344438, 26.681678, 37.273621, 39.3~
## $ wetb_mean_citymean_7d <dbl> 21.19709, 25.34444, 26.68168, 33.25449, 33.75989~

```

```
# Export as csv
write.csv(daily_climate_data, "data/clean/nyc_2015_2022_climate_data.csv")
```

References

- LoPalo, M. (2023). Temperature, Worker Productivity, and Adaptation: Evidence from Survey Data Production. *American Economic Journal: Applied Economics*, 15(1), 192–229. <https://doi.org/10.1257/app.20200547>
- Menne, M. J., Durre, I., Korzeniewski, B., McNeill, S., Thomas, K., Yin, X., Anthony, S., Ray, R., Vose, R. S., Gleason, B. E., & Houston, T. G. (2012). Global Historical Climatology Network - Daily (GHCN-Daily), Version 3. NOAA National Centers for Environmental Information. <https://doi.org/10.7289/V5D21VHZ>
- Muñoz Sabater, J. (2021). ERA5-Land hourly data from 1950 to present. Copernicus Climate Change Service (C3S) Climate Data Store (CDS). Accessed July 31, 2025. <https://doi.org/10.24381/cds.e2161bac>
- New York City Department of City Planning. (2025). Borough Boundaries (Clipped to Shoreline). NYC Open Data. Accessed July 31, 2025, from <https://www.nyc.gov/content/planning/pages/resources/datasets/borough-boundaries>
- Omni Calculator. Relative Humidity Calculator. Accessed July 31, 2025, from <https://www.omnicalculator.com/physics/relative-humidity>
- Omni Calculator. Wet Bulb Temperature Calculator. Accessed July 31, 2025, from <https://www.omnicalculator.com/physics/wet-bulb>