<table>
<thead>
<tr>
<th>Station Code</th>
<th>STATION NAME</th>
<th>LATITUDE</th>
<th>LONGITUDE</th>
<th>STATE</th>
<th>ICAO CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>69007093217</td>
<td>WEATHER WXPOD8270</td>
<td>+37.186</td>
<td>-115.567</td>
<td>US</td>
<td>US</td>
</tr>
<tr>
<td>69007093218</td>
<td>WEATHER WXPOD7018</td>
<td>+33.667</td>
<td>-115.567</td>
<td>US</td>
<td>US</td>
</tr>
<tr>
<td>69007093219</td>
<td>WEATHER WXPOD8270</td>
<td>+33.700</td>
<td>-115.567</td>
<td>US</td>
<td>US</td>
</tr>
<tr>
<td>69007099999</td>
<td>WEATHER WXPOD 7018</td>
<td>+36.000</td>
<td>-115.567</td>
<td>US</td>
<td>US</td>
</tr>
</tbody>
</table>

You can also use logic indexing, just as a fast way to filter our data.
Why did this happen? Remember that in Pandas, datatypes are important to perform calculations and plotting.

You can also apply more complex operations and group by using time.

Standarize your data cleaning processes, this is helpful for several reasons:

- ...
Sometimes we have different data sources of related data. In our base case we have a dataframe of stations with station metadata and some time series data. The shape of the dataframe is `2 rows × 8784 columns`.

### SOLUTIONS

We first need to set date and time as index and convert columns to numeric types using `pandas.to_numeric` method.

```python
chicago_weather['date'] = pd.to_datetime(chicago_weather['date'])
chicago_weather['LAT'] = pd.to_numeric(chicago_weather['LAT'], errors = 'coerce')
```

This produces a `SettingWithCopyWarning` and `IndexingWithCopyWarning` as the index is being re-arranged and a view of a copy object is created.

```python
chicago_weather.set_index('date', inplace=True)
```

This can be avoided by using `.loc` instead of `[ ]` to change the values.

```python
chicago_weather.loc[0, 'LAT'] = pd.to_numeric(chicago_weather.loc[0, 'LAT'], errors = 'coerce')
```

Then we can merge the station data with the time series data.

```python
stations_il = pd.read_csv('stations_il.csv')
```

```python
stations_il.head()
```

```python
Out[97]:
station         latitude
0  CHICAGO OHARE INTERNATIONAL AIRPORT, IL US  41.978125
1  CHICAGO MIDWAY AIRPORT, IL US  41.673125
2  OHARE INTERNATIONAL AIRPORT, IL US  41.878125
3  OHARE INTERNATIONAL AIRPORT, IL US  41.978125
4  OHARE INTERNATIONAL AIRPORT, IL US  41.978125
```

This can be done using `merge` function.

```python
chicago_weather_wide = chicago_weather.merge(stations_il, on='station')
```

### Merge data

We then need to transform the data to `long` format if we want to use `pandas.plotting.scatter_matrix` method. This will allow us to explore the data.

```python
chicago_weather = pd.melt(chicago_weather, id_vars=['date'], value_vars=['LAT', 'WMO_WIND'])
```

```python
chicago_weather_wide = pd.melt(chicago_weather_wide, id_vars=['date', 'station'], value_vars=['LAT', 'WMO_WIND'])
```

```python
chicago_weather_wide.head()
```

```python
Out[100]:
   date          variable   value station              value
0 01-01 00:00:00           LAT -13.972668  CHICAGO OHARE  -13.972668
1 01-01 00:00:00     WMO_WIND -13.972668  CHICAGO OHARE -13.972668
2 01-01 01:00:00           LAT -13.972668  CHICAGO MIDWAY -13.972668
3 01-01 01:00:00     WMO_WIND -13.972668  CHICAGO MIDWAY -13.972668
4 01-01 02:00:00           LAT -13.972668  OHARE INTERNATIONAL -13.972668
```

### Data transformation

Once we have the data in `long` format, we can use the `rolling` method to calculate rolling averages and standard deviations.

```python
chicago_weather['tmp'] = chicago_weather['value'].rolling(30).mean()
chicago_weather['tmp'] = chicago_weather['value'].rolling(30).std()
```

```python
chicago_weather_wide['tmp'] = chicago_weather_wide['value'].rolling(30).mean()
chicago_weather_wide['tmp'] = chicago_weather_wide['value'].rolling(30).std()
```

The most useful transformation, at least for me is the ability of storing big data files taking less space on disk.

```python
chicago_weather_wide.to_csv('chicago_weather_wide.csv', index=False)
```

### Data visualization

We can visualize the data using the `pandas.plotting.scatter_matrix` method.

```python
plt = seaborn.scatterplot(x='LAT', y='WMO_WIND', data=chicago_weather_wide, hue='date', size='date', sizes=(10, 100), alpha=0.8)
```

```python
plt = seaborn.scatterplot(x='LAT', y='WMO_WIND', data=chicago_weather_wide, hue='station', size='date', sizes=(10, 100), alpha=0.8)
```

```python
plt = seaborn.scatterplot(x='LAT', y='WMO_WIND', data=chicago_weather_wide, hue='date', size='date', sizes=(10, 100), alpha=0.8)
```

### Conclusion

By using the `pandas` library, we can easily manipulate and visualize the data. This allows us to explore the data and gain insights into anomalies in our data in the Section I.