PDF–Data_Cleaning_Report

September 1, 2021

0.1 Imports

We'll import the *Pandas* and *mysq.connector* packages to import database data into a Pandas dataframe which we'll name **raw_temperature_df** (raw temperature dataframe). We'll also import a dictionary containing the login information to the MySQL server as **CREDS**.

Note that we'll leave **id** out of our SQL *SELECT* statement since Pandas provides automatic row indexing. These new indexes will be equal to the orignal **id** of the data minus 1.

[1]:
```
import pandas as pd
import mysql.connector as connector
from database_credentials import MySQL_credentials as CREDS

connection = connector.connect(
    host = CREDS['host'],
    user = CREDS['user'],
    password = CREDS['password'],
    database = CREDS['database']
)

raw_temperature_df = pd.read_sql(f"SELECT inside_temperature,␣
 ↪outside_temperature, time, date FROM {CREDS['table']}", con=connection)
```

0.2 Data Types

Next, we'll check the shape of the dataframe to confirm we imported all of the rows. We should have, at the time of writing, 348 rows. This is confirmed by accessing the *shape* attribute.

[2]:
```
raw_temperature_df.shape
```

[2]:
```
(347, 4)
```

We'll check that our columns (attributes) are of the proper data type by accessing the *dtypes* attribute of the dataframe. In this case, the **date** column was incorrectly typed as an *object* (string) since Pandas doesn't support the *date* type that the **date** column was stored as in the database.

Note that Pandas also doesn't support the *time* type that the **time** column is stored as so it has been converted to a *timedelta*. This is fine for the purposes of our analysis.

[3]:
```
raw_temperature_df.dtypes
```

[3]:
```
inside_temperature               int64
outside_temperature              int64
time                   timedelta64[ns]
date                            object
dtype: object
```

To convert the **date** column to a *datetime* we'll use the *to_datetime* function that Pandas offers.

Another route to take would be re-querying the data, *CAST*ing the **date** column as *datetime* but that would be less computation efficient and less time efficient. This would also likely muddle the clarity of the data cleaning process.

Secondly, we'll create extra columns containing the year, month, and day of the month for each observation.

[4]:
```
raw_temperature_df['date'] = pd.to_datetime(raw_temperature_df['date'])  #␣
 ↪convert date column type from object to datetime

raw_temperature_df['year'] = raw_temperature_df['date'].map(lambda x: x.year)
raw_temperature_df['month'] = raw_temperature_df['date'].map(lambda x: x.month)
raw_temperature_df['day'] = raw_temperature_df['date'].map(lambda x: x.day)
```

A final check of the data types in each column reveals exactly the desired outcome. The **year**, **month**, and **day** columns are as integers but this is fine for our purposes.

[5]:
```
raw_temperature_df.dtypes
```

[5]:
```
inside_temperature               int64
outside_temperature              int64
time                   timedelta64[ns]
date                    datetime64[ns]
year                             int64
month                            int64
day                              int64
dtype: object
```

While data is only uploaded to the database if there are no null values in the observation, we should remove rows in the dataframe containing NaN values, the Pandas equivalent for null values, to be confident we won't raise any arithmetic exceptions during the analysis phase.

[6]:
```
raw_temperature_df.dropna(axis='index')
```

[6]:

	inside_temperature	outside_temperature	time	date	year	\
0	67	62	0 days 00:00:00	2021-06-25	2021	
1	66	64	0 days 04:00:00	2021-06-25	2021	
2	69	64	0 days 08:00:00	2021-06-25	2021	
3	72	78	0 days 12:00:00	2021-06-25	2021	
4	70	76	0 days 16:00:00	2021-06-25	2021	

```
..                  …                  …              …        …    …
342                  70                   72 0 days 04:00:00 2021-08-31  2021
343                  71                   73 0 days 08:00:00 2021-08-31  2021
344                  74                   86 0 days 12:00:00 2021-08-31  2021
345                  74                   84 0 days 16:00:00 2021-08-31  2021
346                  78                   78 0 days 20:00:00 2021-08-31  2021

     month  day
0        6   25
1        6   25
2        6   25
3        6   25
4        6   25
..     …  …
342      8   31
343      8   31
344      8   31
345      8   31
346      8   31

[347 rows x 7 columns]
```

0.3 Consistency of Data and Duplicates

We should make sure that the amount of data in each group that we'll be analyzing is consistent so as not to introduce bias into our analysis. This isn't a necessary prerequisite if we wish to perform an ANOVA test, so long as the variance between the groups is similar. [^1] For this reason we'll also check the variance of the groups by looking at the standard deviation for each. Let's first start with the **time** groups.

[7]:
```
raw_temperature_df.groupby(['time']).describe().loc[:, ['inside_temperature',␣
↪'outside_temperature']]
```

[7]:
```
                 inside_temperature                                          \
                             count       mean       std   min    25%   50%
time
0 days 00:00:00               60.0  71.900000  2.790419  66.0  70.00  72.0
0 days 00:00:04                3.0  70.333333  2.081666  68.0  69.50  71.0
0 days 00:00:08                3.0  72.333333  0.577350  72.0  72.00  72.0
0 days 00:00:12                2.0  75.500000  0.707107  75.0  75.25  75.5
0 days 00:00:16                3.0  73.666667  1.527525  72.0  73.00  74.0
0 days 00:00:20                4.0  75.250000  2.500000  72.0  74.25  75.5
0 days 04:00:00               52.0  70.346154  1.866995  66.0  69.00  70.0
0 days 04:01:00                1.0  79.000000       NaN  79.0  79.00  79.0
0 days 08:00:00               53.0  70.830189  2.190327  66.0  69.00  71.0
0 days 08:01:00                1.0  70.000000       NaN  70.0  70.00  70.0
```

0 days 12:00:00	56.0	73.553571	2.682737	68.0	72.00	74.0	
0 days 13:19:00	1.0	72.000000	NaN	72.0	72.00	72.0	
0 days 16:00:00	53.0	73.981132	3.091493	67.0	72.00	74.0	
0 days 20:00:00	54.0	73.333333	2.555054	67.0	72.00	74.0	
0 days 20:01:00	1.0	81.000000	NaN	81.0	81.00	81.0	

	outside_temperature						\
	75%	max	count	mean	std	min	
time							
0 days 00:00:00	74.00	80.0	60.0	72.483333	4.575142	62.0	
0 days 00:00:04	71.50	72.0	3.0	72.666667	1.527525	71.0	
0 days 00:00:08	72.50	73.0	3.0	74.000000	2.645751	72.0	
0 days 00:00:12	75.75	76.0	2.0	88.500000	2.121320	87.0	
0 days 00:00:16	74.50	75.0	3.0	91.333333	3.785939	87.0	
0 days 00:00:20	76.50	78.0	4.0	83.250000	3.685557	79.0	
0 days 04:00:00	72.00	75.0	52.0	70.634615	4.401554	62.0	
0 days 04:01:00	79.00	79.0	1.0	72.000000	NaN	72.0	
0 days 08:00:00	72.00	76.0	53.0	72.037736	5.045799	61.0	
0 days 08:01:00	70.00	70.0	1.0	68.000000	NaN	68.0	
0 days 12:00:00	75.00	79.0	56.0	83.267857	6.934738	67.0	
0 days 13:19:00	72.00	72.0	1.0	77.000000	NaN	77.0	
0 days 16:00:00	76.00	81.0	53.0	82.207547	6.805989	67.0	
0 days 20:00:00	75.00	79.0	54.0	77.648148	5.508681	65.0	
0 days 20:01:00	81.00	81.0	1.0	74.000000	NaN	74.0	

	25%	50%	75%	max
time				
0 days 00:00:00	69.75	73.0	75.25	82.0
0 days 00:00:04	72.00	73.0	73.50	74.0
0 days 00:00:08	72.50	73.0	75.00	77.0
0 days 00:00:12	87.75	88.5	89.25	90.0
0 days 00:00:16	90.00	93.0	93.50	94.0
0 days 00:00:20	82.00	83.0	84.25	88.0
0 days 04:00:00	68.00	72.0	73.25	80.0
0 days 04:01:00	72.00	72.0	72.00	72.0
0 days 08:00:00	69.00	72.0	75.00	85.0
0 days 08:01:00	68.00	68.0	68.00	68.0
0 days 12:00:00	78.00	83.0	89.25	95.0
0 days 13:19:00	77.00	77.0	77.00	77.0
0 days 16:00:00	78.00	81.0	87.00	97.0
0 days 20:00:00	74.00	76.5	81.00	91.0
0 days 20:01:00	74.00	74.0	74.00	74.0

The first thing to notice is that, there are some observations that were recorded at odd times. This could be caused due to a power outage delaying the running of the data collection script or an error in uploading to the database. We can allow for a margin of error of a minute, since temperature changes are typically negligible on the timescales of seconds. Upon further examination, the

discrepancy between the count of observations at midnight (00:00:00) and the other times seems a bit conspicuous, especially if we were to assign the times between 00:00:04 and 00:00:20 to also be midnight observations.

A more meticulous manual look at the data shows an interesting pattern between indices 64 and 80 and it becomes clear what occurred. From the latter half of 2021-07-06 through 2021-07-08, all times were formatted incorrectly and thus we can adjust them using the *replace* Pandas function. We'll then use the same function to adjust the 04:01:00, 08:01:00, and 20:01:00 timed observations. We'll also create a new dataframe **fixed_times_df** to move forward.

Finally, we'll use the *drop* function to eliminate the observation recorded at 13:19:00 and use *drop_duplicates* based on **date** and **time** to eliminate any conflicting observations and repeated entries.

[8]:
```
fixed_times_df = raw_temperature_df.replace(
    to_replace=pd.to_timedelta(['00:00:04', '00:00:08', '00:00:12', '00:00:16',␣
↪'00:00:20']),
    value=pd.to_timedelta(['04:00:00', '08:00:00', '12:00:00', '16:00:00', '20:
↪00:00']),
    )

fixed_times_df.replace(
    to_replace=pd.to_timedelta(['04:01:00', '08:01:00', '20:01:00']),
    value=pd.to_timedelta(['04:00:00', '08:00:00', '20:00:00']),
    inplace=True
    )

fixed_times_df.drop(fixed_times_df[fixed_times_df['time'] == pd.
↪to_timedelta('13:19:00')].index, inplace=True)

fixed_times_df.drop_duplicates(['date', 'time'], keep='first', inplace=True)
```

Checking again on the description of the dataframe, grouped by time, we now have more consistent group sizes and standard deviations (a measure of variance) across groups.

[9]:
```
fixed_times_df.groupby(['time']).describe().loc[:, ['inside_temperature',␣
↪'outside_temperature']]
```

[9]:
```
                 inside_temperature                                          \
                             count       mean       std   min   25%   50%
time
0 days 00:00:00               60.0  71.900000  2.790419  66.0  70.0  72.0
0 days 04:00:00               56.0  70.500000  2.174229  66.0  69.0  70.0
0 days 08:00:00               57.0  70.894737  2.143797  66.0  69.0  71.0
0 days 12:00:00               58.0  73.620690  2.661141  68.0  72.0  74.0
0 days 16:00:00               56.0  73.964286  3.020923  67.0  72.0  74.0
0 days 20:00:00               59.0  73.593220  2.736211  67.0  72.0  74.0

                           outside_temperature                           \
```

time	75%	max	count	mean	std	min
0 days 00:00:00	74.0	80.0	60.0	72.483333	4.575142	62.0
0 days 04:00:00	72.0	79.0	56.0	70.767857	4.276749	62.0
0 days 08:00:00	72.0	76.0	57.0	72.070175	4.938405	61.0
0 days 12:00:00	75.0	79.0	58.0	83.448276	6.885460	67.0
0 days 16:00:00	76.0	81.0	56.0	82.696429	6.972464	67.0
0 days 20:00:00	75.0	81.0	59.0	77.966102	5.542830	65.0

time	25%	50%	75%	max
0 days 00:00:00	69.75	73.0	75.25	82.0
0 days 04:00:00	68.00	72.0	73.25	80.0
0 days 08:00:00	69.00	72.0	75.00	85.0
0 days 12:00:00	78.25	83.0	89.75	95.0
0 days 16:00:00	78.75	82.0	87.00	97.0
0 days 20:00:00	74.00	77.0	81.50	91.0

We'll check the same, now grouping by **day**, **month**, and **year** (which wil have the same output as **season** since data collection has only occurred during summer). All of these have consistent standard deviations and ranges. There is some inconsistency between **day** groups and **month** groups but, because the variance in the temperature readings is similar, we can proceed. It should be note, however, that any statistical tests run will only have the power based on the smallest group in the group pool.

[10]:
```
fixed_times_df.groupby(['day']).describe().loc[:, ['inside_temperature',␣
↪'outside_temperature']]
```

[10]:

	inside_temperature								\
	count	mean	std	min	25%	50%	75%	max	
day									
1	12.0	71.583333	2.539088	69.0	69.00	72.0	73.00	76.0	
2	12.0	71.250000	2.632835	68.0	69.00	71.0	73.00	76.0	
3	10.0	69.000000	1.490712	67.0	68.00	68.5	70.00	72.0	
4	11.0	69.909091	2.300198	66.0	68.50	70.0	71.00	74.0	
5	9.0	70.666667	2.121320	67.0	70.00	70.0	72.00	74.0	
6	11.0	73.181818	2.993933	70.0	71.50	72.0	74.00	79.0	
7	9.0	73.444444	2.006932	71.0	72.00	72.0	75.00	76.0	
8	10.0	72.800000	1.686548	71.0	72.00	72.0	73.75	76.0	
9	10.0	72.000000	3.162278	68.0	70.00	71.0	74.00	78.0	
10	11.0	72.454545	2.910795	68.0	70.00	73.0	75.00	76.0	
11	7.0	70.857143	2.794553	68.0	69.00	70.0	72.00	76.0	
12	10.0	74.100000	2.726414	71.0	72.00	73.0	76.50	78.0	
13	11.0	74.090909	2.700168	70.0	72.50	74.0	75.00	79.0	
14	11.0	75.454545	2.252272	72.0	73.50	77.0	77.00	78.0	
15	12.0	72.333333	3.200379	67.0	71.00	72.5	73.50	79.0	
16	11.0	73.454545	1.967925	70.0	72.50	74.0	75.00	75.0	

17	9.0	74.222222	3.032234	71.0	73.00	74.0	75.00	81.0
18	10.0	73.200000	1.988858	69.0	72.25	74.0	74.75	75.0
19	10.0	72.100000	2.024846	70.0	71.00	71.5	72.00	77.0
20	10.0	71.600000	2.366432	69.0	70.25	71.0	72.75	77.0
21	9.0	70.000000	1.224745	68.0	70.00	70.0	71.00	71.0
22	10.0	72.600000	2.170509	70.0	71.00	71.5	74.75	76.0
23	10.0	72.400000	1.712698	69.0	72.00	72.0	73.75	75.0
24	11.0	71.363636	2.203303	69.0	70.00	70.0	72.50	75.0
25	17.0	71.294118	3.235829	66.0	70.00	71.0	74.00	78.0
26	15.0	70.933333	3.104528	66.0	69.00	70.0	73.50	76.0
27	14.0	72.214286	3.166618	66.0	71.00	72.0	74.00	78.0
28	14.0	73.000000	2.855494	68.0	71.00	73.0	74.00	78.0
29	15.0	74.466667	3.833437	68.0	72.00	75.0	77.00	81.0
30	16.0	74.687500	2.868652	70.0	72.75	75.0	75.50	80.0
31	9.0	72.888889	2.420973	70.0	71.00	72.0	74.00	78.0

	outside_temperature							
	count	mean	std	min	25%	50%	75%	max
day								
1	12.0	73.750000	6.877169	62.0	70.25	75.0	77.00	87.0
2	12.0	70.916667	5.107184	63.0	69.25	71.0	73.25	79.0
3	10.0	67.800000	4.732864	63.0	64.25	66.0	70.25	77.0
4	11.0	70.000000	6.115554	63.0	64.50	70.0	74.50	80.0
5	9.0	73.333333	6.670832	64.0	68.00	73.0	79.00	83.0
6	11.0	78.000000	9.695360	66.0	70.00	76.0	84.50	93.0
7	9.0	79.333333	9.526279	68.0	72.00	77.0	88.00	94.0
8	10.0	72.800000	7.524774	65.0	67.00	70.5	76.25	87.0
9	10.0	75.400000	6.535374	67.0	70.75	74.0	79.50	87.0
10	11.0	75.727273	5.159281	71.0	71.50	74.0	79.50	85.0
11	7.0	77.857143	6.914443	71.0	72.50	76.0	81.50	90.0
12	10.0	81.500000	8.168367	74.0	75.25	77.0	87.00	95.0
13	11.0	79.727273	7.988628	72.0	73.50	78.0	84.50	94.0
14	11.0	81.363636	6.407382	72.0	77.00	82.0	85.50	91.0
15	12.0	78.500000	5.916080	72.0	74.00	76.0	83.25	88.0
16	11.0	80.181818	7.820718	70.0	74.50	80.0	84.50	93.0
17	9.0	80.222222	4.918785	74.0	77.00	80.0	81.00	91.0
18	10.0	77.900000	5.173651	71.0	74.00	77.5	82.50	85.0
19	10.0	78.100000	6.190495	70.0	72.25	79.0	81.50	89.0
20	10.0	76.300000	5.396501	69.0	72.50	75.0	79.25	87.0
21	9.0	75.888889	4.859127	70.0	73.00	74.0	78.00	86.0
22	10.0	73.000000	5.077182	64.0	71.25	73.0	75.00	81.0
23	10.0	73.100000	6.118279	64.0	68.25	73.0	77.50	83.0
24	11.0	75.454545	7.257598	64.0	70.50	74.0	80.50	89.0
25	17.0	75.647059	8.659830	62.0	71.00	74.0	80.00	91.0
26	15.0	78.266667	6.419464	69.0	73.50	77.0	82.00	92.0
27	14.0	80.142857	7.998626	68.0	74.00	77.5	87.00	94.0
28	14.0	78.071429	7.710775	69.0	73.00	75.5	83.50	93.0

```
29                 15.0  76.800000  8.945869  65.0  70.50  75.0  78.00  95.0
30                 16.0  80.312500  8.584628  70.0  72.00  79.5  85.25  97.0
31                  9.0  74.777778  7.980880  61.0  72.00  75.0  78.00  86.0
```

[11]:
```
fixed_times_df.groupby(['month']).describe().loc[:, ['inside_temperature',␣
↪'outside_temperature']]
```

[11]:
```
      inside_temperature                                                  \
                   count       mean       std   min   25%   50%   75%   max
month
6                   34.0  70.529412  3.057383  66.0  68.0  70.5  73.0  79.0
7                  157.0  72.254777  2.700675  66.0  70.0  72.0  74.0  79.0
8                  155.0  73.000000  2.956393  67.0  71.0  73.0  75.0  81.0

      outside_temperature
                   count       mean       std   min   25%   50%    75%   max
month
6                   34.0  79.823529  9.440309  62.0  74.0  78.0  87.75  97.0
7                  157.0  76.286624  7.227131  61.0  71.0  75.0  81.00  94.0
8                  155.0  76.129032  7.330518  62.0  71.0  74.0  80.00  95.0
```

[12]:
```
fixed_times_df.groupby(['year']).describe().loc[:, ['inside_temperature',␣
↪'outside_temperature']]
```

[12]:
```
     inside_temperature                                                  \
                  count       mean       std   min   25%   50%   75%   max
year
2021              346.0  72.419075  2.934174  66.0  70.0  72.0  74.0  81.0

     outside_temperature
                  count       mean       std   min   25%   50%   75%   max
year
2021              346.0  76.563584  7.569409  61.0  71.0  75.0  81.0  97.0
```

0.4 Conclusion

The data is now sufficiently cleaned and may be used for analysis.

[13]:
```
clean_df = fixed_times_df
```

[14]:
```
clean_df.to_csv('../cleaned_data.csv')
```