CSV Reader

July 11, 2021

1 Custom CSV Reader

1.1 Part One

Import the custom CSV Reader and then place the file path in the function: ReadCSV.read_csv(filepath), there is an option to set header = False but by default this is set to True.

[1]:
```
import ReadCSV
```

Load all four of the CSV files to test the reader.

[2]:
```
barometer = ReadCSV.read_csv('weather-data/barometer-1617.csv')
indoor_temp = ReadCSV.read_csv('weather-data/indoor-temperature-1617.csv')
outdoor_temp = ReadCSV.read_csv('weather-data/outside-temperature-1617.csv')
rainfall = ReadCSV.read_csv('weather-data/rainfall-1617.csv')
```

```
356 lines read out of 356. 0 failed
355 lines read out of 355. 0 failed
356 lines read out of 356. 0 failed
354 lines read out of 354. 0 failed
```

1.2 Part Two

1.2.1 Convert the list of dicitonaries to data frame format to allow for statistical calulations

[3]:
```
#Only imported for part two
import pandas as pd
```

1.2.2 Summary Statistics for Barometer Data

[4]:
```
df_barometer = pd.DataFrame(barometer)
df_barometer['Baro'] = pd.to_numeric(df_barometer["Baro"], errors='coerce')
df_barometer['DateTime'] = pd.to_datetime(df_barometer["DateTime"],␣
 ↪errors='coerce')
df_barometer.dtypes
```

[4]:
```
DateTime    datetime64[ns]
Baro               float64
dtype: object
```

[5]:
```
df_barometer.head()
```

[5]:
```
    DateTime    Baro
0 2016-10-09  1021.9
1 2016-10-10  1019.9
2 2016-10-11  1015.8
3 2016-10-12  1013.2
4 2016-10-13  1005.9
```

[6]:
```
df_barometer.describe()
```

[6]:
```
              Baro
count   355.000000
mean   1009.998873
std       9.869662
min     979.600000
25%    1004.850000
50%    1010.500000
75%    1016.050000
max    1035.600000
```

1.2.3 Summary Statistics for Rainfall Data

[7]:
```
df_rainfall = pd.DataFrame(rainfall)
df_rainfall['mm'] = pd.to_numeric(df_rainfall["mm"], errors='coerce')
df_rainfall['DateTime'] = pd.to_datetime(df_rainfall["DateTime"],␣
 ↪errors='coerce')
df_rainfall.describe()
```

[7]:
```
               mm
count  353.000000
mean     1.548725
std      3.324599
min      0.000000
25%      0.000000
50%      0.000000
75%      1.100000
max     23.200000
```

[8]:
```
df_rainfall.tail()
```

[8]:
```
      DateTime   mm
348 2017-10-05  1.0
349 2017-10-06  0.0
350 2017-10-07  1.1
351 2017-10-08  0.0
352 2017-10-09  0.0
```

1.2.4 Summary Statistics for Both Indoor and Outdoor Temprature Data

[9]:
```
df_indoor = pd.DataFrame(indoor_temp)
df_outdoor = pd.DataFrame(outdoor_temp)
```

1.2.5 Outdoor Data

[10]:
```
df_outdoor.set_index(df_outdoor.DateTime, drop=True, inplace = True)
df_outdoor.drop(columns =  'DateTime')
df_outdoor['Temperature'] = pd.to_numeric(df_outdoor["Temperature"],␣
 ↪errors='coerce')
df_outdoor['Temperature_range (low)'] = pd.
 ↪to_numeric(df_outdoor["Temperature_range (low)"], errors='coerce')
df_outdoor['Temperature_range (high)'] = pd.
 ↪to_numeric(df_outdoor["Temperature_range (high)"], errors='coerce')
df_outdoor.describe()
```

[10]:

	Temperature	Temperature_range (low)	Temperature_range (high)
count	355.000000	355.000000	355.000000
mean	11.138877	7.865634	15.524225
std	5.355042	4.878930	7.034445
min	-1.810000	-4.100000	1.500000
25%	7.390000	4.350000	10.250000
50%	10.960000	8.000000	15.100000
75%	15.050000	12.050000	19.850000
max	26.380000	18.700000	38.500000

1.2.6 Indoor Data

[11]:
```
df_indoor.set_index(df_indoor.DateTime, drop=True, inplace = True)
df_indoor.drop(columns =  'DateTime')
df_indoor['Humidity'] = pd.to_numeric(df_indoor["Humidity"], errors='coerce')
df_indoor['Temperature'] = pd.to_numeric(df_indoor["Temperature"],␣
 ↪errors='coerce')
df_indoor['Temperature_range (low)'] = pd.
 ↪to_numeric(df_indoor["Temperature_range (low)"], errors='coerce')
df_indoor['Temperature_range (high)'] = pd.
 ↪to_numeric(df_indoor["Temperature_range (high)"], errors='coerce')
df_indoor.describe()
```

[11]:

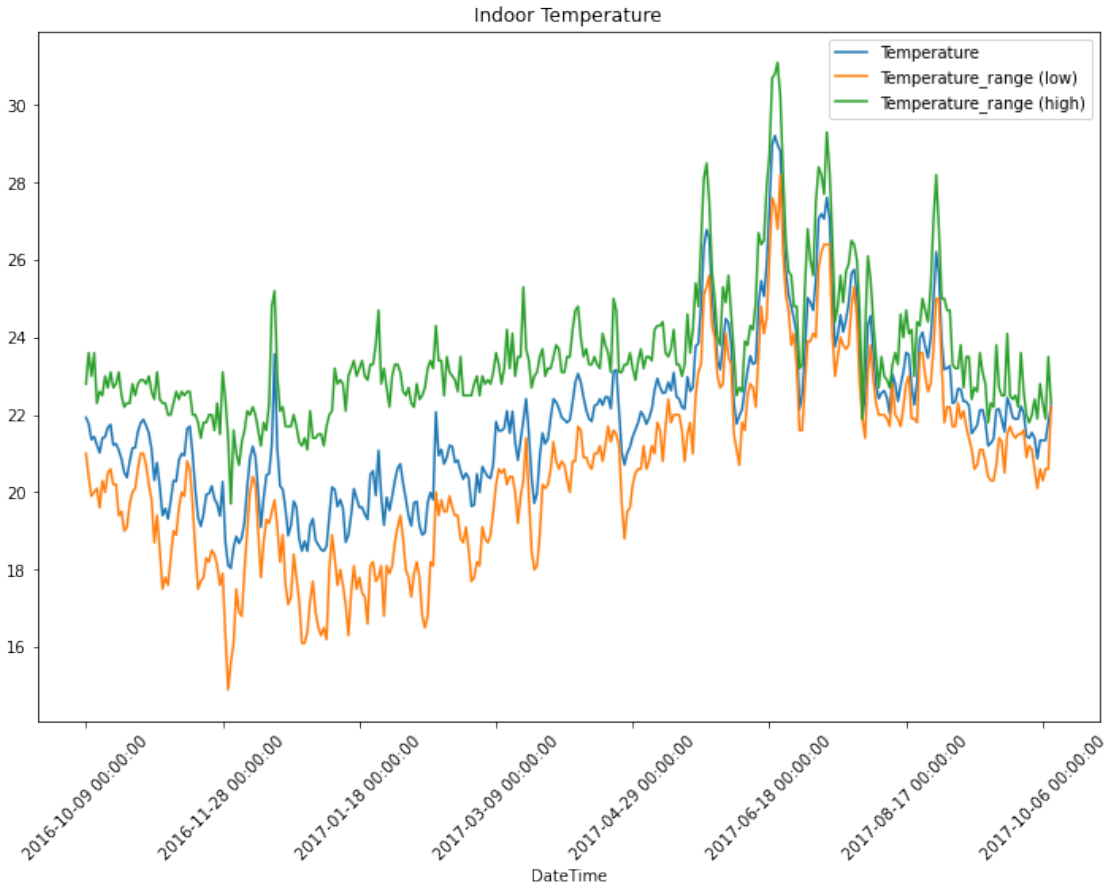
	Humidity	Temperature	Temperature_range (low)	\
count	354.000000	354.000000	354.000000	
mean	48.519774	21.827885	20.555932	
std	5.188886	2.058307	2.405125	
min	37.000000	18.040000	14.900000	
25%	44.000000	20.345000	18.725000	

```
50%      48.000000    21.710000                  20.600000
75%      52.000000    22.710000                  21.900000
max      59.000000    29.210000                  28.200000

       Temperature_range (high)
count                354.000000
mean                  23.533616
std                    1.701466
min                   19.700000
25%                   22.500000
50%                   23.200000
75%                   24.100000
max                   31.100000
```

Plot of the indoor data to show what the correct data should look like. This plot is compared later on with an incorrect CSV file to highlight the errors in the ammended file.

[12]:
```
plt = df_indoor.plot(x="DateTime", y=["Temperature", "Temperature_range (low)",␣
↪'Temperature_range (high)'],\
                     kind="line", rot = 45, title = 'Indoor Temperature',␣
↪figsize = (12,8))
```



2 Error Data

The errors in this ammended files are as follows:

- Use the mid temp above and below the range to show in plot and summary statistics.
- Miss a line so it is seen in the count.
- Set a zero so it is seen in the min and also the plot.

[13]:
```
error_data = ReadCSV.read_csv('weather-data/indoor-temperature-1617-errors.csv')
```

```
355 lines read out of 355. 0 failed
```

In the output below it shows how the missing line with a date of 2016-10-16 is now skipped over. While NaN or some other type could have been inserted this was not considered necessary in this case since the user is made aware of the total number of lines parsed. This can then be compared with the number of items in the list.

[14]:
```
error_data[6:8]
```

[14]:
```
[{'DateTime': '2016-10-15 00:00:00',
  'Humidity': '53',
  'Temperature': '21.4',
  'Temperature_range (low)': '20.3',
  'Temperature_range (high)': '22.5'},
 {'DateTime': '2016-10-17 00:00:00',
  'Humidity': '53',
  'Temperature': '21.67',
  'Temperature_range (low)': '20.5',
  'Temperature_range (high)': '22.7'}]
```

[15]:
```
df_errors = pd.DataFrame(error_data)
df_errors.set_index(df_errors.DateTime, drop=True, inplace = True)
df_errors.drop(columns =  'DateTime')
df_errors['Humidity'] = pd.to_numeric(df_errors["Humidity"], errors='coerce')
df_errors['Temperature'] = pd.to_numeric(df_errors["Temperature"],␣
 ↪errors='coerce')
df_errors['Temperature_range (low)'] = pd.
 ↪to_numeric(df_errors["Temperature_range (low)"], errors='coerce')
df_errors['Temperature_range (high)'] = pd.
 ↪to_numeric(df_errors["Temperature_range (high)"], errors='coerce')
df_errors.describe()
```

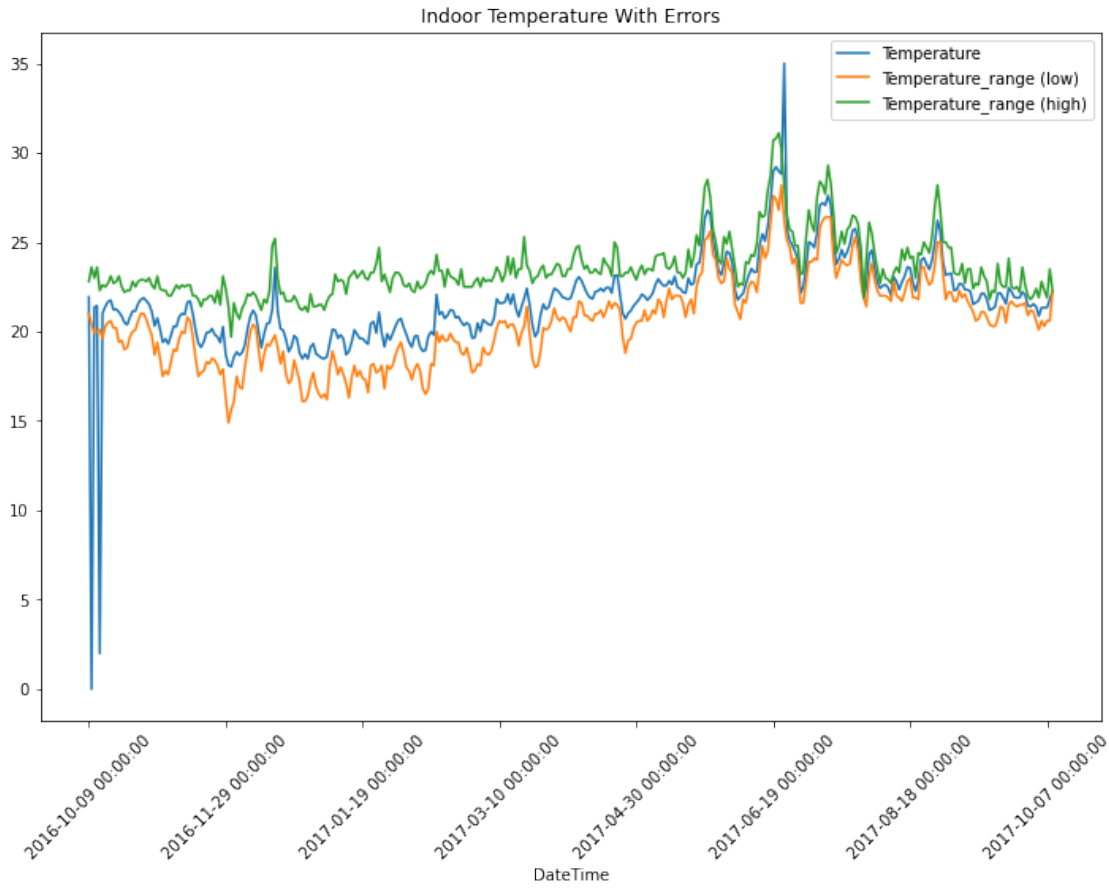
[15]:
```
          Humidity  Temperature  Temperature_range (low)  \
count  353.000000   353.000000               353.000000
mean    48.507082    21.736179                20.557507
```

```
std      5.190746     2.670660                 2.408356
min     37.000000     0.000000                14.900000
25%     44.000000    20.300000                18.700000
50%     48.000000    21.700000                20.600000
75%     52.000000    22.710000                21.900000
max     59.000000    35.000000                28.200000

       Temperature_range (high)
count                353.000000
mean                  23.535127
std                    1.703643
min                   19.700000
25%                   22.500000
50%                   23.200000
75%                   24.100000
max                   31.100000
```

- A total of 355 lines read but it has a count of 353.
- Min temperature is 0 not within the ranges of the low and high.
- Same for the Max temprature.
- These errors are evident from the plot below where the blue line now crosses the boundaries of the low and high clearly illustrating the errors in the data.

[16]:
```
plt = df_errors.plot(x="DateTime", y=["Temperature", "Temperature_range (low)",␣
↪'Temperature_range (high)'],\
                     kind="line", rot = 45, title = 'Indoor Temperature With␣
↪Errors', figsize = (12,8))
```



2.1 Conclusion

The advantage of this reader over a traditional CSV reader is that it allows for a more custamisable appraoch. For exmaple the ingestion of the data allows for the traditional CSV structure to be preserverd in a fashion that allows the creater to personalise to their specifications. Whilst it is slower than the existing packages speed is not neccesarily always a factor espcially with smaller files and less is known about the data structure and format of the file being ingested.

[ ]: