```python
# -*- coding: utf-8 -*-
"""
The module analyzes numina data.

@author: [Mike Lowry]

"""

__copyright__ = "Copyright 2023, Michael B. Lowry"
__license__ = "cc-by-nc-nd"
__email__ = "mlowry@uidaho.edu"
__credits__ = ["Mike Lowry", "Matt Thompson"]



from pathlib import Path
import time

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

import numina
import weatherdata as wd



start_time = time.time()
tc = time.time()

# ============================================================================
# Folder Paths
# ============================================================================

project_folder = Path(__file__).parents[2].__str__()
example_data_folder = project_folder +r'\ExampleData'
input_folder = example_data_folder + r'\Input'
output_folder = example_data_folder + r'\Output'


# ============================================================================
# Get Device information
# ============================================================================


df = numina.devices()
print(df)


df = numina.devices(serialno='NUM221P002S0030')
print(df)


df = numina.behaviorZones(serialnos=['NUM221P002S0024'])
print(df)

df = numina.behaviorZones()
print(df)


# ============================================================================
# Get Volume counts
# ============================================================================
startyear = 2022
startmonth = 11
startday = 16

endyear = 2022
endmonth = 12
```

```
endday = 16

serialnos = ["NUM221P002S0024", "NUM221P002S0030"]

startTime = (f"{startyear}-{startmonth}-{startday}T00:00:00")
endTime = (f"{endyear}-{endmonth}-{endday}T00:00:00")


interval = "1h"
objClasses = ["pedestrian", "car"]


# serialnos = ["NUM221P002S0024"]
# startTime = "2023-01-11T00:00:00"
# endTime = "2023-01-12T00:00:00"
# interval = "6h"
# objClasses = ["car"]


df = numina.feedCountMetrics(serialnos, startTime, endTime, interval, objClasses)
print(df)



df = df.loc[df.serialno =='NUM221P002S0030']
print(df)

df = df.loc[df.objClass =='pedestrian']
print(df)


zoneIds = [52850]
objClasses = ["bicycle"]
df = numina.zoneCountMetrics(zoneIds, startTime, endTime, interval, objClasses)
print(df)



# zoneIds = [52829]
# df = numina.zoneCountMetrics(zoneIds, startTime, endTime, interval, objClasses)
# print(df)




# ================================================================================
# Matt's Tests For Data Validation
# ================================================================================


# serialnos = ["NUM221P002S0024"]
# startTime = "2022-11-16T00:00:00"
# endTime = "2023-02-24T00:00:00"
# interval = "1d"
# objClasses = ["bicycle","pedestrian"]




# df = numina.feedCountMetrics(serialnos, startTime, endTime, interval, objClasses)
# df2 = df.loc[df.objClass =='pedestrian']
# df3 =df.loc[df.objClass =='bicycle']
# print(df)

# output_file = output_folder + r'\DataValidation216to223.xlsx'

# writer = pd.ExcelWriter(output_file)
# df.to_excel(writer,'Sheet1')
```

```
# writer.save()
# x =df3['time']
# y = df3['result']
# y2 = df2['result']

# # Creating a scatter plot.
# plt.figure(1)
# plt.plot(x, y, color='r', marker='+', linestyle=':')
# plt.plot(x, y2, color='g', marker='x', linestyle=':')
# plt.title("Bicycle Counts per Day")
# plt.xlabel("Day")
# plt.ylabel("Bicycle Counts")
# plt.legend(["Bicycles", "Pedestrians"], loc='upper left')
# plt.show()



# serialnos = ["NUM221P002S0024"]
# startTime = "2022-12-01T00:00:00"
# endTime = "2023-02-27T00:00:00"
# interval = "1m"
# objClasses = ["bicycle"]
# zoneId1 = [52850]
# zoneId2 = [52851]
# df = numina.zoneCountMetrics(zoneId1, startTime, endTime, interval, objClasses)
# df2 = numina.zoneCountMetrics(zoneId2, startTime, endTime, interval, objClasses)
# df2.columns= ('objClass', 'result','timedf2', 'zoneIds')

# print(df)

# print(df2)
# dfcombined = df[['time','result',]]
# dfcombined.append(df2[['result']])

# dfmerged = pd.concat([df, df2])

# ================================================================================
# Replicating Preliminary Analysis
# ================================================================================

# Input Parameters
startyear = 2022
startmonth = 12
startday = 1

endyear = 2023
endmonth = 2
endday = 7

if startday <= 9: # convert start day
    startdayurl = (f"0{startday}")
else:
    startdayurl =startday
if endday <= 9: # convert end day
    enddayurl = (f"0{endday}")
else:
    enddayurl = endday
if startmonth <= 9:
     startmonthurl = (f"0{startmonth}")
else:
     startmonthurl =startmonth
if endmonth <= 9:
     endmonthurl = (f"0{endmonth}")
else:
     endmonthurl = endmonth


serialnos = ["NUM221P002S0024"] # This is Camera West 1
```

```
startTime = (f"{startyear}-{startmonthurl}-{startdayurl}T00:00:00")
endTime = (f"{endyear}-{endmonthurl}-{enddayurl}T00:00:00")


interval = "1d"
objClasses = ["car", "bicycle"]
zoneId1 = [52853] # This zone ID is the Bike Lane Zone
zoneId2 = [52850] # This zone ID is the Street Zone

# Collecting Data

# This collects the data for the Street Zone
df = numina.zoneCountMetrics(zoneId1, startTime, endTime, interval, objClasses)

dfbike_street =df.loc[df.objClass =='bicycle']
dfcar_street =df.loc[df.objClass =='car']



# This collects the data for the Bike Lane Zone
df2 = numina.zoneCountMetrics(zoneId2, startTime, endTime, interval, objClasses)

dfbike_bikelane =df2.loc[df2.objClass =='bicycle']
dfcar_bikelane =df2.loc[df2.objClass =='car']

#This resets the indices of the count dataframes
dfbike_street.reset_index(drop=True, inplace=True)
dfbike_bikelane.reset_index(drop=True, inplace=True)
dfcar_bikelane.reset_index(drop=True, inplace=True)
dfcar_street.reset_index(drop=True, inplace=True)


# Analyzing Data
dfanalysis = pd.DataFrame(columns=['Date','delta', 'SNOW', 'PRECIP', 'snow_ind', 'precip_ind'])
dfanalysis['delta'] = dfbike_bikelane['result'] - dfbike_street['result']

#This identifies which zone has a higher count
dfanalysis.loc[dfanalysis['delta'] > 0, 'which_is_greater?'] = 'Bikelane'
dfanalysis.loc[dfanalysis['delta'] <= 0, 'which_is_greater?'] = 'Street'

#This uses the weather package to pull weather data for the desired range
dfweather = wd.weather(startyear, startmonth, startday, endyear, endmonth, endday)

print(dfweather)

#This transfers weather data to the analysis dataframe
dfanalysis['Date'] = dfweather['Date']
dfanalysis['SNOW'] = dfweather['SNOW']
dfanalysis['PRECIP'] = dfweather['PRECIP']
dfanalysis['snow_ind'] = dfweather['snow_ind']
dfanalysis['cold_ind'] = dfweather['cold_ind']
dfanalysis['precip_ind'] = dfweather['precip_ind']
dfanalysis['bikecount_bikelane'] = dfbike_bikelane['result']
dfanalysis['bikecount_street'] = dfbike_street['result']


dfanalysis['snow_flag'] = dfanalysis.apply(wd.flagsnow, axis = 1)


dfanalysis['precip_flag'] = dfanalysis.apply(wd.flagprecip, axis = 1)
dfanalysis['nonprecip_flag'] = dfanalysis.apply(wd.flagnonprecip, axis = 1)

dfanalysis['snowcountstreet'] = dfanalysis.apply(wd.countsnowstreet, axis = 1)
dfanalysis['snowcountbikelane'] = dfanalysis.apply(wd.countsnowbikelane, axis = 1)

dfanalysis['snowcountstreetcold'] = dfanalysis.apply(wd.countsnowstreetcold, axis = 1)
dfanalysis['snowcountbikelanecold'] = dfanalysis.apply(wd.countsnowbikelanecold, axis = 1)

dfanalysis['precipcountstreet'] = dfanalysis.apply(wd.countprecipstreet, axis = 1)
```

```
dfanalysis['precipcountbikelane'] = dfanalysis.apply(wd.countprecipbikelane, axis = 1)

dfanalysis['normalcountbikelane'] = dfanalysis.apply(wd.countnormalstreet, axis = 1)
dfanalysis['normalcountstreet'] = dfanalysis.apply(wd.countnormalbikelane, axis = 1)

# ============================================================================
# Secondary Analysis- Proportion of Counts
# ============================================================================

plt.rcParams.update({'font.size': 14})

# Proportion all days
total_bikelane = dfanalysis['bikecount_bikelane'].sum()
total_street = dfanalysis['bikecount_street'].sum()
print('The total count for the bike lane on all days is', total_bikelane)
print('The total count for the street on all days is', total_street)

count_proportion_all = total_bikelane / (total_street +total_bikelane)
print(round(count_proportion_all, 4))

categories = ["Bikelane", "Street"]
x = [total_bikelane, total_street]
plotlabellist = []
for plotlabel in x:
    plotlabels = (f"{plotlabel} cyclists")
    plotlabellist.append(plotlabels)
plt.figure(1)

plt.pie(x, labels=plotlabellist, autopct='%0.0f%%', colors = ['cornflowerblue', 'green'], wedgeprops = {'edgecolor' : 'b
plt.axis("equal")
plt.legend(categories, loc = 'center right', bbox_to_anchor = (1,0))
plt.title("All Days")
plt.show()

dfanalysis['biketotal'] = dfanalysis['bikecount_bikelane'] + dfanalysis['bikecount_street']
bikeaverage = dfanalysis['biketotal'].mean()
print("The average bike count is ", round(bikeaverage,2))
#Proportion non precip days
total_bikelane_normal = dfanalysis['normalcountbikelane'].sum()
total_street_normal = dfanalysis['normalcountstreet'].sum()
print('The total count for the bike lane on nonprecip days is', total_bikelane_normal)
print('The total count for the street on nonprecip days is', total_street_normal)

count_proportion_normal = total_bikelane_normal / (total_street_normal +total_bikelane_normal)
print(round(count_proportion_normal, 4))

x = [total_bikelane_normal, total_street_normal]
plotlabellist = []
for plotlabel in x:
    plotlabels = (f"{plotlabel} cyclists")
    plotlabellist.append(plotlabels)
plt.figure(2)

plt.pie(x, labels=plotlabellist, autopct='%0.0f%%', colors = ['cornflowerblue', 'green'], wedgeprops = {'edgecolor' : 'b
plt.axis("equal")
plt.legend(categories, loc = 'center right', bbox_to_anchor = (1,0))
plt.title("Non Precipitation Days")
plt.show()

#Proportion Snow days
total_snow_bikelane = dfanalysis['snowcountbikelane'].sum()
total_snow_street = dfanalysis['snowcountstreet'].sum()
print('The total count for the bike lane on snowy days is', total_snow_bikelane)
print('The total count for the street on snowy days is', total_snow_street)

count_proportion_snow = total_snow_bikelane / (total_snow_street + total_snow_bikelane)
print(round(count_proportion_snow, 4))

x = [total_snow_bikelane, total_snow_street]
```

```
plotlabellist = []
for plotlabel in x:
    plotlabels = (f"{plotlabel} cyclists")
    plotlabellist.append(plotlabels)
plt.figure(3)

plt.pie(x, labels=plotlabellist, autopct='%0.0f%%', colors = ['cornflowerblue', 'green'], wedgeprops = {'edgecolor' : 'b
plt.axis("equal")
plt.legend(categories, loc = 'center right', bbox_to_anchor = (1,0))
plt.title("Snowy Days")
plt.show()

# Proportion cold snowy days
total_snow_bikelane_cold = dfanalysis['snowcountbikelanecold'].sum()
total_snow_street_cold = dfanalysis['snowcountstreetcold'].sum()
print('The total count for the bike lane on cold snowy days is', total_snow_bikelane_cold)
print('The total count for the street on cold snowy days is', total_snow_street_cold)

count_proportion_snow_cold = total_snow_bikelane_cold / (total_snow_street_cold + total_snow_bikelane_cold)
print(round(count_proportion_snow_cold, 4))

x = [total_snow_bikelane_cold, total_snow_street_cold]
plotlabellist = []
for plotlabel in x:
    plotlabels = (f"{plotlabel} cyclists")
    plotlabellist.append(plotlabels)
plt.figure(4)

plt.pie(x, labels=plotlabellist, autopct='%0.0f%%', colors = ['cornflowerblue', 'green'], wedgeprops = {'edgecolor' : 'b
plt.axis("equal")
plt.legend(categories, loc = 'center right', bbox_to_anchor = (1,0))
plt.title("Cold Snowy Days")
plt.show()

# Proportion Precip days
total_precip_bikelane = dfanalysis['precipcountbikelane'].sum()
total_precip_street = dfanalysis['precipcountstreet'].sum()
print('The total count for the bike lane on precipitation days is', total_precip_bikelane)
print('The total count for the street on precipitation days is', total_precip_street)

count_proportion_precip = total_precip_bikelane / (total_precip_street + total_precip_bikelane)
print(round(count_proportion_precip, 4))

x = [total_precip_bikelane, total_precip_street]
plotlabellist = []
for plotlabel in x:
    plotlabels = (f"{plotlabel} cyclists")
    plotlabellist.append(plotlabels)
plt.figure(5)

plt.pie(x, labels = plotlabellist , autopct='%0.0f%%', colors = ['cornflowerblue', 'green'], wedgeprops = {'edgecolor' :
plt.axis("equal")
plt.legend(categories, loc = 'center right', bbox_to_anchor = (1,0))
plt.title("Precipitation Days")
plt.show()

# ============================================================================
# Original Analysis- Proportions of days
# ============================================================================
# Returns the Proportion for all days
CountBikeLaneGreater = (dfanalysis['which_is_greater?'] == 'Bikelane').sum(axis=0)
print("The number of days which have more bikes in the bike lane are", CountBikeLaneGreater)

CountStreetGreater = (dfanalysis['which_is_greater?'] == 'Street').sum(axis=0)
print("The number of days which have more bikes in the street are", CountStreetGreater)

# Returns the Proportion for Non Precipitation Days
non_precip_count_bike_lane_greater = (dfanalysis['nonprecip_flag'] == 'Bikelane').sum(axis=0)
print("The number of days without precipitation which have more bikes in the bike lane are",non_precip_count_bike_lane_g
```

```
non_precip_count_street_greater = (dfanalysis['nonprecip_flag'] == 'Street').sum(axis=0)
print("The number of days without precipitation which have more bikes in the street are",non_precip_count_street_greater

# Returns the Proportion for Precipitation Days
PrecipCountBikeLaneGreater = (dfanalysis['precip_flag'] == 'Bikelane').sum(axis=0)
print("The number of days with precipitation which have more bikes in the bike lane are", PrecipCountBikeLaneGreater)

PrecipCountStreetGreater = (dfanalysis['precip_flag'] == 'Street').sum(axis=0)
print("The number of days with precipitation which have more bikes in the street are", PrecipCountStreetGreater)

# Returns the Proportion for Snow Days
SnowCountBikeLaneGreater = (dfanalysis['snow_flag'] == 'Bikelane').sum(axis=0)
print("The number of days with snow which have more bikes in the bike lane are", SnowCountBikeLaneGreater)

SnowCountStreetGreater = (dfanalysis['snow_flag'] == 'Street').sum(axis=0)
print("The number of days with precipitation which have more bikes in the street are", SnowCountStreetGreater)

proportion_all = CountStreetGreater / (CountStreetGreater + CountBikeLaneGreater)
proportion_nonprecip = non_precip_count_street_greater / (non_precip_count_street_greater + non_precip_count_bike_lane_g
proportion_precipitation = PrecipCountStreetGreater / (PrecipCountStreetGreater + PrecipCountBikeLaneGreater)
proportion_snow = SnowCountStreetGreater / (SnowCountStreetGreater + SnowCountBikeLaneGreater)

print("The proportion for all days is", round(proportion_all, 2))
print("The proportion for non-precipitation days is", round(proportion_nonprecip, 2))
print("The proportion for precipitation days is", round(proportion_precipitation, 2))
print("The proportion for snowy days is", round(proportion_snow, 2))


print("\n", "Finished \n", time.time() - start_time, "seconds")
```