lab07

February 28, 2021

1 lab 07- Data preprocessing II

Name: Robb Alexander and Ryan Bailis
Class: CSCI349
Semester: 2021SP
Instructor: Brian King

[1]:
```
# Setting things up
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

1) Create a Python function called process_FAA_hourly_data that takes a filename (with path) as a string, and returns a completely processed pandas data frame of the data, ready for analysis. It should do everything that the previous lab did to clean and prepare the file, including a. converting all numeric variables to their simplest numeric types b. converting the date/time stamp (first variable) to a pandas DatetimeIndex, which becomes the actual index for the data frame. c. It should drop the date time variable after moving it to become the index. d. If you did not do this in the last lab, make sure that the DatetimeIndex is localized to a specific timezone! This is very important! What time zone? Did you notice the header? The time stamp is in GMT, so be sure to localize the index accordingly. HOW? After you set up the index, you can do: df.index = df.index.tz_localize(tz='GMT')

[2]:
```
def process_FAA_hourly_data(path):
    df_temps = pd.read_csv(path, skiprows=16)
    df_temps = df_temps.iloc[:,:-1]
    df_temps['Number of Observations (n/a)'] = pd.to_numeric(df_temps['Number␣
↪of Observations (n/a)'], downcast='unsigned')
    df_temps.iloc[:,2:13] = df_temps.iloc[:,2:13].apply(pd.
↪to_numeric,downcast="float")
    df_temps["Date/Time (GMT)"] = pd.to_datetime(df_temps["Date/Time (GMT)"])
    df_temps.set_index('Date/Time (GMT)', inplace=True)
    df_temps.index = df_temps.index.tz_localize(tz='GMT')

    return df_temps
```

2) Use your new function to read in the KIPT data file you downloaded in the last lab. Store your data frame as df_kipt. Output the results of info() and describe() to confirm you read it in correctly.

[3]:
```
df_kipt = process_FAA_hourly_data("/Users/rale/Documents/Programming/
↪csci349_2021sp/data/faa_hourly-KIPT_20000101-20201231_raw.csv")
df_kipt.info()
df_kipt.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 181943 entries, 2000-01-01 00:00:00+00:00 to 2020-12-31
23:00:00+00:00
Data columns (total 12 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   Number of Observations (n/a)    181943 non-null  uint8
 1   Average Temp (F)                180938 non-null  float32
 2   Max Temp (F)                    180938 non-null  float32
 3   Min Temp (F)                    180938 non-null  float32
 4   Average Dewpoint Temp (F)       180816 non-null  float32
 5   1 Hour Precip (in)              30294 non-null   float32
 6   Max Wind Gust (mph)             24708 non-null   float32
 7   Average Relative Humidity (%)   177114 non-null  float32
 8   Average Wind Speed (mph)        181394 non-null  float32
 9   Average Station Pressure (mb)   181647 non-null  float32
 10  Average Wind Direction (deg)    148822 non-null  float32
 11  Max Wind Speed (mph)            181394 non-null  float32
dtypes: float32(11), uint8(1)
memory usage: 9.2 MB
```

[3]:

	Number of Observations (n/a)	Average Temp (F)	Max Temp (F)
count	181943.000000	180938.000000	180938.000000
mean	1.336990	51.373653	51.484375
std	0.851021	18.850195	18.868101
min	0.000000	-11.900000	-11.900000
25%	1.000000	36.000000	36.000000
50%	1.000000	52.000000	52.000000
75%	1.000000	66.900002	66.900002
max	10.000000	102.000000	102.000000

	Min Temp (F)	Average Dewpoint Temp (F)	1 Hour Precip (in)
count	180938.000000	180816.000000	30294.000000
mean	51.269070	40.277889	0.030405
std	18.843729	18.966587	0.078683
min	-11.900000	-20.900000	0.000000
25%	36.000000	26.100000	0.000000
50%	51.799999	41.000000	0.000000
75%	66.900002	57.000000	0.030000

```
max       102.000000                    79.000000           2.350000

       Max Wind Gust (mph)  Average Relative Humidity (%)  \
count         24708.000000                  177114.000000
mean             22.367857                      68.680901
std               7.489910                      19.677162
min               0.000000                       0.000000
25%              19.600000                      54.000000
50%              21.900000                      71.000000
75%              26.500000                      86.000000
max              88.599998                     100.000000

       Average Wind Speed (mph)  Average Station Pressure (mb)  \
count             181394.000000                  181647.000000
mean                   5.907989                    1016.748596
std                    5.187293                       7.636579
min                    0.000000                     508.600006
25%                    0.000000                    1012.200012
50%                    5.400000                    1016.900024
75%                    9.200000                    1021.700012
max                   76.000000                    1044.400024

       Average Wind Direction (deg)  Max Wind Speed (mph)
count                 148822.000000         181394.000000
mean                     175.469009              6.176690
std                      119.212242              5.303467
min                        0.000000              0.000000
25%                       70.000000              0.000000
50%                      210.000000              5.800000
75%                      280.000000              9.200000
max                      360.000000             76.000000
```

3) In the last lab, you assessed the number of missing dates in your data, under the assumption that every hour should have an observation. For now, we'll ignore the fact that there are completely missing hourly observations from the weather station. Report the number of missing values in each variable of df_kipt from the data you have.

[4]:
```
df_kipt.isna().sum()
```

[4]:
```
Number of Observations (n/a)          0
Average Temp (F)                   1005
Max Temp (F)                       1005
Min Temp (F)                       1005
Average Dewpoint Temp (F)          1127
1 Hour Precip (in)               151649
Max Wind Gust (mph)              157235
```

```
Average Relative Humidity (%)      4829
Average Wind Speed (mph)            549
Average Station Pressure (mb)       296
Average Wind Direction (deg)      33121
Max Wind Speed (mph)                549
dtype: int64
```

4) Let's pay attention to "Average Temp (F)". Are there hours of the day are most likely to have missing values? Report the frequency over each hour that has missing "Average Temp (F)" values. Be sure to report the LOCAL times according to the time zone "US/Eastern". Output the hours in order of the most frequently missing to least. Then, as a comment, interpret your findings. Do you see a pattern? Do missing temps tend to happen at a certain time of day? (HINT: This might be challenging. First, as always, select the subset of your data matching your criteria. Then, for these data, look at the index. Date / time data types have LOTS of attributes themselves... such as hour. What do you get if you count these values?)

[5]:
```python
"""
The top missing data is from 10AM to 2PM,
 this is around the middle of the day. So
 it could be lunch time, or the heat/sun
 makes faulty recordings.
"""


missing_hours = df_kipt.index[df_kipt["Average Temp (F)"].isna()]
missing_hours.tz_convert("US/Eastern").hour.value_counts()
```

[5]:
```
11     78
12     68
10     63
13     56
14     46
7      42
9      42
6      42
8      40
15     40
5      38
4      37
3      37
1      36
2      36
17     36
16     35
19     34
22     34
0      34
```

```
18    34
23    33
20    32
21    32
Name: Date/Time (GMT), dtype: int64
```

5) Repeat the previous exercise, but this time, assess the same variable for the day of the week. (NOTE: Be sure to note what a 0 is. In pandas, a 0 for day of the week is a Monday!

[6]:
```python
"""
The top three missing data are Monday Tuesday
 and Wednesday, this means that the earlier
 days of the week tend to have more missing
 data than the later few days.
"""
missing_day = df_kipt.index[df_kipt["Average Temp (F)"].isna()]
missing_day.tz_convert("US/Eastern").dayofweek.value_counts()
```

[6]:
```
1    212
2    195
0    169
3    162
6    115
4    114
5     38
Name: Date/Time (GMT), dtype: int64
```

6) Read in the file FAA_PA_stations.csv provided on Moodle. It's not actually a comma separated file, but a tab separated file. Store the data frame as stations. Show stations.info() after you read in the data.

[7]:
```python
stations = pd.read_csv("/Users/rale/Documents/Programming/csci349_2021sp/data/
↪FAA_PA_stations.csv", sep="\t")
stations.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46 entries, 0 to 45
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ID                46 non-null     object
 1   Name              46 non-null     object
 2   County            45 non-null     object
 3   State             46 non-null     object
 4   Lat               46 non-null     float64
 5   Lon               46 non-null     float64
 6   Elevation (feet)  46 non-null     float64
```

```
dtypes: float64(3), object(4)
memory usage: 2.6+ KB
```

7) As usual, you must always assess your missing data, if any. Are there any observations (rows) in stations that have missing data? Output them, then eliminate them from your data. Be sure to reset_index(drop=True) to reset the index in case any observations are dropped. Output stations.info() again.

[8]:
```
# filter stations 'na', at least one, get trues
# and only use their index, then drop them

stations = stations.drop(stations.isna().any(axis=1)[stations.isna().
 ↪any(axis=1) == True].index)
stations.reset_index(drop=True)
stations.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45 entries, 0 to 45
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ID                45 non-null     object
 1   Name              45 non-null     object
 2   County            45 non-null     object
 3   State             45 non-null     object
 4   Lat               45 non-null     float64
 5   Lon               45 non-null     float64
 6   Elevation (feet)  45 non-null     float64
dtypes: float64(3), object(4)
memory usage: 2.8+ KB
```

8) Examine the data frame of stations by showing the first few observations using stations.head(10) In particular, pay close attention to the variables Lat and Lon. These represent the precise latitude and longitude geolocation for the weather station.

[9]:
```
stations.head(10)
```

[9]:

	ID	Name	County	State	Lat	Lon	Elevation (feet)
0	KABE	ALLENTOWN	LEHIGH	PA	40.65	-75.44	376.0
1	KAOO	ALTOONA	BLAIR	PA	40.29	-78.32	1504.0
2	KBVI	BEAVER FALLS	BEAVER	PA	40.77	-80.39	1230.0
3	KBFD	BRADFORD	MCKEAN	PA	41.80	-78.64	2142.0
4	KBTP	BUTLER	BUTLER	PA	40.77	-79.95	1250.0
5	KCXY	CAPITAL CITY	YORK	PA	40.22	-76.85	340.0
6	KFIG	CLEARFIELD	CLEARFIELD	PA	41.04	-78.41	1516.0
7	KDYL	DOYLESTOWN	BUCKS	PA	40.33	-75.12	394.0
8	KDUJ	DUBOIS	JEFFERSON	PA	41.18	-78.90	1814.0
9	KERI	ERIE	ERIE	PA	42.08	-80.17	730.0

9) Create a new variable in stations called "distKIPT" that stores the distance of every station in PA to Williamsport (KIPT). Use a standard Euclidean distance calculation (over latitude and longitude) to compute the distance between the stations.

[10]:
```
kipt = stations[stations['ID'] == 'KIPT'][['Lon','Lat']]
lon, lat = kipt.values[0][0], kipt.values[0][1]

stations['distKIPT'] = np.sqrt((stations['Lon'] - lon) ** 2 + (stations['Lat']␣
↪- lat) ** 2)
```

10) Output the top 10 stations that are closest to KIPT. (The closest one should be to itself!) The stations should be listed in order of increasing distance from KIPT.

[11]:
```
stations.sort_values(by=['distKIPT'])[:10]['ID']
```

[11]:
```
30    KIPT
27    KSEG
18    KMUI
28    KUNV
5     KCXY
16    KMDT
26    KAVP
13    KLNS
25    KRDG
32    KTHV
Name: ID, dtype: object
```

11) Using your results, go back to the PSU climate website (http://climate.met.psu.edu/data/ida/) and download the faa_hourly data for the THREE closest stations that have hourly data available in the same date range as the data you downloaded from KIPT (i.e. 2000-01-01 à 2020-12-31). (HINT: You may need to skip a station because it does not have data available in this range.) Copy the data into your data folder. Then, read in each data file into its own data frame using your function. You should have four data frames: df_kipt, and three other data frames representing the three closest stations. Show the result of info() on your three new data frames. (HINT: KSEG, KUNV, KCXY)

[12]:
```
df_kseg = process_FAA_hourly_data("/Users/rale/Documents/Programming/
↪csci349_2021sp/data/faa_hourly-KSEG_20000101-20201231_raw.csv")
df_kunv = process_FAA_hourly_data("/Users/rale/Documents/Programming/
↪csci349_2021sp/data/faa_hourly-KUNV_20000101-20201231_raw.csv")
df_kcxy = process_FAA_hourly_data("/Users/rale/Documents/Programming/
↪csci349_2021sp/data/faa_hourly-KCXY_20000101-20201231_raw.csv")

df_kseg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 180858 entries, 2000-01-01 00:00:00+00:00 to 2020-12-31
23:00:00+00:00
```

```
Data columns (total 12 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   Number of Observations (n/a)   180858 non-null  uint8
 1   Average Temp (F)               180242 non-null  float32
 2   Max Temp (F)                   180242 non-null  float32
 3   Min Temp (F)                   180242 non-null  float32
 4   Average Dewpoint Temp (F)      180049 non-null  float32
 5   1 Hour Precip (in)             27623 non-null   float32
 6   Max Wind Gust (mph)            19268 non-null   float32
 7   Average Relative Humidity (%)  176224 non-null  float32
 8   Average Wind Speed (mph)       180029 non-null  float32
 9   Average Station Pressure (mb)  180610 non-null  float32
 10  Average Wind Direction (deg)   131220 non-null  float32
 11  Max Wind Speed (mph)           180029 non-null  float32
dtypes: float32(11), uint8(1)
memory usage: 9.1 MB
```

[13]:
```
df_kunv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 177251 entries, 2000-01-01 00:00:00+00:00 to 2020-12-31
23:00:00+00:00
Data columns (total 12 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   Number of Observations (n/a)   177251 non-null  uint8
 1   Average Temp (F)               175777 non-null  float32
 2   Max Temp (F)                   175777 non-null  float32
 3   Min Temp (F)                   175777 non-null  float32
 4   Average Dewpoint Temp (F)      175766 non-null  float32
 5   1 Hour Precip (in)             7731 non-null    float32
 6   Max Wind Gust (mph)            33669 non-null   float32
 7   Average Relative Humidity (%)  170826 non-null  float32
 8   Average Wind Speed (mph)       176919 non-null  float32
 9   Average Station Pressure (mb)  175686 non-null  float32
 10  Average Wind Direction (deg)   160305 non-null  float32
 11  Max Wind Speed (mph)           176919 non-null  float32
dtypes: float32(11), uint8(1)
memory usage: 9.0 MB
```

[14]:
```
df_kcxy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 135921 entries, 2000-01-01 00:00:00+00:00 to 2020-12-31
23:00:00+00:00
Data columns (total 12 columns):
 #   Column                         Non-Null Count   Dtype
```

```
---  ------                         --------------   -----
 0   Number of Observations (n/a)   135921 non-null  uint8
 1   Average Temp (F)               135445 non-null  float32
 2   Max Temp (F)                   135445 non-null  float32
 3   Min Temp (F)                   135445 non-null  float32
 4   Average Dewpoint Temp (F)      135298 non-null  float32
 5   1 Hour Precip (in)             18708 non-null   float32
 6   Max Wind Gust (mph)            15967 non-null   float32
 7   Average Relative Humidity (%)  131757 non-null  float32
 8   Average Wind Speed (mph)       135712 non-null  float32
 9   Average Station Pressure (mb)  135246 non-null  float32
 10  Average Wind Direction (deg)   123585 non-null  float32
 11  Max Wind Speed (mph)           135712 non-null  float32
dtypes: float32(11), uint8(1)
memory usage: 6.9 MB
```

12) Create a new data frame called df_ave_temps that contains the average temperature from all four stations. Name the variables with the four-letter station identifier (e.g. "KIPT"). The index should have a COMPLETE hourly date range from the start date "20000101 00:00:00 GMT" to finish date "20201231 23:00:00 GMT". The results should be a complete dataset with an observation for every hour. If hourly observations are missing from the station you are copying from, then a NaN value should be stored for that entry. You will use these data for the remainder of this exercise. Show df_ave_temps.info() (NOTE – Depending on how you do this, it might take a bit of processing time. Be patient.)

[15]:
```python
df_ave_temps = pd.date_range(start=df_kipt.index[0], end=df_kipt.index[-1], freq=pd.Timedelta("1H"))
df_ave_temps = pd.DataFrame(index=df_ave_temps)

df_ave_temps = pd.concat([df_ave_temps, df_kipt["Average Temp (F)"].rename('KIPT'), df_kseg["Average Temp (F)"].rename('KSEG'), df_kunv["Average Temp (F)"].rename('KUNV'), df_kcxy["Average Temp (F)"].rename('KCXY')], axis=1)
df_ave_temps.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 184104 entries, 2000-01-01 00:00:00+00:00 to 2020-12-31
23:00:00+00:00
Freq: H
Data columns (total 4 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   KIPT    180938 non-null  float32
 1   KSEG    180242 non-null  float32
 2   KUNV    175777 non-null  float32
 3   KCXY    135445 non-null  float32
dtypes: float32(4)
```

```
memory usage: 4.2 MB
```

13) Each station has missing observations for average temperature. Report the number of missing average temperature readings in df_ave_temps for each location.

[16]:
```
df_ave_temps.isna().sum()
```

[16]:
```
KIPT     3166
KSEG     3862
KUNV     8327
KCXY    48659
dtype: int64
```

14) Now, let's get to why we are considering these alternative stations. Report the number of missing data in KIPT that have at least one alternative station with an existing value. You should output a statement like, "There are XXXX out of XXXX missing KIPT temps that can be restored from other locations." Also, show the first 10 observations of these data that meet this criteria using head(10).

[17]:
```
missing_conditional = df_ave_temps['KIPT'].isna()
available_conditional = df_ave_temps[['KSEG', 'KUNV', 'KCXY']].notna().
 ↪any(axis=1)
final_condition = np.logical_and(missing_conditional, available_conditional)
restorable = df_ave_temps[final_condition]

print("There are", len(restorable.index), "out of", df_ave_temps['KIPT'].isna().
 ↪sum(), "missing KIPT temps that can be restored from other locations.")
restorable.head(10)
```

```
There are 1924 out of 3166 missing KIPT temps that can be restored from other
locations.
```

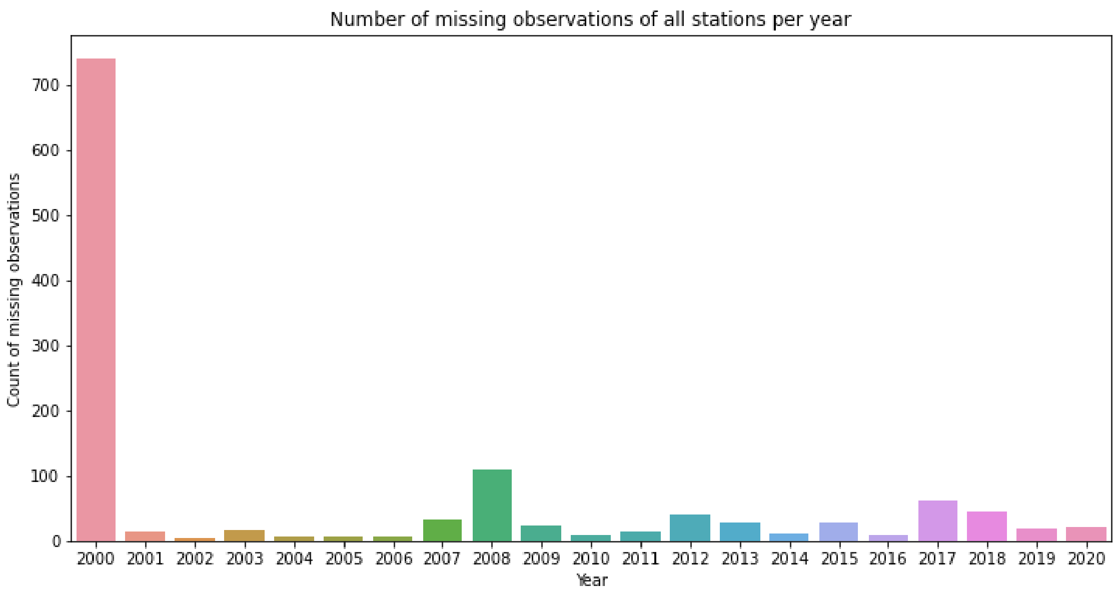
[17]:

	KIPT	KSEG	KUNV	KCXY
2000-01-03 18:00:00+00:00	NaN	53.099998	57.200001	NaN
2000-01-05 17:00:00+00:00	NaN	35.099998	32.000000	NaN
2000-01-06 20:00:00+00:00	NaN	41.000000	39.200001	NaN
2000-01-07 14:00:00+00:00	NaN	36.000000	35.599998	35.599998
2000-01-10 02:00:00+00:00	NaN	39.000000	37.400002	NaN
2000-01-10 03:00:00+00:00	NaN	39.000000	35.599998	NaN
2000-01-10 04:00:00+00:00	NaN	36.000000	NaN	NaN
2000-01-10 05:00:00+00:00	NaN	32.000000	NaN	NaN
2000-01-10 06:00:00+00:00	NaN	34.000000	NaN	NaN
2000-01-10 07:00:00+00:00	NaN	35.349998	NaN	NaN

15) Remember that exercise in the previous lab that gathered the number of missing data by year? Display a barchart showing the number of missing data in KIPT by year that CANNOT be restored from any of the other stations. Annotate the chart with the year that is standing out as the least likely to be successfully restored.

[18]:
```python
unrestorable = df_ave_temps[df_ave_temps.isnull().all(1)]
df_missing_by_year = unrestorable['KIPT'].isna().resample('Y').count()
df_missing_by_year

plt.figure(figsize=(12,6))
ax = sns.barplot(x=df_missing_by_year.index.year, y=df_missing_by_year.values)
ax.set(xlabel='Year', ylabel='Count of missing observations',title="Number of␣
↪missing observations of all stations per year")
```

[18]:
```
[Text(0.5, 0, 'Year'),
 Text(0, 0.5, 'Count of missing observations'),
 Text(0.5, 1.0, 'Number of missing observations of all stations per year')]
```

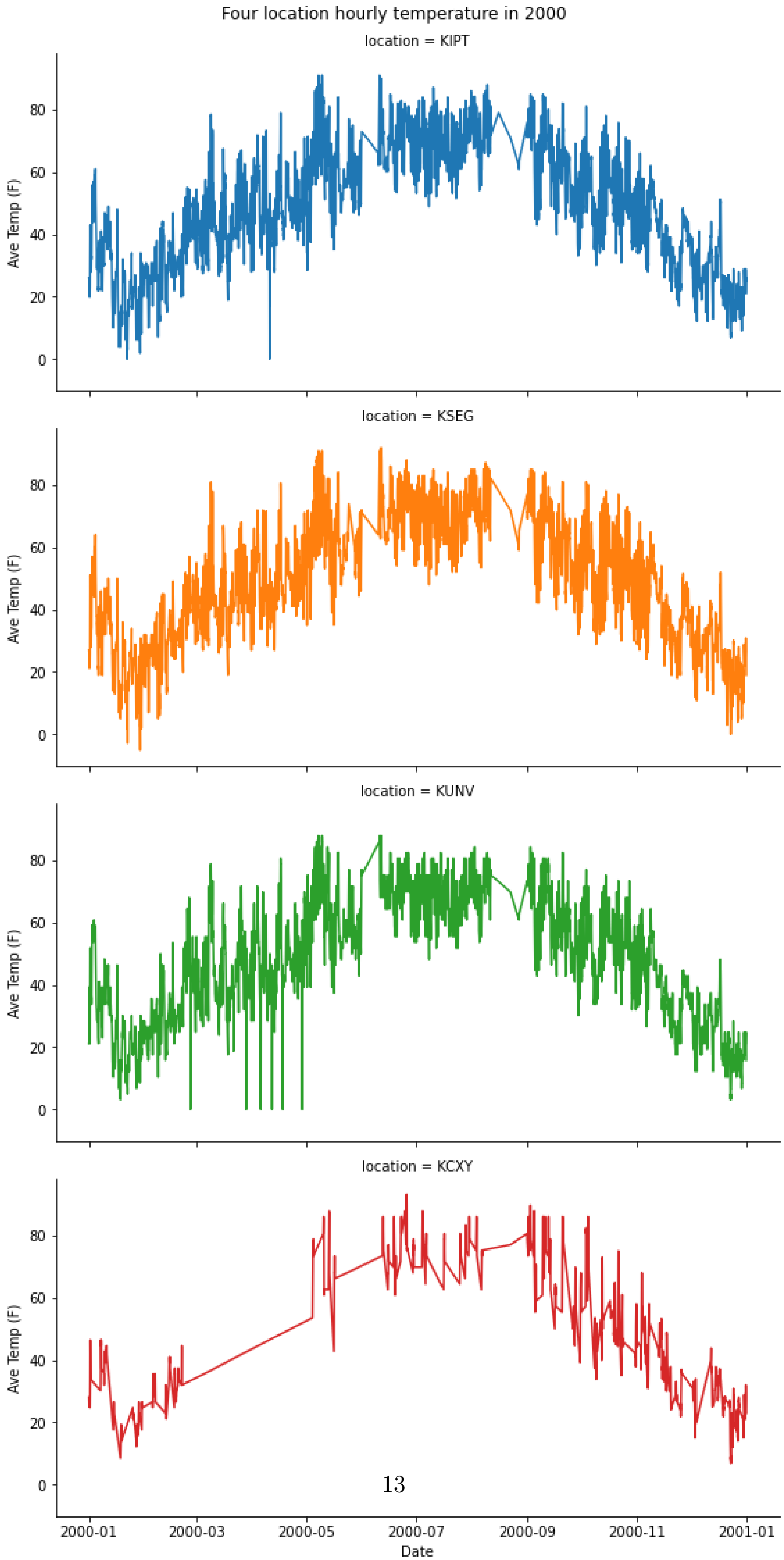


16) It still looks like one year in particular is pretty bad. Confirm this visually by creating a line plot that plots all four stations for that one year, with each station a different color. Make sure KIPT stands out in some way. Only show the data for the one year you answered in the previous exercise. Interpret your results. In particular, do you see any other problems from any stations? Label your plot (e.g. title, axis, legend)

[19]:
```python
"""
There is a gap of data in all locations around june
 and september. KCXY is missing the most data out of
 all of the locations. There is also a few datapoints
 where the temp drops/spikes down to 0. This is very
 problematic.
"""
```

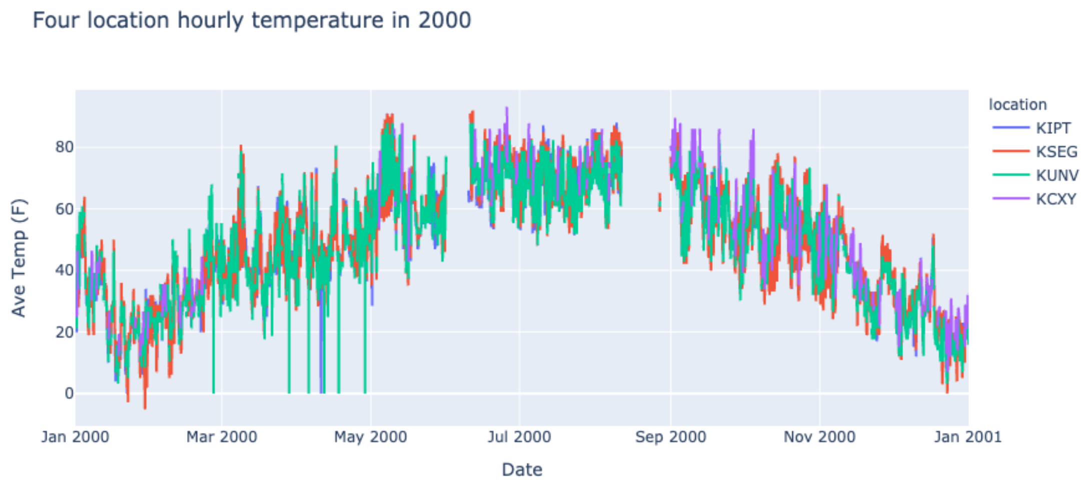
```
melt = df_ave_temps.loc[:"2001-01-01 00:00:00"].
↪melt(value_vars=["KIPT","KSEG","KUNV","KCXY"], var_name="location",␣
↪value_name="Ave Temp (F)", ignore_index=False)
fg = sns.FacetGrid(melt.reset_index(), row="location", hue="location",␣
↪height=4, aspect=2)
fg.map(sns.lineplot, "index", "Ave Temp (F)")

fg.set_axis_labels(x_var="Date")
fg.fig.subplots_adjust(top=0.95)
fg.fig.suptitle('Four location hourly temperature in 2000')
```

[19]: Text(0.5, 0.98, 'Four location hourly temperature in 2000')

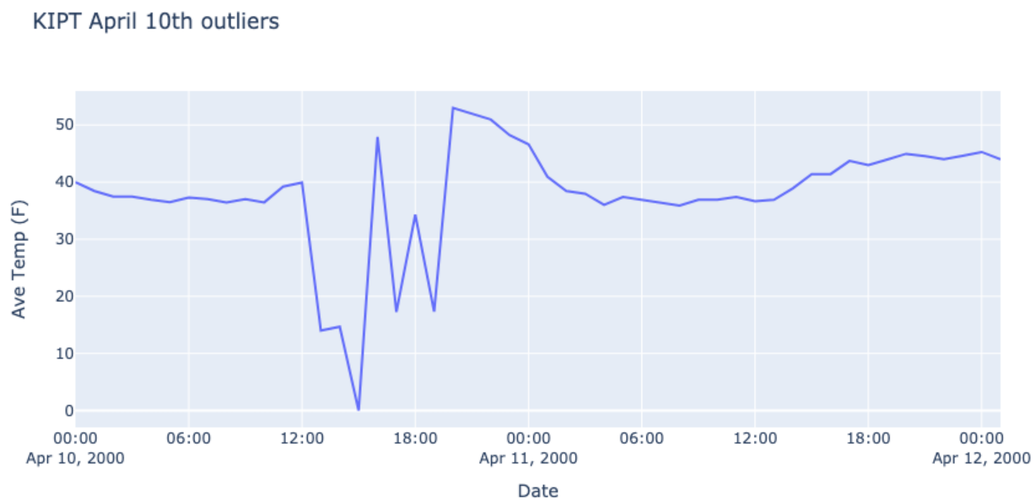

Four location hourly temperature in 2000
location = KIPT
location = KSEG
location = KUNV
location = KCXY
Ave Temp (F)
0
20
40
60
80
2000-01
2000-03
2000-05
2000-07
2000-09
2000-11
2001-01
Date

---

[20]:
```
import plotly.express as px

fig = px.line(melt.reset_index(), x="index", y="Ave Temp (F)",␣
↪color="location", line_group="location",labels={"index": "Date"},␣
↪title="Four location hourly temperature in 2000")
fig.show()
```



[21]:
```
fig = px.line(melt.reset_index()[2400:2450], x="index", y="Ave Temp␣
↪(F)",labels={"index": "Date"}, title="KIPT April 10th outliers")
fig.show()
```



---

17) Looking at your plot of the year 2000 over all stations should reveal that KUNV is problematic at 6 different times. Report these observations, but report them from your full KUNV dataframe. Show only those observations.

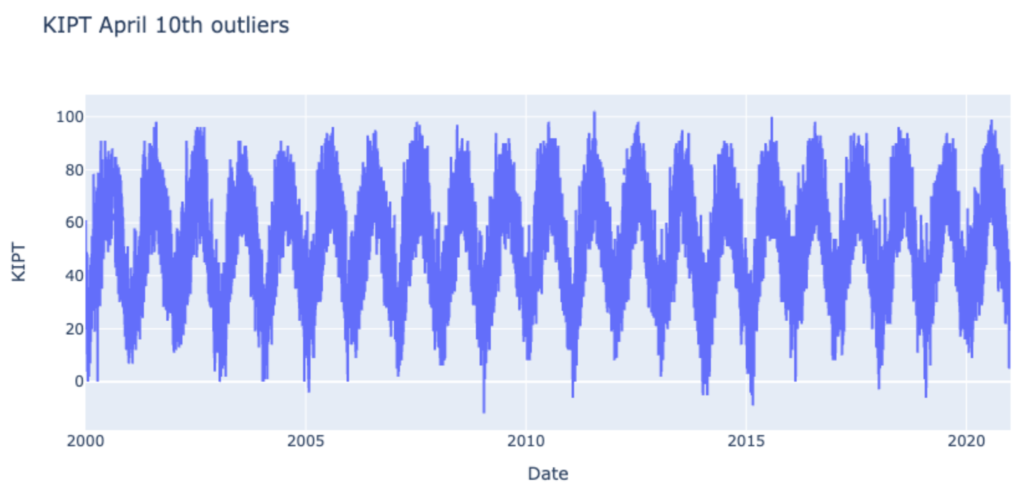
[22]:
```
df_ave_temps["KUNV"][df_ave_temps["KUNV"] == 0]
```

[22]:
```
2000-02-26 13:00:00+00:00    0.0
2000-03-28 14:00:00+00:00    0.0
2000-04-05 14:00:00+00:00    0.0
2000-04-11 22:00:00+00:00    0.0
2000-04-17 21:00:00+00:00    0.0
2000-04-28 15:00:00+00:00    0.0
Name: KUNV, dtype: float32
```

18) How could you algorithmically detect those problems? Keep in mind that simply saying to turn 0.0 into NaN is not an acceptable solution. 0.0 may very well be a real value! The algorithmic approach would to use outliers in each timeframe to see if there are any extreme shifts hour to hour. By looking at the 0 values above, none of them are consecutive. By comparing and using statistics with hours close to each other we can see if they're good data or not. NaNing out the values if they are deemed to be outliers.

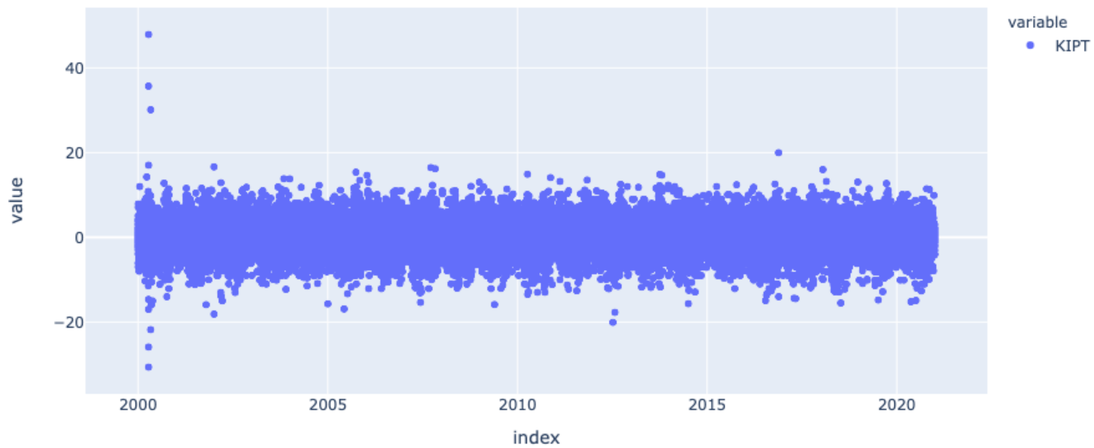
19) Now, write the code to generate line plot(s) for all of KIPT visually, and only KIPT. Look for peculiarities, usually indicated by a sudden change that is outside of what would be considered normal, or an extreme temperature reading that would be impossible to observe in reality. Then, document your findings of areas that you think may be problematic, if any.

[23]:
```
"""
The biggest issues that there are can be seen
 in 2000, where there is a spike to 0 during
 April.
Another could be April 2002, where it hit 91
 but this could be chance.
The issues would primarily show up if there's
 a time during winter where it spikes up, or
 a time during summer where it drops to 0.0
"""

fig = px.line(df_ave_temps.reset_index(), x="index", y="KIPT", labels={"index":␣
↪"Date"}, title="KIPT April 10th outliers")
fig.show()
```



20) Compute a new Series that represents a running delta temperature between adjacent average temperature readings for KIPT. Then, plot the distribution of these data using whatever visualization you think characterizes this distribution best. (HINT: It's a series of observations over a single numeric variable. What type of plot can reveal the distribution of these data?)

[24]:
```
fig = px.scatter(df_ave_temps["KIPT"].diff())
fig.show()
```



21) Perhaps it's more important to select the station that has the most similar values. Write a function called compare_station that takes two Series objects of numeric data, and computes the sum of the absolute value of the difference between each pair of numbers in both Series. You should only sum the values that have valid values for both entries. Return the average of these absolute differences. Then, call compare_station on KIPT and each of the other station, but pass only the average temp vector from each station using your df_ave_temps

[25]:
```
def compare_station(df1, df2):
    diff = df1.sub(df2)
    absolute = diff.abs()
    total = absolute.sum()

    return total / absolute.count()


print("KSEG average difference:", compare_station(df_ave_temps["KIPT"],␣
 ↪df_ave_temps["KSEG"]))
print("KUNV average difference:", compare_station(df_ave_temps["KIPT"],␣
 ↪df_ave_temps["KUNV"]))
print("KCXY average difference:", compare_station(df_ave_temps["KIPT"],␣
 ↪df_ave_temps["KCXY"]))
```

```
KSEG average difference: 2.06606332340859
KUNV average difference: 3.0001271909905665
KCXY average difference: 3.78025037000952
```

22) As we learned in class, you could compute a correlation coefficient between columns of data to determine similarity. Compute the correlation coefficient between the av-

erage temp of KIPT, and each of the other stations you downloaded. They should all be very close to 1, but not quite.

[26]:
```
df_ave_temps.corr().iloc[:, 0][1:]
```

[26]:
```
KSEG    0.988238
KUNV    0.979862
KCXY    0.980908
Name: KIPT, dtype: float64
```

23) Interpret what you have observed so far. Which station is most similar? How would this affect your approach to cleaning your data? Are there other things you might do to clean your data? The most similar station is KSEG, which was the available station that was the shortest distance away. It has also the least amount of missing data out of the three alternative ones. It also has the closest temperature difference from KIPT. All of this is seen in the correlation coefficient, which is the highest out of the three. This means we should take as many missing points in KIPT and use KSEG as the alternative. If KSEG is also missing, then we can take from the second choice of KUNV, since it is missing much less than KCXY, and the temp is closer; even if the correlation coefficient is minisculely lower.

24) Create a new attribute called KIPT_GOOD in your df_ave_temps data frame that keeps all of the original average temp data, but takes the readings from the closest station with available data to replace in the NA values. Be sure to replace the data from the best representative first, then the second best. Ignore the third. When you perform data cleaning, NEVER DELETE YOUR ORIGINAL DATA! Either store it, or just create a separate attribute of cleaned data, or create a separate data frame. Be sure to print out what you are doing as your cell executes. Be sure to include a before and after report to indicate how many values you fixed.

[27]:
```
df_ave_temps["KIPT_GOOD"] = df_ave_temps["KIPT"].copy()
init_missing = df_ave_temps["KIPT_GOOD"].isna().sum()

df_ave_temps["KIPT_GOOD"].fillna(df_ave_temps["KSEG"], inplace=True)
kseg_missing = df_ave_temps["KIPT_GOOD"].isna().sum()
df_ave_temps["KIPT_GOOD"].fillna(df_ave_temps["KUNV"], inplace=True)
final_missing = df_ave_temps["KIPT_GOOD"].isna().sum()

print("Missing", init_missing, "entries initially")
print("KSEG filled", init_missing - kseg_missing, "entries")
print("KUNV filled", kseg_missing - final_missing, "entries")
print("A total of", init_missing - final_missing, "entries fixed, with",␣
↪final_missing, "NaNs left in KIPT")
```

```
Missing 3166 entries initially
KSEG filled 1564 entries
KUNV filled 343 entries
A total of 1907 entries fixed, with 1259 NaNs left in KIPT
```

25) We want to consider setting singleton missing observations, i.e. those missing values that are surrounded by two good observations, as candidates to fill in with the average of their surrounding values. Before we do that, report the number of missing values left in KIPT_GOOD that are singleton missing values.

[28]:
```
df_ave_temps["KIPT_GOOD"][(df_ave_temps["KIPT_GOOD"].isna()) &
                          (df_ave_temps["KIPT_GOOD"].shift(1).notna()) &
                          (df_ave_temps["KIPT_GOOD"].shift(-1).notna())]\
                          .isna().count()
```

[28]: 161

26) Now, convert all singletons missing values in KIPT_GOOD to an average of the surrounding observations. For example [..., 2, NaN, 5, ...] would be filled in with (2+5)/2 = 3.5 for the NaN value. Then, report the number of values that are still missing in KIPT_GOOD.

[29]:
```
df_ave_temps["KIPT_GOOD"][df_ave_temps["KIPT_GOOD"].isnull()] =␣
↪(df_ave_temps["KIPT_GOOD"].shift(-1) + df_ave_temps["KIPT_GOOD"].shift(1)) /␣
↪2
df_ave_temps["KIPT_GOOD"].isna().sum()
```

[29]: 1098

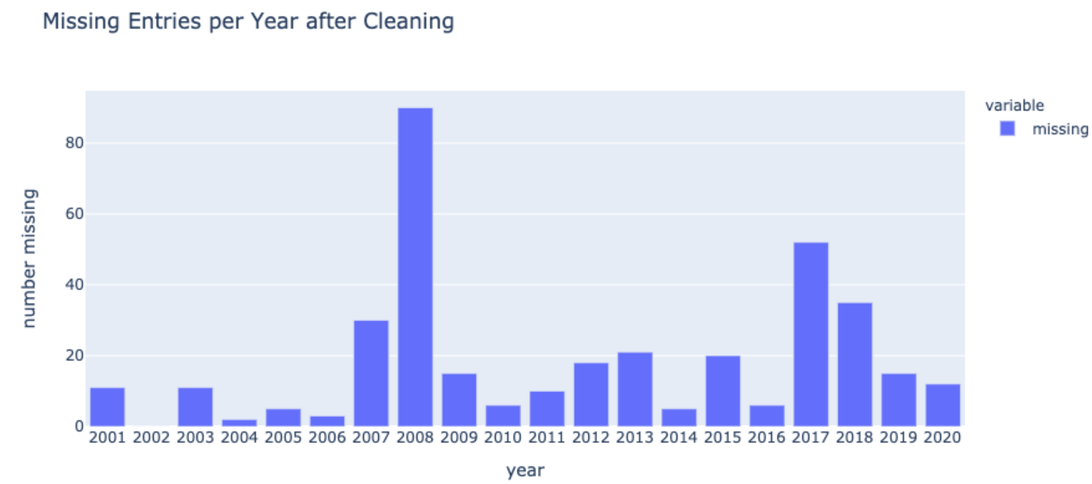
27) Eliminate that first year of data from df_ave_temps. There are too many missing values in these data to make it worthwhile.

[30]:
```
df_ave_temps = df_ave_temps.loc["2001-01-01 00:00:00":]
df_ave_temps["KIPT_GOOD"].isna().sum()
```

[30]: 367

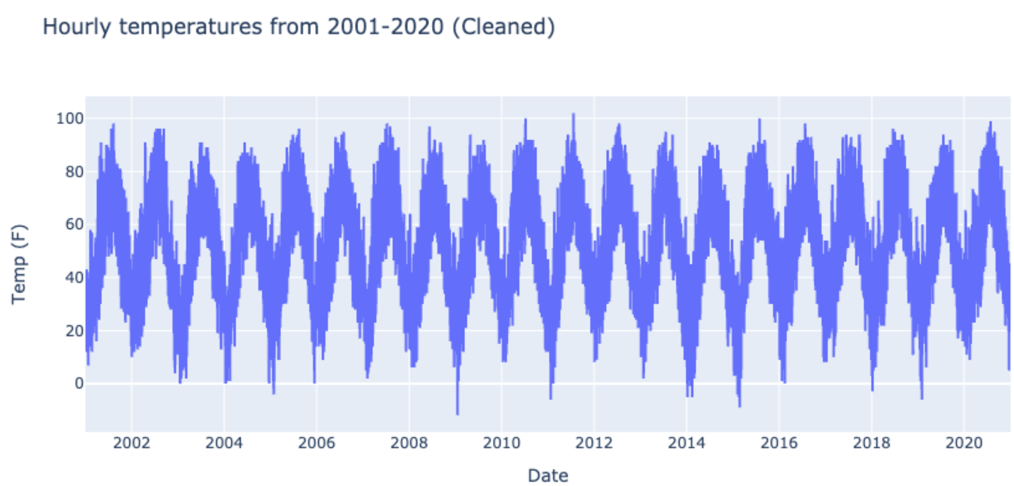
28) Generate an updated barplot of the total number of missing values in df_ave_temps.KIPT_GOOD by year.

[31]:
```
df_missing = pd.DataFrame(1, index=df_ave_temps["KIPT_GOOD"].
↪isna()[df_ave_temps["KIPT_GOOD"].isna() == True].index, columns=['missing'])
fig = px.bar(df_missing.resample('Y').count(), labels={'index':'year','value':
↪'number missing'}, title="Missing Entries per Year after Cleaning")

fig.update_layout(
    xaxis = dict(
        tick0="2000-01-01",
        dtick = 86400000*365,
        tickformat = '%Y'
    )
)

fig.show()
```

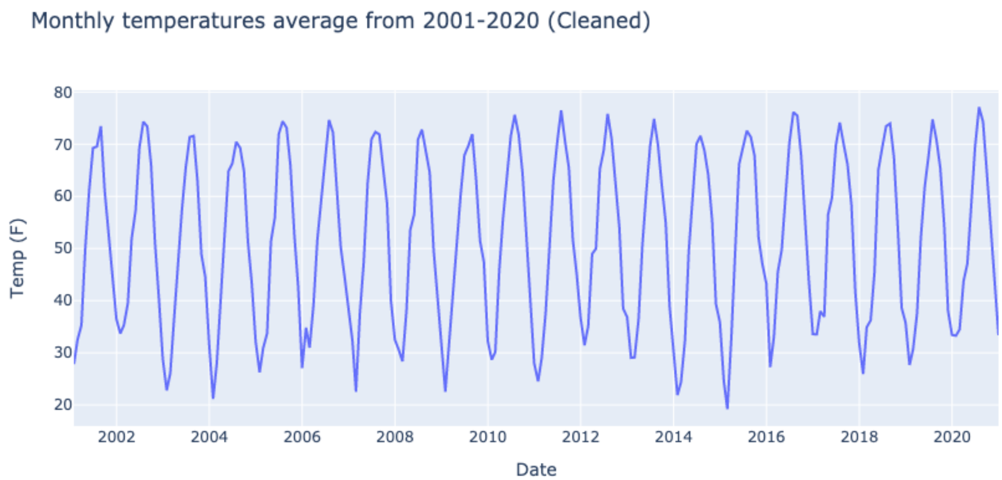


29) Finally, create some good, clean line plots of KIPT_GOOD. Create at least three plots using different averaging times. One should be the raw data. I would suggest creating another one by month, and then the final one by year. Be sure they are labeled.

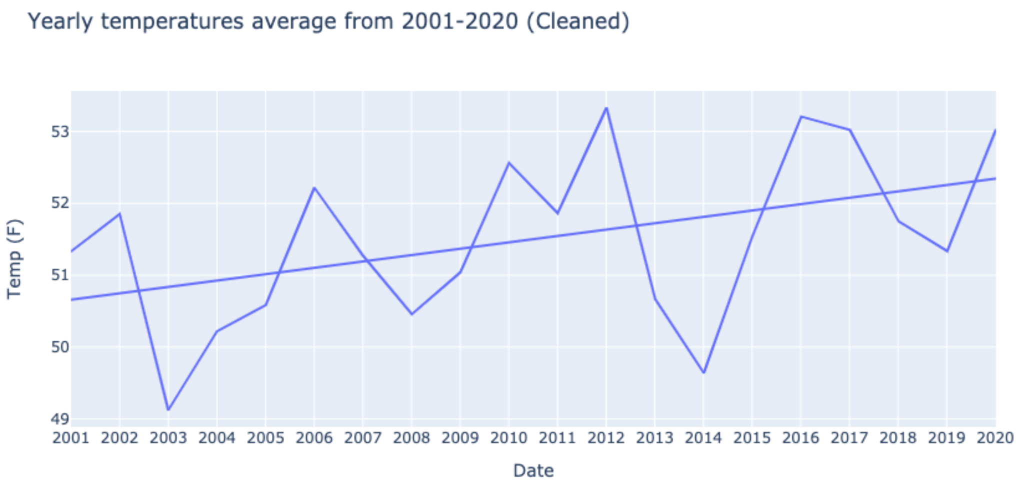
[32]:
```
fig = px.line(df_ave_temps.reset_index(), x="index", y="KIPT_GOOD",␣
↪labels={"index": "Date","KIPT_GOOD":"Temp (F)"}, title="Hourly temperatures␣
↪from 2001-2020 (Cleaned)")
fig.show()
```



[33]:
```
fig = px.line(df_ave_temps.resample('M').mean().reset_index(), x="index",␣
↪y="KIPT_GOOD", labels={"index": "Date","KIPT_GOOD":"Temp (F)"},␣
↪title="Monthly temperatures average from 2001-2020 (Cleaned)")
fig.show()
```



[34]:
```
fig = px.line(df_ave_temps.resample('Y').mean().reset_index(), x="index",␣
↪y="KIPT_GOOD", labels={"index": "Date","KIPT_GOOD":"Temp (F)"},␣
↪title="Yearly temperatures average from 2001-2020 (Cleaned)")

scat = px.scatter(df_ave_temps.resample('Y').mean().reset_index(), x="index",␣
↪y="KIPT_GOOD", trendline="ols")
trendline = scat.data[1]
fig.add_trace(trendline)

fig.update_layout(
    xaxis = dict(
        tick0="2000-01-01",
        dtick = 86400000*365,
        tickformat = '%Y'
    )
)
fig.show()
```


Yearly temperatures average from 2001-2020 (Cleaned)
Temp (F)
53
52
51
50
49
2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020
Date