Cálculos, ploteas, group-by, y multi-indices

Andrew Reeve
School of Earth and Climate Sciences
University of Maine

Interpolar Series de Tiempo

- dataframes de serie de tiempo
  - indices con **`datetime`**
  - posiblemente será muchos NaNs debido la hora de colecciona de datos
- interpolación para llenar valores de NaN values con otros valores
  - palabra de clave **`method`** para metido de interpolación
  - **`linear`**, **`time`**, **`pad`**, etc.

```python
# this is asreeve's custom module for reading levelogger files
from read_logger_files import to_dataframe
import pandas as pd

# create a data structure to hold individual data sets, and read them into dictionary
# to_dataframe returns a tuple contining the data set and header info
data_dict = {}
data_dict["shal"] = to_dataframe("examples/data/MSH_shallow_may2020Compensated.xle")[0]
data_dict["deep"] = to_dataframe("examples/data/MSH_deep_may2020Compensated.xle")[0]

# put the data into a dataframe with a multiindex
df = pd.DataFrame()
for depth in data_dict.keys():
    data_dict[depth] = data_dict[depth].drop("block", axis=1)
    m_idx = pd.MultiIndex.from_product(([depth], data_dict[depth].columns))
    data_dict[depth].columns = m_idx
    df = pd.concat((df, data_dict[depth]))
# change index stings to datetime objects and sort index
df.index = pd.to_datetime(df.index)
df = df.sort_index()
# interpolate the deep data onto the times shallow data was recorded
df.deep = df.deep.interpolate(method="time")
# drop rows with NaN values, save data
df = df.dropna()
df.to_hdf("examples/data/MSH_shal&deep_may2020.hdf", key="root")

# reampling aggregates data based on time
# to get daily mean values
df.resample('d').mean()
```

Ventana Rodante y Plotear

- aplicación permite funciones pesadas sobre subconjunto de dataframe
- tipo de ventana cambia el pesado de datos con posición de muestro
- 'ventana' desliza por una columna de datos

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as pl
import scipy.signal.windows as win

df = pd.read_hdf("examples/data/MSH_shal&deep_may2020.hdf")
# lets look at the multiindexed data,
df.plot(y=[("deep", "Lev"), ("shal", "Lev")])
# or use matplotlib directly
fig, sp = pl.subplots()
sp.plot(df.index, df.loc[:, ("shal", "Lev")], label="shallow")
sp.plot(df.index, df.loc[:, ("deep", "Lev")], label="deep")
sp.legend()
sp.set_ylabel("Water Pressure (m of H2O)")
pl.close()

# make a rolling window, calulate mean
a_win = df.loc[:, ("shal", "Lev")].rolling(9, center=True)
a_roll = a_win.agg("mean")

# can do other window types defined in scipy.signal.windows library
b_win = df.loc[:, ("shal", "Lev")].rolling(9, center=True, win_type="triang")
b_roll = b_win.agg("mean")

df = df.assign(box=a_roll)
df = df.assign(tri=b_roll)

fig, sp = pl.subplots()
sp.plot(df.index, df.loc[:, "box"], label="box")
sp.plot(df.index, df.loc[:, "tri"], label="tri")
sp.plot(df.index, df.loc[:, ("shal", "Lev")], label="shallow")
sp.legend()
sp.set_ylabel("Water Pressure (m of H2O)")
```

Más Ventana Rodante

```python
# continued from previous script
# rolling function using two columns (correlation)
A=df.loc[:,('deep','Lev')].rolling(10,center=True).corr(df.loc[:,('shal','Lev')])
df=df.assign(corr=A)

# using your own function, a rolling IQR
def iqr(x):
    value=np.quantile(x,.75)-np.quantile(x,.25)
    return value

A=df.loc[:,('deep','Lev')].rolling(10,center=True).agg(iqr)
df=df.assign(iqr=A)

fig, sp = pl.subplots(nrows=2,ncols=1, sharex=True)
sp[0].plot(df.index, df.loc[:, "box"], label="box")
sp[0].plot(df.index, df.loc[:, "tri"], label="tri")
sp[0].plot(df.index, df.loc[:, ("shal", "Lev")], label="shallow")
sp[0].plot(df.index, df.loc[:, ("deep", "Lev")], label="deep")
sp[0].legend()
sp[0].set_ylabel("Water Pressure (m of H2O)")

sp[1].plot(df.index, df.loc[:, 'iqr'],color='black')
sp[1].set_ylabel('interquartile range', color='black')

sp1tw=sp[1].twinx()
sp1tw.plot(df.index, df.loc[:, 'corr'],color='red')
sp1tw.set_ylabel('correlation',color='red')
sp1tw.yaxis.set_tick_params(labelcolor='red')
pl.show()
```

Ploteas elegantes con Seaborn

- `seaborn`: hace diferentes ploteas estadísticas
- aparece mejor? que matplotlib
- hecho para funcionar bien con pandas
- ploteas complicadas, hecho sencillas!

```python
import pandas as pd
import matplotlib.pyplot as pl
import seaborn as sns
# read in data and get into usable format , as done before
df = pd.read_table("./examples/USGS01037000.tab", comment="#")
col_dict = {}
for col in df.columns:
    if ("_00065" in col) and ("_cd" not in col):
        col_dict[col] = "gage_ht_ft"
    elif ("_00060" in col) and ("_cd" not in col):
        col_dict[col] = "Q_cfs"
df = df.rename(col_dict, axis=1)
df = df.loc[1:, ["datetime", "Q_cfs", "gage_ht_ft"]]
df.loc[:, "datetime"] = pd.to_datetime(df.loc[:, "datetime"])
df.loc[:, "gage_ht_ft"] = pd.to_numeric(df.loc[:, "gage_ht_ft"], errors="coerce")
df.loc[:, "Q_cfs"] = pd.to_numeric(df.loc[:, "Q_cfs"], errors="coerce")
df = df.assign(month=df.loc[:, "datetime"].dt.month)

sns.set_theme(style="darkgrid")
# making a nice looking regression plot
fig, sps = pl.subplots(ncols=1, nrows=2)
# can mix seaborn and matplotlib plots
sps[0].xaxis.update_units(df.datetime)
sps[0].semilogy(df.datetime.values, df.Q_cfs.values)
# seaborn regresion plot, display minor gridlines
sns.regplot( x=df.gage_ht_ft, y=df.Q_cfs, ax=sps[1], order=1, scatter_kws={"color": "black"}, line_kws={"color": "red"})
sps[0].grid(which="both", axis="y")
pl.show()
```

Dataframes con Mult-indices

- niveles diferentes con nombres de indices y columnas
- maneja mas que 2-D datos
- configurando unos indices de 'multi-level' indices y columna es diferente
    - sobrescribe el encabezamiento de columna
    - para indice, lista las columnas que quiere para 'multi-index'

```
import pandas as pd
import seaborn as sb
# increase number of visible rows
pd.options.display.min_rows = 30

# get one of the seaborn data sets, need internet connection
# list available data sets with 'sb.get_dataset_names()'
df = sb.load_dataset("penguins")
# drop rows with NaN values, get a list of unique island names
df = df.dropna(axis=0, how="any")
islands = df.island.unique()
# overwrite original index with multiindex created from columns of
       data
df2 = df.set_index(["island", "sex", "species"])
# extract data for each island, create multindex for each island, and
        concat
island_dict = {}
df3 = pd.DataFrame()
for isle in islands:
   island_dict[isle] = df2.loc(axis=0)[isle]
   # multi-index has to have the same number of columns as dataframe
       assigned to
   # there are several ways in pandas to create the multiindex, this
       is just one
   m_idx = pd.MultiIndex.from_product([[isle], island_dict[isle].
       columns])
   # overwrite original column with multilevel columns (multiindex)
   island_dict[isle].columns = m_idx
   df3 = pd.concat((df3, island_dict[isle].iloc[:, 0:2]))
df3 = df3.sort_index()
```

Agrupar y Agregar los Datos

- agrega los valores (sum, mean, variance, etc.)
- metido de `groupby`
    - agrupa valores juntos

```python
import pandas as pd
import numpy as np

# read in data and get into usable format
df = pd.read_table("./example/USGS01037000.tab", comment="#")

# rename columns
col_dict = {}
for col in df.columns:
    if ("_00065" in col) and ("_cd" not in col):
   col_dict[col] = "gage_ht_ft"
    elif ("_00060" in col) and ("_cd" not in col):
   col_dict[col] = "Q_cfs"
df = df.rename(col_dict, axis=1)

df = df.loc[1:, ["datetime", "Q_cfs", "gage_ht_ft"]]
df.loc[:, "datetime"] = pd.to_datetime(df.loc[:, "datetime"])
df.loc[:, "gage_ht_ft"] = pd.to_numeric(df.loc[:, "gage_ht_ft"],
       errors='coerce')
df.loc[:, "Q_cfs"] = pd.to_numeric(df.loc[:, "Q_cfs"], errors='
       coerce')

# calculate aggragate values for columns
mean = df.mean()
median = df.median()
var = df.var()
# df.describe() for short table of summary stats
```

Más Agrupar y Agregar los Datos

- usa **assign** para hacer columnas nuevas 'para agrupar'
- puede encadenar metidos **groupby** y **aggregate**
- puede hacer funciones personalizadas para agregar

```
#### CONTINUED FROM PREVIOUS CODE BLOCK ####
# Grouping data
# calculate monthly statistics:
# make column of months
df = df.assign(month=df.loc[:, "datetime"].dt.month)
# group by month
# splitting data, applying function, combinging into dataframe
g_month = df.groupby("month")
month_mean = g_month.agg("mean")
# or chain these together
# monthly_mean = df.groupby('month').agg('mean')

# calculate weekly stats
# careful with isocalander, all weeks start on monday,
# Dec 31 could be in the first week of the next year
df = df.assign(week=df.loc[:, "datetime"].dt.isocalendar()["week"])
week_stats = df.groupby("week")["Q_cfs", "gage_ht_ft"].agg([min, np.
        median, max])

# making a custom function
def my_slope(series):
    slope, inter = np.polyfit(series.index, series.values, 1)
    return slope

week_stats2 = df.groupby("week")["Q_cfs","gage_ht_ft"].agg(my_slope)
```

Replantear, Desremar y Rebanar

- `unstack`: mueve una porción del indices al columnas
- `stack`: mueve una porción de columnas al indices
- usa `level` palabra clave para indicar niveles pare mover
  - nivel cero es más externa de multi-index

```python
import pandas as pd
# increase number of visible rows
pd.options.display.min_rows = 30
# get one of the seaborn data sets
df = sb.load_dataset("penguins")
# drop rows with NaN values, change to multiindex
df = df.dropna(axis=0, how="any")
df1 = df.set_index(["species", "sex", "island"])
# multiindex dataframes perform better if sorted
df1 = df1.sort_index()
# get into a series format
df2 = df1.stack()
# Can't unstack df2 because has duplicate values, can't set columns to same value
# to get around this, include original index in reindexed dataframe
df3 = df.set_index([df.index, "species", "sex", "island"])
df3 = df3.sort_index()
# can control level(s) unstacked and moved to column heading with keyword: level=2 or level=[1,2]
df4 = df3.unstack()
df5 = df4.stack()
## slicing the dataframe
A = df3.loc[(0, "Adelie", "Male", "Torgersen"):(19, "Adelie", "Male", "Torgersen"),
    "bill_length_mm",]
## if duplicate values in dateframe, can specify axis to get all duplicated values
B = df2.loc["Adelie"]
C = df2.loc(axis=0)["Adelie", "Male"]
# also need to specify axis to be able slice on 2nd variable
D = df1.loc(axis=0)[:, "Male"]
E = df1.loc(axis=0)[:, "Male"]["body_mass_g"]
```

Swaplevel y Reorderlevel

- para cambiar la organización de niveles de multi-index
  - `swaplevel` intercambia dos niveles
  - `reorderlevel` intercambia más que dos niveles
    - lista niveles en orden que quiere
    - `axis` palabra clave para indicar indices de filas o columnas
    - puede ordenar indices de columna con `sort_index(axis=1)`

```
import pandas as pd
import seaborn as sb

# increase number of visible rows
pd.options.display.min_rows = 30
# get one of the seaborn data sets
df1 = pd.read_hdf('./data/penguins.hdf')
# drop rows with NaN values, change to multiindex
df2 = df1.set_index([df1.index, "species", "sex", "island"])
df2 = df2.sort_index()
# can control level(s) that is unstacked and moved to column heading with the keyword
# level level=2 or level=[1,2]
df2 = df2.unstack()
# commands to change ordering in indices
df2.swaplevel(0,2)
df2.swaplevel(axis=1)
df2.swaplevel(axis=1).sort_index(axis=1)
#note extra parentheses
df2.reorder_levels((1,0,2))
```

Xs, Pivot y Pivot_table

- sección transversal (`xs`): subconjunto de multi-index
  - solo función con multi-index
- `pivot`: remodelar dataframe
  - requiere valores únicos
  - todos los valores vuelven
- `pivot_table`:
  - remodelar dataframe
  - agrega valores duplicados

```
import pandas as pd
import seaborn as sb
import numpy as np

df0 = sb.load_dataset("penguins")
df0 = df0.dropna()
df1 = df0.set_index([df0.index, "sex", "species"])
df1 = df1.sort_index()
# xs only works on a multiindex, selects subset of values
df1.xs("Gentoo", level=2)
df1.xs(["Female", "Gentoo"], level=[1, 2])

# reshape df, pivot requires no duplicate values with no aggregation
# pivot_table will aggregate duplicate data (dafaults to mean)
df1.pivot_table(values=["bill_length_mm", "bill_depth_mm"], index=["sex"])
df1.pivot_table(
    values=["bill_length_mm", "bill_depth_mm"], index=["sex", "island"], aggfunc=np.std
)
# dummy data to show pivot method
df2 = pd.DataFrame(
    {
        "Na": np.random.uniform(0, 1, 10),
        "Ca": np.random.uniform(0, 1, 10),
        "K": np.random.uniform(0, 1, 10),
        "Mg": np.random.uniform(0, 1, 10),
        "loc": [1, 2, 3, 4, 5] * 2,
        "depth": [0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
    }
)
df2.pivot(index="loc", columns="depth", values=["K", "Na"])
```