

7.3 Data Cleaning

Name: Jann Moises Nyll B. De los Reyes

Section: CPE22S3

Cleaning Data

About the data

In this notebook, we will be collecting daily temperature data from the [National Centers for Environmental Information \(NCEI\) API](#). We will use the Global Historical Climatology Network - Daily (GHCND) data set; see the documentation [here](#).

This data was collected for the LaGuardia Airport station in New York City for October 2018. It contains:

- the daily minimum temperature (TMIN)
- the daily maximum temperature (TMAX)
- the daily average temperature (TAVG)

Note: The NCEI is part of the National Oceanic and Atmospheric Administration (NOAA) and, as you can see from the URL for the API, this resource was created when the NCEI was called the NCDC. Should the URL for this resource change in the future, you can search for the NCEI weather API to find the updated one.

In addition, we will be using S&P 500 stock market data for the S&P 500 and data for bitcoin for 2017 through 2018.

Setup

We need to import pandas and read in our data to get started:

```
1 import pandas as pd
2
3 df = pd.read_csv('nyc_temperatures.csv')
4 df.head()
```

	date	datatype	station	attributes	value
0	2018-10-01T00:00:00	TAVG	GHCND:USW00014732	H,S,	21.2
1	2018-10-01T00:00:00	TMAX	GHCND:USW00014732	,W,2400	25.6
2	2018-10-01T00:00:00	TMIN	GHCND:USW00014732	,W,2400	18.3
3	2018-10-02T00:00:00	TAVG	GHCND:USW00014732	H,S,	22.7
4	2018-10-02T00:00:00	TMAX	GHCND:USW00014732	,W,2400	26.1

Renaming Columns

We start out with the following columns:

```
1 df.columns
2
Index(['date', 'datatype', 'station', 'attributes', 'value'], dtype='object')
```

We want to rename the `value` column to indicate it contains the temperature in Celsius and the `attributes` column to say `flags` since each value in the comma-delimited string is a different flag about the data collection. For this task, we use the `rename()` method and pass in a dictionary mapping the column names to their new names. We pass `inplace=True` to change our original dataframe instead of getting a new one back:

```
1 df.rename(
2     columns = {
3         'value' : 'temp_C',
4         'attributes' : 'flags'
5     }, inplace = True
6 )
```

Those columns have been successfully renamed:

```
1 df.columns
Index(['date', 'datatype', 'station', 'flags', 'temp_C'], dtype='object')
```

We can also perform string operations on the column names with `rename()` :

```
1 df.rename(str.upper,axis = 'columns').columns
Index(['DATE', 'DATATYPE', 'STATION', 'FLAGS', 'TEMP_C'], dtype='object')
```

~ Type Conversion

The date column is not currently being stored as a datetime :

```
1 df.dtypes

date           object
datatype       object
station        object
flags          object
temp_C         float64
dtype: object
```

Let's perform the conversion with `pd.to_datetime()` :

```
1 df.loc[:, 'date'] = pd.to_datetime(df.date)
2 df.dtypes

<ipython-input-8-80606e5f8dec>:1: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inplace instead of always setting a new
df.loc[:, 'date'] = pd.to_datetime(df.date)
date           datetime64[ns]
datatype       object
station        object
flags          object
temp_C         float64
dtype: object
```

Now we get useful information when we use `describe()` on this column:

```
1 df.date.describe()

<ipython-input-9-f7d3fa946723>:1: FutureWarning: Treating datetime data as categorical rather than numeric in `.describe` is deprecated and will be removed in a future vers
df.date.describe()
count          93
unique          31
top    2018-10-01 00:00:00
freq           3
first    2018-10-01 00:00:00
last     2018-10-31 00:00:00
Name: date, dtype: object
```

We can use `tz_localize()` on a `DatetimeIndex` / `PeriodIndex` to convert to a desired timezone:

```
1 pd.date_range(start = '2018-10-25', periods =2,freq='D').tz_localize('EST')

DatetimeIndex(['2018-10-25 00:00:00-05:00', '2018-10-26 00:00:00-05:00'], dtype='datetime64[ns, EST]', freq=None)
```

This also works with a `Series` / `DataFrame` with one of the aforementioned as its `Index` . Let's read in the CSV again for this example and set the `date` column to be the index and stored as a datetime:

```
1 eastern =pd.read_csv('nyc_temperatures.csv',index_col = 'date', parse_dates = True
2               ).tz_localize('EST')
3 eastern.head()
```

	datatype	station	attributes	value
date				
2018-10-01 00:00:00-05:00	TAVG	GHCND:USW00014732	H,S	21.2
2018-10-01 00:00:00-05:00	TMAX	GHCND:USW00014732	„W,2400	25.6
2018-10-01 00:00:00-05:00	TMIN	GHCND:USW00014732	„W,2400	18.3
2018-10-02 00:00:00-05:00	TAVG	GHCND:USW00014732	H,S	22.7
2018-10-02 00:00:00-05:00	TMAX	GHCND:USW00014732	„W,2400	26.1

We can use `tz.convert()` to convert to another timezone from there. If we convert the Eastern datetimes to UTC, they will now be at 5 AM, since `pandas` will use the offsets to convert:

```
1 eastern.tz_convert('UTC').head()
```

	datatype	station	attributes	value
date				
2018-10-01 05:00:00+00:00	TAVG	GHCND:USW00014732	H,S	21.2
2018-10-01 05:00:00+00:00	TMAX	GHCND:USW00014732	„W,2400	25.6
2018-10-01 05:00:00+00:00	TMIN	GHCND:USW00014732	„W,2400	18.3
2018-10-02 05:00:00+00:00	TAVG	GHCND:USW00014732	H,S	22.7
2018-10-02 05:00:00+00:00	TMAX	GHCND:USW00014732	„W,2400	26.1

be discussed in [chapter 4](#)).

```
1 eastern.to_period('M').index
```

[illegible]

We now get a `PeriodIndex` which we can change back into a `DatetimeIndex` with `pd.to_timestamp()`:

```
1 eastern.to period('M').to timestamp().index
```

[illegible]

We can use the `assign()` method for working with multiple columns at once (or creating new ones). Since our `date` column has already been converted, we need to read in the data again:

```
1 df = pd.read_csv('nyc_temperatures.csv').rename(
2     columns = {
3         'value' : 'temp_C',
4         'attributes' : 'flags'
5     }
6 )
7
8 new_df = df.assign(
9     date=pd.to_datetime(df.date),
10    temp_F = (df.temp_C *9/5 ) +32
11 )
12
13 new_df.dtypes
```

```
date          datetime64[ns]
datatype      object
station       object
flags         object
temp_C        float64
temp_F        float64
dtype: object
```

The `date` column now has datetimes and the `temp_F` column was added:

```
1 new_df.head()
```

	date	datatype	station	flags	temp_C	temp_F
0	2018-10-01	TAVG	GHCND:USW00014732	H,S,	21.2	70.16
1	2018-10-01	TMAX	GHCND:USW00014732	„W,2400	25.6	78.08
2	2018-10-01	TMIN	GHCND:USW00014732	„W,2400	18.3	64.94
3	2018-10-02	TAVG	GHCND:USW00014732	H,S,	22.7	72.86
4	2018-10-02	TMAX	GHCND:USW00014732	„W,2400	26.1	78.98

We can also use `astype()` to perform conversions. Let's create columns of the integer portion of the temperatures in Celsius and Fahrenheit:

```
1 df = df.assign(date = pd.to_datetime(df.date),
2               temp_C_whole = df.temp_C.astype('int'),
3               temp_F = (df.temp_C*9/5)+32,
4               temp_F_whole = lambda x: x.temp_F.astype('int')
5               )
6
7 df.head()
```

	date	datatype	station	flags	temp_C	temp_C_whole	temp_F	temp_F_whole
0	2018-10-01	TAVG	GHCND:USW00014732	H,S,	21.2	21	70.16	70
1	2018-10-01	TMAX	GHCND:USW00014732	„W,2400	25.6	25	78.08	78
2	2018-10-01	TMIN	GHCND:USW00014732	„W,2400	18.3	18	64.94	64
3	2018-10-02	TAVG	GHCND:USW00014732	H,S,	22.7	22	72.86	72
4	2018-10-02	TMAX	GHCND:USW00014732	„W,2400	26.1	26	78.98	78

Creating categories:

```
1 df_with_categories = df.assign(
2     station =df.station.astype('category'),
3     datatype =df.datatype.astype('category')
4 )
5
6 df_with_categories.dtypes
```

```
date          datetime64[ns]
datatype      category
station       category
flags         object
temp_C        float64
temp_C_whole  int64
temp_F        float64
temp_F_whole  int64
dtype: object
```

Our categories have no order, but this is something pandas supports:

```
1 pd.Categorical(
2     ['med','med','low','high'],
3     categories = ['low','med','high'],
4     ordered= True
5 )

['med', 'med', 'low', 'high']
Categories (3, object): ['low' < 'med' < 'high']
```

✓ Reordering,reindexing, and sorting

Say we want to find the hottest days in the temperature data; we can sort our values by the `temp_C` column with the largest on top to find this:

```
1 df.sort_values(by = 'temp_C',ascending = False).head(10)
```

	date	datatype	station	flags	temp_C	temp_C_whole	temp_F	temp_F_whole
19	2018-10-07	TMAX	GHCND:USW00014732	„W,2400	27.8	27	82.04	82
28	2018-10-10	TMAX	GHCND:USW00014732	„W,2400	27.8	27	82.04	82
31	2018-10-11	TMAX	GHCND:USW00014732	„W,2400	26.7	26	80.06	80
4	2018-10-02	TMAX	GHCND:USW00014732	„W,2400	26.1	26	78.98	78
10	2018-10-04	TMAX	GHCND:USW00014732	„W,2400	26.1	26	78.98	78
25	2018-10-09	TMAX	GHCND:USW00014732	„W,2400	25.6	25	78.08	78
1	2018-10-01	TMAX	GHCND:USW00014732	„W,2400	25.6	25	78.08	78
7	2018-10-03	TMAX	GHCND:USW00014732	„W,2400	25.0	25	77.00	77
27	2018-10-10	TAVG	GHCND:USW00014732	H,S,	23.8	23	74.84	74
30	2018-10-11	TAVG	GHCND:USW00014732	H,S,	23.4	23	74.12	74

```
1 df.sort_values(by = ['temp_C','date'],ascending =False).head(10)
```

	date	datatype	station	flags	temp_C	temp_C_whole	temp_F	temp_F_whole
28	2018-10-10	TMAX	GHCND:USW00014732	„W,2400	27.8	27	82.04	82
19	2018-10-07	TMAX	GHCND:USW00014732	„W,2400	27.8	27	82.04	82
31	2018-10-11	TMAX	GHCND:USW00014732	„W,2400	26.7	26	80.06	80
10	2018-10-04	TMAX	GHCND:USW00014732	„W,2400	26.1	26	78.98	78
4	2018-10-02	TMAX	GHCND:USW00014732	„W,2400	26.1	26	78.98	78
25	2018-10-09	TMAX	GHCND:USW00014732	„W,2400	25.6	25	78.08	78
1	2018-10-01	TMAX	GHCND:USW00014732	„W,2400	25.6	25	78.08	78
7	2018-10-03	TMAX	GHCND:USW00014732	„W,2400	25.0	25	77.00	77
27	2018-10-10	TAVG	GHCND:USW00014732	H,,S,	23.8	23	74.84	74
30	2018-10-11	TAVG	GHCND:USW00014732	H,,S,	23.4	23	74.12	74

When just looking for the n-largest values, rather than wanting to sort all the data, we can use `nlargest()` :

```
1 df.nlargest(n=5,columns = 'temp_C')
```

	date	datatype	station	flags	temp_C	temp_C_whole	temp_F	temp_F_whole
19	2018-10-07	TMAX	GHCND:USW00014732	„W,2400	27.8	27	82.04	82
28	2018-10-10	TMAX	GHCND:USW00014732	„W,2400	27.8	27	82.04	82
31	2018-10-11	TMAX	GHCND:USW00014732	„W,2400	26.7	26	80.06	80
4	2018-10-02	TMAX	GHCND:USW00014732	„W,2400	26.1	26	78.98	78
10	2018-10-04	TMAX	GHCND:USW00014732	„W,2400	26.1	26	78.98	78

We use `nsmallest()` for the n-smallest values. Note that these can also take a list of columns; however, it won't work with the `date` column.

```
1 df.nsmallest(n=5,columns=['temp_C','date']).sort_index()
```

	date	datatype	station	flags	temp_C	temp_C_whole	temp_F	temp_F_whole
53	2018-10-18	TMIN	GHCND:USW00014732	„W,2400	6.7	6	44.06	44
62	2018-10-21	TMIN	GHCND:USW00014732	„W,2400	6.1	6	42.98	42
65	2018-10-22	TMIN	GHCND:USW00014732	„W,2400	5.6	5	42.08	42
74	2018-10-25	TMIN	GHCND:USW00014732	„W,2400	6.1	6	42.98	42
77	2018-10-26	TMIN	GHCND:USW00014732	„W,2400	5.6	5	42.08	42

```
1 df.nsmallest(n=5,columns=['temp_C','date'])
```

	date	datatype	station	flags	temp_C	temp_C_whole	temp_F	temp_F_whole
65	2018-10-22	TMIN	GHCND:USW00014732	„W,2400	5.6	5	42.08	42
77	2018-10-26	TMIN	GHCND:USW00014732	„W,2400	5.6	5	42.08	42
62	2018-10-21	TMIN	GHCND:USW00014732	„W,2400	6.1	6	42.98	42
74	2018-10-25	TMIN	GHCND:USW00014732	„W,2400	6.1	6	42.98	42
53	2018-10-18	TMIN	GHCND:USW00014732	„W,2400	6.7	6	44.06	44

The `sample()` method will give us rows (or columns with `axis=1`) at random. We can provide the `random_state` to make this reproducible. The index after we do this is jumbled:

```
1 df.sample(5, random_state = 0).index
Int64Index([2, 30, 55, 16, 13], dtype='int64')
```

We can use `sort_index()` to order it again:

```
1 df.sample(5,random_state = 0).sort_index().index
Int64Index([2, 13, 16, 30, 55], dtype='int64')
```

The `sort_index()` method can also sort columns alphabetically:

```
1 df.sort_index(axis =1).head()
```

	datatype	date	flags	station	temp_C	temp_C_whole	temp_F	temp_F_whole
0	TAVG	2018-10-01	H,S	GHCND:USW00014732	21.2	21	70.16	70
1	TMAX	2018-10-01	„W,2400	GHCND:USW00014732	25.6	25	78.08	78
2	TMIN	2018-10-01	„W,2400	GHCND:USW00014732	18.3	18	64.94	64
3	TAVG	2018-10-02	H,S	GHCND:USW00014732	22.7	22	72.86	72
4	TMAX	2018-10-02	„W,2400	GHCND:USW00014732	26.1	26	78.98	78

This can make selection with `loc` easier for many columns:

```
1 df.sort_index(axis = 1).head().loc[:, 'temp_C': 'temp_F_whole']
```

	temp_C	temp_C_whole	temp_F	temp_F_whole
0	21.2	21	70.16	70
1	25.6	25	78.08	78
2	18.3	18	64.94	64
3	22.7	22	72.86	72
4	26.1	26	78.98	78

We must sort the index to compare two dataframes. If the index is different, but the data is the same, they will be marked not-equal:

```
1 df.equals(df.sort_values(by = 'temp_C'))
```

False

Sorting the index solves this issue:

```
1 df.equals(df.sort_values(by = 'temp_C').sort_index())
```

True

We can also use `reset_index()` to get a fresh index and move our current index into a column for safe keeping. This is especially useful if we had data, such as the date, in the index that we don't want to lose:

```
1 df[df.datatype == 'TAVG'].head().reset_index()
```

	index	date	datatype	station	flags	temp_C	temp_C_whole	temp_F	temp_F_whole
0	0	2018-10-01	TAVG	GHCND:USW00014732	H,S	21.2	21	70.16	70
1	3	2018-10-02	TAVG	GHCND:USW00014732	H,S	22.7	22	72.86	72
2	6	2018-10-03	TAVG	GHCND:USW00014732	H,S	21.8	21	71.24	71
3	9	2018-10-04	TAVG	GHCND:USW00014732	H,S	21.3	21	70.34	70
4	12	2018-10-05	TAVG	GHCND:USW00014732	H,S	20.3	20	68.54	68

Let's set the `date` column as our index:

```
1 df.set_index('date', inplace = True)
2 df.head()
```

	datatype	station	flags	temp_C	temp_C_whole	temp_F	temp_F_whole
date							
2018-10-01	TAVG	GHCND:USW00014732	H,S	21.2	21	70.16	70
2018-10-01	TMAX	GHCND:USW00014732	„W,2400	25.6	25	78.08	78
2018-10-01	TMIN	GHCND:USW00014732	„W,2400	18.3	18	64.94	64
2018-10-02	TAVG	GHCND:USW00014732	H,S	22.7	22	72.86	72
2018-10-02	TMAX	GHCND:USW00014732	„W,2400	26.1	26	78.98	78

Now that we have a `DatetimeIndex`, we can do datetime slicing. As long as we provide a date format that pandas understands, we can grab the data. To select all of 2018, we simply use `df['2018']`, for the third quarter of 2018 we can use `['2018-Q3']`, grabbing October is as simple as using `df['2018-10']`; these can also be combined to build ranges. Let's grab October 11, 2018 through October 12, 2018 (inclusive of both endpoints):

```
1 df['2018-10-11': '2018-10-12']
```

	datatype	station	flags	temp_C	temp_C_whole	temp_F	temp_F_whole
date							
2018-10-11	TAVG	GHCND:USW00014732	H,,S,	23.4	23	74.12	74
2018-10-11	TMAX	GHCND:USW00014732	„W,2400	26.7	26	80.06	80
2018-10-11	TMIN	GHCND:USW00014732	„W,2400	21.7	21	71.06	71
2018-10-12	TAVG	GHCND:USW00014732	H,,S,	18.3	18	64.94	64
2018-10-12	TMAX	GHCND:USW00014732	„W,2400	22.2	22	71.96	71
2018-10-12	TMIN	GHCND:USW00014732	„W,2400	12.2	12	53.96	53

```
1 df['2018'] # select all 2018
```

```
<ipython-input-38-dc4e98db4798>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single df['2018']
```

	datatype	station	flags	temp_C	temp_C_whole	temp_F	temp_F_whole
date							
2018-10-01	TAVG	GHCND:USW00014732	H,,S,	21.2	21	70.16	70
2018-10-01	TMAX	GHCND:USW00014732	„W,2400	25.6	25	78.08	78
2018-10-01	TMIN	GHCND:USW00014732	„W,2400	18.3	18	64.94	64
2018-10-02	TAVG	GHCND:USW00014732	H,,S,	22.7	22	72.86	72
2018-10-02	TMAX	GHCND:USW00014732	„W,2400	26.1	26	78.98	78
...
2018-10-30	TMAX	GHCND:USW00014732	„W,2400	13.9	13	57.02	57
2018-10-30	TMIN	GHCND:USW00014732	„W,2400	7.2	7	44.96	44
2018-10-31	TAVG	GHCND:USW00014732	H,,S,	12.6	12	54.68	54
2018-10-31	TMAX	GHCND:USW00014732	„W,2400	17.8	17	64.04	64
2018-10-31	TMIN	GHCND:USW00014732	„W,2400	7.2	7	44.96	44

93 rows × 7 columns

```
1 df['2018-10'] # grabbing october 2018
```

```
<ipython-input-40-c4c69e646553>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single df['2018-10']
```

	datatype	station	flags	temp_C	temp_C_whole	temp_F	temp_F_whole
date							
2018-10-01	TAVG	GHCND:USW00014732	H,,S,	21.2	21	70.16	70
2018-10-01	TMAX	GHCND:USW00014732	„W,2400	25.6	25	78.08	78
2018-10-01	TMIN	GHCND:USW00014732	„W,2400	18.3	18	64.94	64
2018-10-02	TAVG	GHCND:USW00014732	H,,S,	22.7	22	72.86	72
2018-10-02	TMAX	GHCND:USW00014732	„W,2400	26.1	26	78.98	78
...
2018-10-30	TMAX	GHCND:USW00014732	„W,2400	13.9	13	57.02	57
2018-10-30	TMIN	GHCND:USW00014732	„W,2400	7.2	7	44.96	44
2018-10-31	TAVG	GHCND:USW00014732	H,,S,	12.6	12	54.68	54
2018-10-31	TMAX	GHCND:USW00014732	„W,2400	17.8	17	64.04	64
2018-10-31	TMIN	GHCND:USW00014732	„W,2400	7.2	7	44.96	44

93 rows × 7 columns

Reindexing allows us to conform our axis to contain a given set of labels. Let's turn to the S&P 500 stock data in the `data/sp500.csv` file to see an example of this. Notice we only have data for trading days (weekdays, excluding holidays):

```
1 sp = pd.read_csv (
2     'sp500.csv', index_col = 'date', parse_dates = True
3     ).drop(columns=['adj_close'])
4
5 sp.head(10).assign(
6     day_of_week=lambda x:x.index.day_name()
7 )
```

	high	low	open	close	volume	day_of_week
date						
2017-01-03	2263.879883	2245.129883	2251.570068	2257.830078	3770530000	Tuesday
2017-01-04	2272.820068	2261.600098	2261.600098	2270.750000	3764890000	Wednesday
2017-01-05	2271.500000	2260.449951	2268.179932	2269.000000	3761820000	Thursday
2017-01-06	2282.100098	2264.060059	2271.139893	2276.979980	3339890000	Friday
2017-01-09	2275.489990	2268.899902	2273.590088	2268.899902	3217610000	Monday
2017-01-10	2279.270020	2265.270020	2269.719971	2268.899902	3638790000	Tuesday
2017-01-11	2275.320068	2260.830078	2268.600098	2275.320068	3620410000	Wednesday
2017-01-12	2271.780029	2254.250000	2271.139893	2270.439941	3462130000	Thursday
2017-01-13	2278.679932	2271.510010	2272.739990	2274.639893	3081270000	Friday
2017-01-17	2272.080078	2262.810059	2269.139893	2267.889893	3584990000	Tuesday

If we want to look at the value of a portfolio (group of assets) that trade on different days, we need to handle the mismatch in the index. Bitcoin, for example, trades daily.

```
1 bitcoin = pd.read_csv(
2     'bitcoin.csv', index_col = 'date', parse_dates = True
3 ).drop(columns=['market_cap'])
4
5 # every day's closing price = S&P 500 close + Bitcoin close (same for other metrics)
6 portfolio = pd.concat(
7     [sp, bitcoin], sort = False
8 ).groupby(pd.Grouper(freq = 'D')).sum()
9
10 portfolio.head(10).assign(
11     day_of_week = lambda x: x.index.day_name()
12 )
```

	high	low	open	close	volume	day_of_week
date						
2017-01-01	1003.080000	958.700000	963.660000	998.330000	147775008	Sunday
2017-01-02	1031.390000	996.700000	998.620000	1021.750000	222184992	Monday
2017-01-03	3307.959883	3266.729883	3273.170068	3301.670078	3955698000	Tuesday
2017-01-04	3432.240068	3306.000098	3306.000098	3425.480000	4109835984	Wednesday
2017-01-05	3462.600000	3170.869951	3424.909932	3282.380000	4272019008	Thursday
2017-01-06	3328.910098	3148.000059	3285.379893	3179.179980	3691766000	Friday
2017-01-07	908.590000	823.560000	903.490000	908.590000	279550016	Saturday
2017-01-08	942.720000	887.250000	908.170000	911.200000	158715008	Sunday
2017-01-09	3189.179990	3148.709902	3186.830088	3171.729902	3359486992	Monday
2017-01-10	3194.140020	3166.330020	3172.159971	3176.579902	3754598000	Tuesday

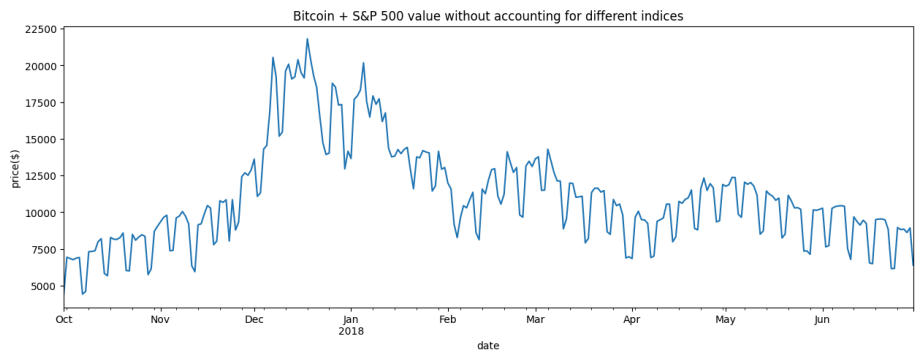
It may not be immediately obvious what is wrong with the previous data, but with a visualization we can easily see the cyclical pattern of drops on the days the stock market is closed.

We will need to import matplotlib now:

```
1 import matplotlib.pyplot as plt # we use this module for plotting
```

Now, we can see why we need reindex:

```
1 portfolio['2017-Q4' : '2018-Q2'].plot(
2     y='close', figsize = (15,5), legend=False,
3     title = 'Bitcoin + S&P 500 value without accounting for different indices'
4 ) #plot the closing price from Q4 2017 through Q2 2018
5 plt.ylabel('price($')) # label the y-axis
6 plt.show() # show the plot
```

We need to align index of the S&P 500 to match bitcoin in order to fix this. We will use the `reindex()` method, but by default we get `NaN` for the values don't have data for:

```
1 sp.reindex(bitcoin.index).head(10).assign(  
2     day_of_week=lambda x: x.index.day_name()  
3 )
```

date	high	low	open	close	volume	day_of_week
2017-01-01	NaN	NaN	NaN	NaN	NaN	Sunday
2017-01-02	NaN	NaN	NaN	NaN	NaN	Monday
2017-01-03	2263.879883	2245.129883	2251.570068	2257.830078	3.770530e+09	Tuesday
2017-01-04	2272.820068	2261.600098	2261.600098	2270.750000	3.764890e+09	Wednesday
2017-01-05	2271.500000	2260.449951	2268.179932	2269.000000	3.761820e+09	Thursday
2017-01-06	2282.100098	2264.060059	2271.139893	2276.979980	3.339890e+09	Friday
2017-01-07	NaN	NaN	NaN	NaN	NaN	Saturday
2017-01-08	NaN	NaN	NaN	NaN	NaN	Sunday
2017-01-09	2275.489990	2268.899902	2273.590088	2268.899902	3.217610e+09	Monday
2017-01-10	2279.270020	2265.270020	2269.719971	2268.899902	3.638790e+09	Tuesday

So now we have rows for every day of the year, but all the weekends and holidays have `NaN` values. To address this, we can specify how to handle missing values with the `method` argument. In this case, we want to forward fill, which will put the weekend and holiday values as the value they had for the Friday (or end of trading week) before:

```
1 sp.reindex(  
2     bitcoin.index, method = 'ffill'  
3 ).head(10).assign(  
4     day_of_week=lambda x: x.index.day_name()  
5 )
```

date	high	low	open	close	volume	day_of_week
2017-01-01	NaN	NaN	NaN	NaN	NaN	Sunday
2017-01-02	NaN	NaN	NaN	NaN	NaN	Monday
2017-01-03	2263.879883	2245.129883	2251.570068	2257.830078	3.770530e+09	Tuesday
2017-01-04	2272.820068	2261.600098	2261.600098	2270.750000	3.764890e+09	Wednesday
2017-01-05	2271.500000	2260.449951	2268.179932	2269.000000	3.761820e+09	Thursday
2017-01-06	2282.100098	2264.060059	2271.139893	2276.979980	3.339890e+09	Friday
2017-01-07	2282.100098	2264.060059	2271.139893	2276.979980	3.339890e+09	Saturday
2017-01-08	2282.100098	2264.060059	2271.139893	2276.979980	3.339890e+09	Sunday
2017-01-09	2275.489990	2268.899902	2273.590088	2268.899902	3.217610e+09	Monday
2017-01-10	2279.270020	2265.270020	2269.719971	2268.899902	3.638790e+09	Tuesday

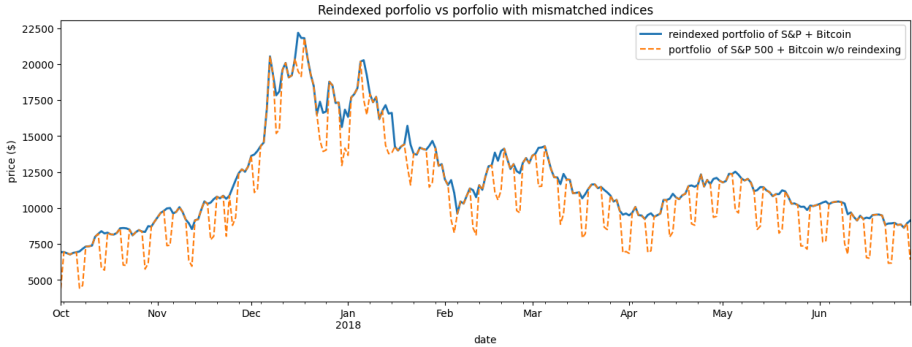
This isn't perfect though. We probably want 0 for the volume traded and to put the closing price for the open, high, low, and close on the days the market is closed:

```
1 import numpy as np
2
3 sp_reindexed = sp.reindex(
4     bitcoin.index
5 ).assign(
6     volume =lambda x: x.volume.fillna(0), # put 0 when market is closed
7     close = lambda x: x.close.fillna(method='ffill'), # carry this forward
8     #take the closing price if these aren't available
9     open=lambda x: np.where(x.open.isnull(),x.close,x.open),
10    high=lambda x: np.where(x.high.isnull(),x.close,x.high),
11    low=lambda x: np.where(x.low.isnull(),x.close,x.low)
12 )
13
14 sp_reindexed.head(10).assign(
15     day_of_week=lambda x: x.index.day_name()
16 )
```

	high	low	open	close	volume	day_of_week
date						
2017-01-01	NaN	NaN	NaN	NaN	0.000000e+00	Sunday
2017-01-02	NaN	NaN	NaN	NaN	0.000000e+00	Monday
2017-01-03	2263.879883	2245.129883	2251.570068	2257.830078	3.770530e+09	Tuesday
2017-01-04	2272.820068	2261.600098	2261.600098	2270.750000	3.764890e+09	Wednesday
2017-01-05	2271.500000	2260.449951	2268.179932	2269.000000	3.761820e+09	Thursday
2017-01-06	2282.100098	2264.060059	2271.139893	2276.979980	3.339890e+09	Friday
2017-01-07	2276.979980	2276.979980	2276.979980	2276.979980	0.000000e+00	Saturday
2017-01-08	2276.979980	2276.979980	2276.979980	2276.979980	0.000000e+00	Sunday
2017-01-09	2275.489990	2268.899902	2273.590088	2268.899902	3.217610e+09	Monday
2017-01-10	2278.879980	2265.879980	2268.719980	2268.899902	3.207700e+09	Tuesday

If we create visualization comparing the reindexed data to the first attempt, we see how reindexing helped maintain the asset value when the market was closed:

```
1 #every day's closing price = S&P 500 close adjusted for market closure + Bitcoin close(same for other metrics)
2 fixed_portfolio = pd.concat([sp_reindexed,bitcoin],sort=False).groupby(pd.Grouper(freq='D')).sum()
3
4 ax = fixed_portfolio['2017-Q4':'2018-Q2'].plot(
5     y='close', label = 'reindexed portfolio of S&P + Bitcoin', figsize=(15,5), linewidth=2,
6     title='Reindexed porfolio vs porfolio with mismatched indices'
7 ) #plot the reindexed portfolio's closing price from Q4 2017 through Q2 2018
8
9 portfolio['2017-Q4': '2018-Q2'].plot(
10    y='close', ax=ax, linestyle = '--', label = 'portfolio of S&P 500 + Bitcoin w/o reindexing'
11 ).set_ylabel('price ($)') # add line for original portfolio for comparison and label y-axis
12
13 plt.show() #show the plot
```



Double-click (or enter) to edit

Double-click (or enter) to edit