Time Series Aggregation

PROFESSIONAL & CONTINUING EDUCATION
UNIVERSITY of WASHINGTON

W

Overview

- Download the PM2.5 dataset
- Convert EAV formatted data to time series dataframe and parse data
- Aggregate data by timestamp and attribute
- Upsample data to regular time intervals
- Interpoloate missing data points

Downloading the Data Set

Pulling information from the web:

```python
import pandas as pd


pm2_file =
pd.read_csv("https://library.startlearninglabs.uw
.edu/DATASCI400/Datasets/BeijingPM2_IOT.csv")
print(pm2_file.head())
```

Create pandas time stamp object

- Built in functions to bin by timeframe
- Bin data by time intervals:
  - Minute
  - Month
- Pull out useful information such as weekday name

Converting CSV to a Pandas Time Series

```python
import pandas as pd

# you can have pandas convert the date directly
pm2_file = pd.read_csv(file_path,
    parse_dates=['TimeStamp'],
    infer_datetime_format=True)

print(pm2_file.head())
```

```
                 TimeStamp      Attribute Value
0 2009-12-31 22:17:00            Iws  1.79
1 2009-12-31 22:43:00  precipitation     0
2 2009-12-31 23:19:00            Iws  4.92
3 2009-12-31 23:22:00           HUMI    43
4 2010-01-01 00:02:00           PRES  1019
```

Explore the Data Set

UCI PM2.5 dataset:

- Hourly meteorological sensor readings for five different cities in China
- Each sensor contains approximate timestamp and an attribute in EAV format
- Time stamps are approximate

PROFESSIONAL & CONTINUING EDUCATION
UNIVERSITY of WASHINGTON

Steps to Parsing the Data Set

- Subset the Data
  - Pull out temperature, precipitation and humidity readings
- Index the data by time stamp
- Coerce column data types
- Replace missing values

Why Subset the data?

- Work with just the data we need
  - Smaller
  - Faster

Name	Type	Size	Value
pm2_df	DataFrame	(21905, 3)	Column names: TimeStamp, Attribute, Value
pm2_file	DataFrame	(65143, 3)	Column names: TimeStamp, Attribute, Value

Subsetting the data

```
pm2_df = pm2_file[(pm2_file['Attribute'] == 'precipitation') |
    (pm2_file['Attribute'] == 'TEMP') |
    (pm2_file['Attribute'] == 'HUMI')].copy()


print(pm2_df.head())
```

```
          TimeStamp           Attribute Value
1   2009-12-31 22:43:00   precipitation     0
3   2009-12-31 23:22:00            HUMI    43
7   2010-01-01 01:01:00            TEMP   -14
8   2010-01-01 01:04:00            TEMP   -12
10  2010-01-01 01:11:00            TEMP   -11
```

Setting an Index

`Set_index()`

- Set index on column of interest
- Allows for easy aggregation, resampling and interpolation of data
- Built-in methods for working with time intervals

PROFESSIONAL & CONTINUING EDUCATION
UNIVERSITY *of* WASHINGTON

Setting an Index

```python
import pandas as pd
pm2_file =
pd.read_csv("https://library.startlearninglabs.uw.edu/DATASCI400/D
atasets/BeijingPM2_IOT.csv",
                        parse_dates=['TimeStamp'],
                        infer_datetime_format=True)


pm2_df = pm2_df.set_index(['TimeStamp'])
```

To call a function on an object: **object.function(parameters)**

Set_index([column_name])

Using the Index

```python
pm2_df = pm2_df.set_index(['TimeStamp']
```

	Attribute	Value
TimeStamp		
2009-12-31 22:43:00	precipitation	0
2009-12-31 23:22:00	HUMI	43
2010-01-01 01:01:00	TEMP	-14
2010-01-01 01:04:00	TEMP	-12
2010-01-01 01:11:00	TEMP	-11

Call out year, year & Month

```python
print(pm2_df['2010'])
print(pm2_df['2010-12'])
```

Call out month & year on the index

```python
print(pm2_df.index.year)
print(pm2_df.index.month)
```

Count the number of observations per year

```python
print(pm2_df.groupby(pm2_df.index.year).count())
```

	Attribute	Value
TimeStamp		
2009	2	2
2010	21322	21322
2011	581	581

Coercing Data Types

`astype()`

- Specify a data type
- Avoid programming errors
- Ensure accurate calculations

Coercing Data Types

```
pm2_df['Value'] =
pm2_df['Value'].astype(float)
```

Before

```
                     Attribute Value
TimeStamp
2009-12-31 22:43:00  precipitation    0
2009-12-31 23:22:00           HUMI    43
2010-01-01 01:01:00           TEMP   -14
2010-01-01 01:04:00           TEMP   -12
2010-01-01 01:11:00           TEMP   -11
```

After

```
                     Attribute   Value
TimeStamp
2009-12-31 22:43:00  precipitation    0.0
2009-12-31 23:22:00           HUMI    43.0
2010-01-01 01:01:00           TEMP   -14.0
2010-01-01 01:04:00           TEMP   -12.0
2010-01-01 01:11:00           TEMP   -11.0
```

Deal with Missing Values

- Avoid errors
- Ensure accurate calculations
- `fillna()` – replaces missing values
- `dropna()` – removes missing values

Deal with Missing Values

```
pm2_df = pm2_df.dropna(axis=0,
how='any')
```

Axis=0 drops the entire row

Axis=1 drops the entire column

How=any drops rows with any NaN value

How=all drops rows that are all NaN

Aggregating by a single column

- Pandas `groupby()`

- Group data by a column, such as Attribute

- Perform calculations on each group, such as get the mean of all temperature readings (calculate average temperature)

Aggregating by a single column

```
grouped = pm2_df.groupby('Attribute')


print(grouped)

<pandas.core.groupby.DataFrameGroupBy
object at 0x000001E0A791BE48>
```

HUMI			TEMP			precipitation		
	Attribute	Value		Attribute	Value		Attribute	Value
TimeStamp			TimeStamp			TimeStamp		
2009-12-31 23:22:00	HUMI	43.0	2010-01-01 01:01:00	TEMP	-14.0	2009-12-31 22:43:00	precipitation	0.0
2010-01-01 01:36:00	HUMI	55.0	2010-01-01 01:04:00	TEMP	-12.0	2010-01-01 02:16:00	precipitation	0.0
2010-01-01 02:20:00	HUMI	47.0	2010-01-01 01:11:00	TEMP	-11.0	2010-01-01 03:45:00	precipitation	0.0
2010-01-01 02:21:00	HUMI	51.0	2010-01-01 01:43:00	TEMP	-11.0	2010-01-01 03:53:00	precipitation	0.0
2010-01-01 02:59:00	HUMI	43.0	2010-01-01 04:14:00	TEMP	-12.0	2010-01-01 06:10:00	precipitation	0.0

Aggregating by multiple columns

- Pandas `groupby()`
- List of columns
- Mixed data types use
  `dataframe_name.column_name` format

Aggregating by time stamp

```
year_attr_group =
pm2_df.groupby([pm2_df.index.year,
pm2_df.Attribute])
for x,y in year_attr_group:
    print(x)
    print(y)
```

Resampling a Time Series

- Change the frequency of your time series
- Pandas `resample()`
  - Downsample
  - Upsample

Resampling a time series

```
downsample = pm2_df.resample('1min')
print(downsample.head(100))
```

Resample()
  –Only works on numerical columns

```
                            Value
TimeStamp
2009-12-31 22:43:00          0.0
2009-12-31 22:44:00          NaN
2009-12-31 22:45:00          NaN
2009-12-31 22:46:00          NaN
2009-12-31 22:47:00          NaN
2009-12-31 22:48:00          NaN
2009-12-31 22:49:00          NaN
2009-12-31 22:50:00          NaN
2009-12-31 22:51:00          NaN
2009-12-31 22:52:00          NaN
2009-12-31 22:53:00          NaN
2009-12-31 22:54:00          NaN
```

Upsampling time series

```
upsampled =
pm2_df.groupby('Attribute').resample(
'1S').mean()

for group, x in
upsampled.iteritems():
    print(group)
    print(x)
```

Interpolating Missing Data Points

Interpolate()

- Estimates missing values
- Lots of calculations using up memory

```
upsampled_interpolated =
upsampled.interpolate(method='linear')
```
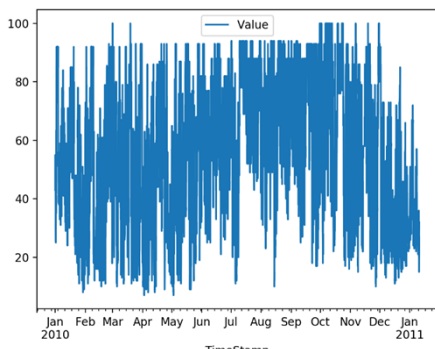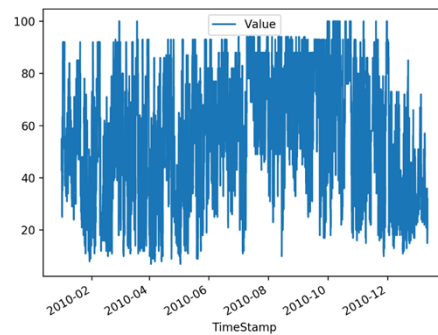
Comparing Plots

Interpolated Data almost identical to original data.

Interpolated

Original

Interpolating Temperature Readings

Your Turn

W

Interpolate Missing Temperature Data

1. Upsample the temperature dataset to 1 second intervals

2. Interpolate the temperature data using a **polynomial method** instead of linear

3. Plot the distributions of the original temperature data and compare with a plot of the upsampled and interpolated data

Summary

> Working with time series data in pandas

> Aggregating time series data

> Resampling and interpreting time series data