```
# -*- coding: utf-8 -*-
"""daily-temp-and-precip-averages_from_monthly-averages-data_release.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/116oKHQXvbE_m5l_BlknvSCHp1j5vdlz8
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.interpolate import CubicSpline

#
https://github.com/jambao24/MATLAB_climatedata_modeling/blob/main/TempSimul3%
20original%20code.rtf

# assign the monthly average temp for each day of a month to a numpy array
with 365 elements
# assumption: temps is an numpy array with 12 elements
def assign_monthly_avg_to_days(temps):
  temps_out = np.zeros(365)
  # initialize the values of temps_out to the monthly averages in temps
  # daily average for each day is the monthly average for that month
  temps_out[0:31] = temps[0]
  temps_out[31:59] = temps[1]
  temps_out[59:90] = temps[2]
  temps_out[90:120] = temps[3]
  temps_out[120:151] = temps[4]
  temps_out[151:181] = temps[5]
  temps_out[181:212] = temps[6]
  temps_out[212:243] = temps[7]
  temps_out[243:273] = temps[8]
  temps_out[273:304] = temps[9]
  temps_out[304:334] = temps[10]
  temps_out[334:365] = temps[11]
  return temps_out

# assumption: annual_temps is a numpy array with 365 elements
def calculate_mean_monthly_temps(annual_temps):
  monthly_temps = np.zeros(12)
  monthly_temps[0] = np.mean(annual_temps[0:31])
  monthly_temps[1] = np.mean(annual_temps[31:59])
  monthly_temps[2] = np.mean(annual_temps[59:90])
  monthly_temps[3] = np.mean(annual_temps[90:120])
  monthly_temps[4] = np.mean(annual_temps[120:151])
  monthly_temps[5] = np.mean(annual_temps[151:181])
  monthly_temps[6] = np.mean(annual_temps[181:212])
  monthly_temps[7] = np.mean(annual_temps[212:243])
  monthly_temps[8] = np.mean(annual_temps[243:273])
  monthly_temps[9] = np.mean(annual_temps[273:304])
  monthly_temps[10] = np.mean(annual_temps[304:334])
  monthly_temps[11] = np.mean(annual_temps[334:365])
  monthly_temps = np.round(monthly_temps, 2)
```

```
  return monthly_temps

# assumption: annual_precip is a numpy array with 365 elements
def calculate_monthly_precip_from_daily(annual_precip):
    monthly_precip = np.zeros(12)
    monthly_precip[0] = np.sum(annual_precip[0:31])
    monthly_precip[1] = np.sum(annual_precip[31:59])
    monthly_precip[2] = np.sum(annual_precip[59:90])
    monthly_precip[3] = np.sum(annual_precip[90:120])
    monthly_precip[4] = np.sum(annual_precip[120:151])
    monthly_precip[5] = np.sum(annual_precip[151:181])
    monthly_precip[6] = np.sum(annual_precip[181:212])
    monthly_precip[7] = np.sum(annual_precip[212:243])
    monthly_precip[8] = np.sum(annual_precip[243:273])
    monthly_precip[9] = np.sum(annual_precip[273:304])
    monthly_precip[10] = np.sum(annual_precip[304:334])
    monthly_precip[11] = np.sum(annual_precip[334:365])
    return monthly_precip

# assumption: monthly_precip is a numpy array with 12 elements
# this can also be used for computing probability of precip on any one day
from num days of precip in a month
def assign_initial_daily_precip(monthly_precip):
  precip_daily = np.zeros(365)
  precip_daily[0:31] = monthly_precip[0]/31
  precip_daily[31:59] = monthly_precip[1]/28
  precip_daily[59:90] = monthly_precip[2]/31
  precip_daily[90:120] = monthly_precip[3]/30
  precip_daily[120:151] = monthly_precip[4]/31
  precip_daily[151:181] = monthly_precip[5]/30
  precip_daily[181:212] = monthly_precip[6]/31
  precip_daily[212:243] = monthly_precip[7]/31
  precip_daily[243:273] = monthly_precip[8]/30
  precip_daily[273:304] = monthly_precip[9]/31
  precip_daily[304:334] = monthly_precip[10]/30
  precip_daily[334:365] = monthly_precip[11]/31
  return precip_daily

# assumption: annual_precip is a numpy array with 365 elements
def calculate_monthly_precip_from_daily(annual_precip):
    monthly_precip = np.zeros(12)
    monthly_precip[0] = np.sum(annual_precip[0:31])
    monthly_precip[1] = np.sum(annual_precip[31:59])
    monthly_precip[2] = np.sum(annual_precip[59:90])
    monthly_precip[3] = np.sum(annual_precip[90:120])
    monthly_precip[4] = np.sum(annual_precip[120:151])
    monthly_precip[5] = np.sum(annual_precip[151:181])
    monthly_precip[6] = np.sum(annual_precip[181:212])
    monthly_precip[7] = np.sum(annual_precip[212:243])
    monthly_precip[8] = np.sum(annual_precip[243:273])
    monthly_precip[9] = np.sum(annual_precip[273:304])
    monthly_precip[10] = np.sum(annual_precip[304:334])
    monthly_precip[11] = np.sum(annual_precip[334:365])
    return monthly_precip

# attempted spline for high and low averages.
# Use Cubic Spline function to fit the daily means to the monthly means;
```

```
# (This requires picking a day within each month for which we think the daily
mean will equal the monthly mean)
# Run the interpolation over 2 calendar years (730 days) so we get a smooth
transition from end of old year to beginning of new year
# Compute a new set of annual daily means using the last 40 days of old year,
first 40 days of new year, and averaging the rest of Old Year and New Year in
between.
# (We discard the first 40 days of Old Year and the last 40 days of New Year)

def compute_daily_temps_spline(interpol_x, monthly_avgs, timespan):
    interpol_y = np.zeros(np.size(interpol_x))
    interpol_y[0:12] = monthly_avgs
    interpol_y[12:24] = monthly_avgs
    interpol = CubicSpline(interpol_x, interpol_y)
    temps_new = interpol(timespan)

    temps_new_ = np.zeros(365)
    temps_new_[0:40] = temps_new[365:40+365]
    temps_new_[-40:] = temps_new[365-40:365]
    temps_new_[40:365-40] = (np.array(temps_new[40:365-40]) +
np.array(temps_new[365+40:730-40]))/2.0

    return temps_new_

# temps_mth = list of monthly average temps (array of size 12), temps_daily =
daily average temps assigned to its corresponding monthly average
# should return 'smoothed' daily average temps
def compute_daily_temps_runAvg(temps_daily, temps_mth, cycle_val):
  # attempt to implement the running average method from my MATLAB script
  temps_temp = np.zeros(730)
  # start out with the daily avgs set to the monthly avgs
  temps_temp[0:365] = temps_daily[0:365]
  temps_temp[365:730] = temps_daily[0:365]

  # run smoothing operation for n cycles (n = cycle_val parameter)
  for i in range(0, cycle_val):
    # for each cycle, re-compute average daily precip based on 4-day running
average across 2 calendar years
    for j in range(0, 730-4):
      temps_temp[j+2] = np.mean(temps_temp[j:j+4])

    # truncate edges of 2-year run
    temps_daily_new_runAvg = np.zeros(365)
    temps_daily_new_runAvg[0:10] = temps_temp[365:10+365]
    temps_daily_new_runAvg[-10:] = temps_temp[365-10:365]
    temps_daily_new_runAvg[10:365-10] = (np.array(temps_temp[10:365-10]) +
np.array(temps_temp[365+10:730-10]))/2.0

    # compute monthly totals from smoothed daily averages
    temps_daily_new_mths_rA =
calculate_mean_monthly_temps(temps_daily_new_runAvg)
    temps_diff_ = temps_daily_new_mths_rA - temps_mth

    # correct each daily average by how much the corresponding monthly
average is off from the starting data by
    # but only for each cycle where this will run next
    if i < cycle_val:
```

```
      temps_daily_new_runAvg[0:31] -= temps_diff_[0]
      temps_daily_new_runAvg[31:59] -= temps_diff_[1]
      temps_daily_new_runAvg[59:90] -= temps_diff_[2]
      temps_daily_new_runAvg[90:120] -= temps_diff_[3]
      temps_daily_new_runAvg[120:151] -= temps_diff_[4]
      temps_daily_new_runAvg[151:181] -= temps_diff_[5]
      temps_daily_new_runAvg[181:212] -= temps_diff_[6]
      temps_daily_new_runAvg[212:243] -= temps_diff_[7]
      temps_daily_new_runAvg[243:273] -= temps_diff_[8]
      temps_daily_new_runAvg[273:304] -= temps_diff_[9]
      temps_daily_new_runAvg[304:334] -= temps_diff_[10]
      temps_daily_new_runAvg[334:365] -= temps_diff_[11]

      # reassign the current daily precip averages to the temp array to be
smoothed in the next cycle
      temps_temp[0:365] = temps_daily_new_runAvg[0:365]
      temps_temp[365:730] = temps_daily_new_runAvg[0:365]


    # smoothing operation by re-assigning the normalized daily average to be
the 21-day mean of the unnormalized daily average
    temps_temp_ = np.zeros(730)
    for k in range(0+10, 730-10):
      temps_temp_[k] = np.mean(temps_temp[k-10:k+11])

    temps_temp_[0:10] = temps_temp[365:365+10]
    temps_temp_[730-10:] = temps_temp[365-10:365]
    temps_temp = temps_temp_

  return temps_temp

# https://www.omnicalculator.com/physics/relative-humidity#how-to-calculate-
relative-humidity
def calculate_wetbulb(drybulb, dp_temps):
  rel_humid = np.zeros(365)
  c1 = 17.625
  c2 = 243.04
  vals1 = c1*dp_temps[0:365] / (dp_temps[0:365] + c2)
  vals2 = c1*drybulb[0:365] / (drybulb[0:365] + c2)
  rel_humid = 100*np.exp(vals1)/np.exp(vals2)

  # https://www.omnicalculator.com/physics/wet-bulb#how-to-calculate-the-wet-
bulb-temperature
  '''
  Although many equations have been created over the years our calculator
uses the Stull formula,
  which is accurate for relative humidities between 5% and 99% and
temperatures between -20°C and 50°C.
  It loses its accuracy in situations where both moisture and heat are low in
value,
  but even then the error range is only between -1°C to +0.65°C.

  T _w = T*arctan(0.151977*sqrt(RH+8.313659)) +
0.00391838*sqrt(RH^3)*arctan(0.023101*RH) - arctan(RH-1.676331) +
arctan(T+RH) - 4.686035
  '''
  wetbulb = np.zeros(rel_humid.shape[0])
```

```
  # https://numpy.org/doc/stable/reference/generated/numpy.power.html
  c3 = np.power(rel_humid, 1.5)
  c4 = np.power(rel_humid+8.313659, 0.5)
  c5 = np.arctan(0.151977*c4)
  c6 = np.arctan(0.023101*rel_humid)
  c7 = np.arctan(rel_humid-1.676331)
  c8 = np.arctan(drybulb+rel_humid)
  wetbulb = drybulb*c5 + 0.00391838*c3*c6 - c7 + c8 - 4.686035
  return wetbulb

# assumption: we know monthly relative humidity but not monthly dew point
# the numpy arrays we're working with should all have 12 values
# https://www.omnicalculator.com/physics/relative-humidity#how-to-calculate-
relative-humidity
def calculate_dp_from_RH(drybulb, rel_humid):
  dp = np.zeros(12)
  c1 = 17.625
  c2 = 243.04
  vals1 = np.log(rel_humid/100)
  vals2 = np.divide(c1*drybulb[0:12], (drybulb[0:12] + c2))
  dp = np.divide(c2*(vals1 + vals2),(c1 - (vals1 + vals2)))
  return dp

# compute daily offsets based on monthly averages
# intended for use for precipitation data; computing offset
# curr_data_daily = numpy array of 365 values, daily averages
# offset_data_mth = numpy array of 12 values, based on average monthly totals
def subtract_offsets(curr_data_daily, offset_data_mth):
  curr_data_daily[0:31] -= offset_data_mth[0]/31
  curr_data_daily[31:59] -= offset_data_mth[1]/28
  curr_data_daily[59:90] -= offset_data_mth[2]/31
  curr_data_daily[90:120] -= offset_data_mth[3]/30
  curr_data_daily[120:151] -= offset_data_mth[4]/31
  curr_data_daily[151:181] -= offset_data_mth[5]/30
  curr_data_daily[181:212] -= offset_data_mth[6]/31
  curr_data_daily[212:243] -= offset_data_mth[7]/31
  curr_data_daily[243:273] -= offset_data_mth[8]/30
  curr_data_daily[273:304] -= offset_data_mth[9]/31
  curr_data_daily[304:334] -= offset_data_mth[10]/30
  curr_data_daily[334:365] -= offset_data_mth[11]/31
  return curr_data_daily

# 31-day average smoothing operation
# input is a numpy array with 365 values
# boolean represents whether this is for average precipitation across 31 days
(sum), or average probabilty for a given day (mean)
def smoothing_31dayavg(input, sumboolean):
  input_base = np.zeros(730)
  input_base[0:365] = input
  input_base[365:730] = input
  input_smooth = np.zeros(730)

  for i in range(0+15, 730-15):
    if (sumboolean):
      input_smooth[i] = np.sum(input_base[i-15:i+16])
    else:
      input_smooth[i] = np.mean(input_base[i-15:i+16])
```

```
  input_smooth[0:15] = input_smooth[365:365+15]
  input_smooth[730-15:] = input_smooth[365-15:365]
  return input_smooth[0:365]

# input monthly averages here from selected climate

dewpoint_mth = np.zeros(12)
rel_humid_mth = np.zeros(12)

highs_mth =
np.array([27.10,26.70,26.30,25.30,23.90,22.30,21.70,23.30,25.30,27.10,28.70,2
8.20])
lows_mth =
np.array([18.00,17.70,16.50,13.70,11.10,8.50,7.30,8.50,10.80,13.50,16.40,18.0
0])
avgs_mth =
np.array([22.55,22.20,21.40,19.50,17.50,15.40,14.50,15.90,18.05,20.30,22.55,2
3.10])
# adding the dewpoint/humidity stat
# default to dewpoint if both are available. if neither is available, can't
compute wet bulb temps
dewpoint_mth =
np.array([17.00,17.00,15.40,11.90,8.20,6.00,4.90,5.80,7.30,10.40,14.00,16.50]
)
#rel_humid_mth = np.array([71,72,69,62,54,53,52,51,50,53,59,66])

precip_mth = np.array([256,222,137,48,20,25,27,33,21,18,41,93])
# adding precip days
precip_days_mth =
np.array([17.7,18.1,12.6,7.2,4.3,3.9,4.2,5,3.4,2.3,4.3,7.6])
# adding 30-year record temps
rec_high_mth =
np.array([35.5,36.0,36.5,35.0,34.5,33.0,32.5,32.5,35.5,36.5,37.0,37.0])
rec_low_mth = np.array([11.0,10.0,5.5,1.5,0.5,0.0,0.5,0.0,0.5,0.5,6.5,9.5])

climate_name = "NouveauYathrib_2290"
# so print doesn't print in scientific mode
np.set_printoptions(suppress=True)

time = np.linspace(0, 730, 730)

# assumption: either dewpoint_mth is all zeros or relhumid_mth is all zeros,
but not both
# default: dewpoint_mth data is available making relhumid_mth redundant
# https://stackoverflow.com/questions/18395725/test-if-numpy-array-contains-
only-zeros
if (np.all(dewpoint_mth == 0)) and not (np.all(rel_humid_mth == 0)):
  print(rel_humid_mth)
  dewpoint_mth = np.round(calculate_dp_from_RH(avgs_mth, rel_humid_mth), 4)
  # Australia climate data uses afternoon RH, use highs_mth
  #dewpoint_mth = np.round(calculate_dp_from_RH(highs_mth, rel_humid_mth), 4)
  print(dewpoint_mth)


# https://stackoverflow.com/questions/48199077/elementwise-aggregation-
average-of-values-in-a-list-of-numpy-arrays-with-same
```

```
#avgs_mth = np.mean([highs_mth, lows_mth], axis=0)
#print(avgs_mth)

# use running_average method to compute daily mean temps monthly mean temps

# initialize the values of high_temps, low_temps, and avg_temps to the
monthly averages
# daily average for each day is the monthly average for that month
high_temps = assign_monthly_avg_to_days(highs_mth)
low_temps = assign_monthly_avg_to_days(lows_mth)
avg_temps = assign_monthly_avg_to_days(avgs_mth)

RH_temps = assign_monthly_avg_to_days(rec_high_mth)
RL_temps = assign_monthly_avg_to_days(rec_low_mth)

# tip for generating dew point estimates from rounded RH: if result above 18
= rounded to nearest 0.1, if result below 18 = rounded to nearest 0.5
dewpoint_temps = assign_monthly_avg_to_days(dewpoint_mth)
dp_temps_new = compute_daily_temps_runAvg(dewpoint_temps, dewpoint_mth, 50)
dp_mth_compute365 = calculate_mean_monthly_temps(dp_temps_new)

high_temps_new_ = compute_daily_temps_runAvg(high_temps, highs_mth, 50)

highs_mth_compute365 = calculate_mean_monthly_temps(high_temps_new_)
print(highs_mth_compute365 - highs_mth)
print(np.mean(high_temps_new_) - np.mean(high_temps))

low_temps_new_ = compute_daily_temps_runAvg(low_temps, lows_mth, 50)

lows_mth_compute365 = calculate_mean_monthly_temps(low_temps_new_)
print(lows_mth_compute365 - lows_mth)
print(np.mean(low_temps_new_) - np.mean(low_temps))

# compute new daily means from high and low temps
high_temps_new_ = 1.0*np.round(1*high_temps_new_, 2)
low_temps_new_ = 1.0*np.round(1*low_temps_new_, 2)
avg_temps_new_ = (np.array(low_temps_new_) + np.array(high_temps_new_))/2.0
#avg_temps_new_ = 1.0*np.round(1*avg_temps_new_, 2)
dp_temps_new_ = 1.0*np.round(1*dp_temps_new, 2)


wetbulb_high_dp = calculate_wetbulb(high_temps_new_[0:365], dp_temps_new_)
wetbulb_low_dp = calculate_wetbulb(low_temps_new_[0:365], dp_temps_new_)
wetbulb_avg_dp = calculate_wetbulb(avg_temps_new_[0:365], dp_temps_new_)


# https://datagy.io/matplotlib-title/
fig, ax = plt.subplots(figsize=(10,8))
#plt.figure(figsize = (10,8))
plt.title(climate_name + " Average Daily Temperatures" + "")
plt.xlabel("Day of year")
plt.ylabel("Temperature (deg C)")
plt.plot(time[0:365], high_temps_new_[0:365], 'g')
#plt.plot(time[0:365], high_temps, 'b')
plt.plot(time[0:365], low_temps_new_[0:365], 'g')
#plt.plot(time[0:365], low_temps, 'b')
plt.plot(time[0:365], avg_temps_new_[0:365], 'r')
```

```
#plt.plot(time[0:365], avg_temps, 'b')
#plt.plot(time[0:365], dewpoint_temps, 'b')
plt.plot(time[0:365], dp_temps_new_[0:365], 'y')
plt.plot(time[0:365], wetbulb_high_dp, 'purple')
#plt.plot(time[0:365], wetbulb_low_dp, 'purple')
plt.plot(time[0:365], RH_temps, 'b')
plt.plot(time[0:365], RL_temps, 'b')
plt.xlim(0, 365)
#plt.ylim(10, 50)
ax.set_xticks([0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365])
# remember to manually adjust the yticks range!
ax.set_yticks(np.arange(0, 55, 5))
ax.grid()
plt.show()

# trying to compute daily precip from monthly precip averages

tim = np.linspace(0, 730, 730)

# daily average for each day is the monthly average for that month
precip_daily = assign_initial_daily_precip(precip_mth)

precip_prob_daily = assign_initial_daily_precip(precip_days_mth)


p_time = np.linspace(0, 365, 365)

'''
https://weatherspark.com/y/137170/Average-Weather-in-Taipei-Taiwan-Year-
Round#Sections-Rain
To show variation within the months and not just the monthly totals, we show
the rainfall accumulated over a sliding 31-day period centered around each
day of the year.
use convolution to get the 31-day moving average after getting the daily
average for each day of the year
https://stackoverflow.com/questions/14313510/how-to-calculate-rolling-moving-
average-using-python-numpy-scipy
'''

# implement the running average method from my MATLAB script
precip_temp = np.zeros(730)
# start out with the daily avgs set to the monthly avgs
precip_temp[0:365] = precip_daily[0:365]
precip_temp[365:730] = precip_daily[0:365]

precip_p_temp = np.zeros(730)
# start out with the daily avgs set to the monthly avgs
precip_p_temp[0:365] = precip_prob_daily[0:365]
precip_p_temp[365:730] = precip_prob_daily[0:365]

# run smoothing operation for n cycles
cycle_val = 50
for i in range(0, cycle_val):
  # for each cycle, re-compute average daily precip based on 4-day running
average across 2 calendar years
  for j in range(0, 730-4):
    precip_temp[j+2] = np.mean(precip_temp[j:j+4])
```

```
    precip_p_temp[j+2] = np.mean(precip_p_temp[j:j+4])

  # truncate edges of 2-year run
  precip_daily_new_runAvg = np.zeros(365)
  precip_daily_new_runAvg[0:10] = precip_temp[365:10+365]
  precip_daily_new_runAvg[-10:] = precip_temp[365-10:365]
  precip_daily_new_runAvg[10:365-10] = (np.array(precip_temp[10:365-10]) +
np.array(precip_temp[365+10:730-10]))/2.0
  # compute monthly totals from smoothed daily averages
  precip_daily_new_mths_rA =
calculate_monthly_precip_from_daily(precip_daily_new_runAvg)
  precip_diff_ = precip_daily_new_mths_rA - precip_mth


  # truncate edges of 2-year run
  precip_p_daily_new_runAvg = np.zeros(365)
  precip_p_daily_new_runAvg[0:10] = precip_p_temp[365:10+365]
  precip_p_daily_new_runAvg[-10:] = precip_p_temp[365-10:365]
  precip_p_daily_new_runAvg[10:365-10] = (np.array(precip_p_temp[10:365-10])
+ np.array(precip_p_temp[365+10:730-10]))/2.0
  # compute monthly totals from smoothed daily averages
  precip_p_daily_new_mths_rA =
calculate_monthly_precip_from_daily(precip_p_daily_new_runAvg)
  precip_p_diff_ = precip_p_daily_new_mths_rA - precip_days_mth



  # correct each daily average by how much the corresponding monthly average
is off from the starting data by
  # but only for each cycle where this will run next
  if i < cycle_val:
    precip_daily_new_runAvg = subtract_offsets(precip_daily_new_runAvg,
precip_diff_)
    # reassign all negative values to equal 0; this is average precipitation
    precip_daily_new_runAvg[precip_daily_new_runAvg < 0] = 0
    # reassign the current daily precip averages to the temp array to be
smoothed in the next cycle
    precip_temp[0:365] = precip_daily_new_runAvg
    precip_temp[365:730] = precip_temp[0:365]

    precip_p_daily_new_runAvg = subtract_offsets(precip_p_daily_new_runAvg,
precip_p_diff_)
    # reassign all negative values to equal 0; probability should always be
between 0 and 1
    precip_p_daily_new_runAvg[precip_p_daily_new_runAvg < 0] = 0
    precip_p_daily_new_runAvg[precip_p_daily_new_runAvg > 1] = 1
    # reassign all indices where precip_daily_new_runAvg < 0 to equal 0
    precip_p_daily_new_runAvg[precip_daily_new_runAvg <= 0] = 0
    # reassign the current daily precip averages to the temp array to be
smoothed in the next cycle
    precip_p_temp[0:365] = precip_p_daily_new_runAvg
    precip_p_temp[365:730] = precip_p_temp[0:365]


precip_D_daily_new_mths_rA =
calculate_monthly_precip_from_daily(precip_daily_new_runAvg)
```

```
print("Monthly precip totals from smoothing function: \n",
np.round(precip_daily_new_mths_rA, decimals=2))
precip_diff_ = precip_daily_new_mths_rA - precip_mth
print("Diff from source data: \n", np.round(precip_diff_, decimals=2))


precip_p_D_daily_new_mths_rA =
calculate_monthly_precip_from_daily(precip_p_daily_new_runAvg)

print("Monthly precip days from smoothing function: \n",
np.round(precip_p_daily_new_mths_rA, decimals=2))
precip_p_diff_ = precip_p_daily_new_mths_rA - precip_days_mth
print("Diff from source data: \n", np.round(precip_p_diff_, decimals=2))

# implementing the 31-day running average sum method for each day of the year
for the climate/s precipitation
# assumption: precip_D_daily_new_runAvg already contains the current
simulated daily average precipitation
precip_daily_new_runAvg_sum = smoothing_31dayavg(precip_daily_new_runAvg,
True)
precip_p_daily_new_runAvg__ = smoothing_31dayavg(precip_p_daily_new_runAvg,
False)


running_avg_yearly_sum =
calculate_monthly_precip_from_daily(precip_daily_new_runAvg_sum/31)
print("Monthly precip totals from 31-day moving average: \n",
np.round(running_avg_yearly_sum, decimals=2))
print("Monthly precip totals (original): \n", precip_mth)
precip_diff_ = running_avg_yearly_sum - precip_mth
print("Diff from source data: \n", np.round(precip_diff_))

running_avg_yearly_sum_prob =
calculate_monthly_precip_from_daily(precip_p_daily_new_runAvg__)
print("Monthly precip days count from 31-day moving average: \n",
np.round(running_avg_yearly_sum_prob, decimals=2))
print("Monthly precip days count (original): \n", precip_days_mth)
precip_diff_p = running_avg_yearly_sum_prob - precip_days_mth
print("Diff from source data: \n", np.round(precip_diff_p,2))



#plt.figure(figsize = (10,8))
fig, ax = plt.subplots(1, figsize = (10,8))
plt.title(climate_name + " Average 31-day Floating Precipitation")
plt.xlabel("Day of year")
plt.ylabel("Precipitation (mm)")
plt.plot(tim[0:365], precip_daily_new_runAvg_sum[0:365], 'g')
#plt.plot(tim[0:365], np.multiply(31,precip_daily_new_runAvg[0:365]), 'r')
plt.ylim(0,)
plt.xlim(0, 365)
ax.set_xticks([0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365])
ax.grid()
plt.show()
```

```
# plot precip
#plt.figure(figsize = (10,8))
fig, ax = plt.subplots(1, figsize = (10,8))
plt.title(climate_name + " Average Daily Precipitation")
plt.xlabel("Day of year")
plt.ylabel("Precipitation (mm/day)")
plt.plot(tim[0:365], precip_daily, 'b')
plt.plot(tim[0:365], precip_daily_new_runAvg, 'r')
plt.plot(tim[0:365], precip_daily_new_runAvg_sum[0:365]/31, 'g')
ax.set_xticks([0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365])
plt.ylim(0,)
plt.xlim(0,365)
ax.grid()
plt.show()
# https://stackoverflow.com/questions/2891790/pretty-print-a-numpy-array-
without-scientific-notation-and-with-given-precision
#print(precip_daily_new_runAvg_sum[0:365])

# plot precip totals + probabilities
#plt.figure(figsize = (10,8))
# https://stackabuse.com/how-to-set-axis-range-xlim-ylim-in-matplotlib/
fig, ax = plt.subplots(1, figsize = (10,8))
plt.title(climate_name + " Average Daily Precipitation + Probability")
plt.xlabel("Day of year")
plt.ylabel("Precipitation (mm/day)")
plt.plot(tim[0:365], precip_daily, 'b')
plt.plot(tim[0:365], precip_daily_new_runAvg_sum[0:365]/31, 'g')
ax.set_xticks([0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365])
plt.ylim(0,)
plt.xlim(0,365)

ax2 = ax.twinx()
ax2.set_ylabel("Precipitation probability")
ax2.plot(tim[0:365], precip_prob_daily[0:365], 'r')
ax2.plot(tim[0:365], precip_p_daily_new_runAvg__, 'y')

ax.grid()
# need to manually set axis limits for each climate
ax.set_ylim(0,20.0)
ax2.set_ylim(0,1.0)
plt.xlim(0,365)
plt.show()


# plot probabilistic precipitation totals on days where it does rain
# to do this we divide mean daily precip by daily prob of precip
#plt.figure(figsize = (10,8))

# https://stackoverflow.com/questions/42540224/conditional-operations-on-
numpy-arrays
predict_precip_int = np.zeros(365)
predict_precip_int = np.where(precip_p_daily_new_runAvg__ > 0.01,
np.divide(precip_daily_new_runAvg_sum[0:365]/31,
precip_p_daily_new_runAvg__), predict_precip_int)
predict_precip_int = np.where(precip_p_daily_new_runAvg__ < 0.01,
100*precip_daily_new_runAvg_sum[0:365]/31, predict_precip_int)
```

```
# let's smooth the predicted precipitation intensity lmao
predict_precip_int_base = np.zeros(730)
predict_precip_int_base[0:365] = predict_precip_int
predict_precip_int_base[365:730] = predict_precip_int
predict_precip_int_smooth = np.zeros(730)
for i in range(0+2, 730-2):
  predict_precip_int_smooth[i] = np.mean(predict_precip_int_base[i-2:i+2])
predict_precip_int_smooth[0:2] = predict_precip_int_smooth[365:365+2]
predict_precip_int_smooth[730-2:] = predict_precip_int_smooth[365-2:365]


fig, ax = plt.subplots(1, figsize = (10,8))
plt.title(climate_name + " Predicted Precipitation Intensity")
plt.xlabel("Day of year")
plt.ylabel("Precipitation/day (mm)")
plt.plot(tim[0:365], predict_precip_int_smooth[0:365], 'g')
plt.ylim(0,)
plt.xlim(0, 365)
ax.set_xticks([0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365])
ax.grid()
plt.show()

from google.colab import files

#https://stackoverflow.com/questions/49394737/exporting-data-from-google-
colab-to-local-machine
#https://stackoverflow.com/questions/6081008/dump-a-numpy-array-into-a-csv-
file

# https://stackoverflow.com/questions/32635911/convert-elements-of-an-array-
from-scientific-notation-to-decimal-notation-in-pyt
# https://www.freecodecamp.org/news/dataframe-to-csv-how-to-save-pandas-
dataframes-by-exporting/
avg_temps = np.zeros((365, 7))
avg_temps[:,0] = high_temps_new_[0:365]
avg_temps[:,1] = avg_temps_new_[0:365]
avg_temps[:,2] = low_temps_new_[0:365]
avg_temps[:,3] = dp_temps_new_[0:365]
avg_temps[:,4] = 1.0*np.round(wetbulb_high_dp[0:365], 2)
avg_temps[:,5] = 1.0*np.round(precip_daily_new_runAvg_sum[0:365], 1)
avg_temps[:,6] = 100*np.round(precip_p_daily_new_runAvg__[0:365], 4)

np.set_printoptions(suppress=True, precision=2)
np.savetxt(climate_name + '_daily_averages_.csv', avg_temps,
delimiter=',',fmt='%f')
```