A Python notebook for extracting and manipulating Macrostrat data

Author

- Author1 = {"name": "Shan Ye", "affiliation": "University of Wisconsin-Madison", "email": "shan.ye@wisc.edu", "orcid": ""}

Purpose

This Python Jupyter notebook is built as a demo to illustrate how to extract column data through the Macrostrat API and how to process and visualize them. It contains functions for data wrangling, tabulating, bootstrapping sampling, and visualization. By providing useful examples and Python scripts, this notebook could promote the Macrostrat database in the geology and paleontology communities.

Technical contributions

- building time series of spatial coverages of marine and non-marine sediments in Phanerozoic North America,
- building time series of spatial coverages of siliciclastic and carbonate lithologies throughout the Mesozoic North America
- bootstrapping samplings and visualizations of these time series

Methodology

NA

Results

This notebook provides some Python codes to retrieve and manipulate data from Macrostrat, especially to remove over-counted columns (due to their multiple environmental and/or lithological attributes within a single time step) when constructing the time series. There are no scientific results. Demo results are shown in the code section.

Funding

NA

Keywords

keywords=["Macrostrat", "time-series", "stratigraphy", "lithology", "paleoenvironment"]

Citation

Shan Ye, 2021, A Python notebook for extracting and manipulating Macrostrat data. Accessed at https://github.com/yeshancqcq/earthcube_notebook

Work In Progress - improvements

- Retrieving and visualizing geological map data (waiting for the finalization of the new Macrostrat API routes)
- Retrieve data from other geoscience databases (like the PBDB) and perform time series statistics (like Pearson's r) between data from different sources

Suggested next steps

- Test on extreme user inputs and add error proofings to functions.
- Possibly create a web-based GUI for geoscientists who do not write codes but still want to use Macrostrat.

Acknowledgements

The Macrostrat group at the University of Wisconsin-Madison, supervised by Dr. Shanan Peters.

Setup

Library import

Import all the required Python libraries.

The code cell below is an example.

In [2]:
```python
import json
import requests
import re
import builtins
import pandas as pd
import numpy as np
from random import choices
from collections import Counter
import matplotlib
import matplotlib.pyplot as plt
from IPython.display import clear_output
import geopandas as gpd
import plotly
```

Parameter definitions

NA

Data import

Data are retrieved from the Macrostrat database via its api (https://macrostrat.org/#api).

Data processing and analysis

The core of the notebook is here. Split this section into subsections as required, and explain processing and analysis steps.

Function 1: get data from the Macrostrat API and turn it into the json format

There are different levels of the lithological classification.

Regular expression is used to detect the level of lithology requested by the user.

This function returns 3 things in the following order:

- the json file
- the detected level of lithology
- the detected lithological name

These results will be used in the next function.

In [3]:
```python
def get_json(api):
    j = requests.get(api)
    l_level = "all"
    l_name = "any"

    if "lith_type" in api:
        x = re.search("(?<=lith_type=)[a-zA-Z]*", api)
        l_level = "lith_type"
        l_name = x.group()
    elif "lith=" in api:
        x = re.search("(?<=lith=)[a-zA-Z]*", api)
        l_level = "lith_name"
        l_name = x.group()
    elif "lith_class" in api:
        x = re.search("(?<=lith_class=)[a-zA-Z]*", api)
        l_level = "lith_class"
        l_name = x.group()
    return j.json(), l_level, l_name
```

Function 2: Process the returned json file and extract relevant column data

The Function 1 is embedded within this function, so users do not need to call the api twice.

This function returns a dataframe containing raw column data.

This dataframe will be used in the next function.

In [4]:
```python
def col_json_proc(api):

    print("Reading api...")
    temp, l_level, l_name = get_json(api)
    df = pd.DataFrame()
    nrows = []

    for i in range(len(temp['success']['data'])):

        clear_output(wait = True)

        new_line = temp['success']['data'][i]['lith']
        nrow = len(temp['success']['data'][i]['lith'])
        df = df.append(pd.DataFrame(data = new_line))
        nrows.append(nrow)
        print('Extracting data from api...',np.round(i/len(temp['success']['data'])*100,1),
'%')

    all_unit_ids = []
    for i in range(len(temp['success']['data'])):
        all_unit_ids.append(temp['success']['data'][i]['unit_id'])

    s = np.array(all_unit_ids)
    df['unit_id'] = list(np.repeat(s, nrows, axis=0))
    df = df.sort_values('unit_id')

    clear_output(wait = True)
    print("Processing lithology...")
    l = []
    if l_level == "lith_type":
        for i in range(len(df)):
            if df["type"].iloc[i] == l_name:
                l.append(i)
    elif l_level == "lith_class":
        for i in range(len(df)):
            if df["class"].iloc[i] == l_name:
                l.append(i)
    elif l_level == "lith_name":
        for i in range(len(df)):
            if df["name"].iloc[i] == l_name:
                l.append(i)
    else:
        for i in range(len(df)):
            l.append(i)

    df_sub = df.iloc[l]

    clear_output(wait = True)
    print("Processing prop...")
    prop_pivot = list(df_sub.pivot_table( columns='unit_id', values='prop', aggfunc='sum').i
loc[0])
    for i in range(len(prop_pivot)):
        if prop_pivot[i] > 1:
            prop_pivot[i] = 1

    clear_output(wait = True)
    print("Assembling the data frame...")
    all_col_ids = []
    all_unit_ids = []
    all_t_age = []
    all_b_age = []
    all_max_thick = []
    all_min_thick = []
    all_pbdb_collections = []
    all_col_area = []
    for i in range(len(temp['success']['data'])):
        all_col_ids.append(temp['success']['data'][i]['col_id'])
        all_unit_ids.append(temp['success']['data'][i]['unit_id'])
        all_t_age.append(temp['success']['data'][i]['t_age'])
        all_b_age.append(temp['success']['data'][i]['b_age'])
        all_max_thick.append(temp['success']['data'][i]['max_thick'])
        all_min_thick.append(temp['success']['data'][i]['min_thick'])
        all_pbdb_collections.append(temp['success']['data'][i]['pbdb_collections'])
        all_col_area.append(temp['success']['data'][i]['col_area'])

    df2 = pd.DataFrame()
    df2['col_id'] = all_col_ids
    df2['unit_id'] = all_unit_ids
    df2['t_age'] = all_t_age
    df2['b_age'] = all_b_age
    df2['max_thick'] = all_max_thick
    df2['min_thick '] = all_min_thick
    df2['pbdb_collections'] = all_pbdb_collections
    df2['col_area'] = all_col_area
    df2 = df2.sort_values('unit_id')
    df2['prop'] = prop_pivot

    return df2
```

Function 3: Converting the raw column data to the time series data, with bootstrapping sampling

This function will be directly used by the user.

It has 2 arguments:

- the raw column data returned from the Function 2
- the number of iterations for the bootstrapping sampling

It will return a dataframe the following fields for each of the 1 million year time step from 540 Ma to the present:

- the mean number of columns
- the standard deviation of the number of columns
- the mean number of stratigraphic units
- the standard deviation of the number of stratigraphic
- the mean column area
- the standard deviation of column area

In [5]:
```python
def get_time_series(df2, bs):
    steps = list(range(3000))
    t = 540
    n = len(df2)
    sample = list(range(n))

    res = np.ndarray(shape=(bs,t,3), dtype=float)

    for b in range(bs):
        print("Processing boostrapping reps:", b+1, "out of", bs)
        if b > 1:
            sample = np.random.choice(list(range(n)), n, replace=True)
        packages = df2.iloc[sample]

        total_cols,total_packages,total_area = [],[],[]

        for i in range(t):
            cand = []
            for j in range(len(packages)):
                if packages['b_age'].iloc[j] >= steps[i]+1 and packages['t_age'].iloc[j] < s
teps[i]+1:
                    cand.append(j)
            col_cand = packages['col_id'].iloc[cand]
            total_cols.append(len(set(col_cand)))
            total_packages.append(len(cand))

            if total_cols[i] > 1:
                freq = Counter(col_cand)
                df_col = pd.DataFrame(freq.keys(),columns=['col_id'])
                df_col['Freq'] =freq.values()
                df_col.rename(columns={'A':'col_id','B':'Freq'})
                df_col_cand = pd.DataFrame(col_cand)
                col_temp = df_col.merge(df_col_cand, left_on='col_id', right_on='col_id')
                cols = col_temp['Freq']
            else:
                cols = [1] * len(cand)

            t_area = 0
            for j in range(len(packages['col_area'].iloc[cand])):
                if total_cols[i] > 1:
                    t_area += (packages['col_area'].iloc[cand].iloc[j]/col_temp['Freq'].iloc
[j])
                else:
                    t_area += (packages['col_area'].iloc[cand].iloc[j])

            total_area.append(t_area)

            clear_output(wait = True)
            print("boostrapping reps:", b+1,'...', np.round(i/t*100,1),'%')

        np1 = np.array(total_cols)
        np2 = np.array(total_packages)
        np3 = np.array(total_area)

        stack = np.stack((np1, np2, np3), axis=1)

        res[b] = stack

    clear_output(wait = True)
    print('calculating mean and std...')
    mean_all = np.mean(res, axis=0)
    std_all = np.std(res, axis=0)

    clear_output(wait = True)
    print('finalizing the result...')
    col_mean, col_std, package_mean, package_std, area_mean, area_std = [],[],[],[],[],[]
    for i in range(len(mean_all)):
        col_mean.append(mean_all[i][0])
        package_mean.append(mean_all[i][1])
        area_mean.append(mean_all[i][2])
        col_std.append(std_all[i][0])
        package_std.append(std_all[i][1])
        area_std.append(std_all[i][2])

    res_ts = pd.DataFrame()
    res_ts['time'] = range(1,541)
    res_ts['col_mean'] = col_mean
    res_ts['col_std'] = col_std
    res_ts['package_mean'] = package_mean
    res_ts['package_std'] = package_std
    res_ts['area_mean'] = area_mean
    res_ts['area_std'] = area_std

    clear_output(wait = True)
    return res_ts
```

Function 4: Creating a map of a specific time step

This function is to plot a map showing the spatial coverage of the specific lithology or environment at the given time.

This function has 3 arguments:

- The API route
- The time (in Ma)
- The color for columns with positive values (optional; the default is brown color)

In [6]:
```python
def plot_col(api, age, color = "brown"):
    df = col_json_proc(api)
    clear_output(wait = True)
    col = gpd.read_file('data/col.geojson')
    col['color'] = 0
    selected_col = []
    for row in df.iterrows():
        if row[1]['t_age'] < age and row[1]['b_age'] >= age:
            if int(row[1]['col_id']) not in selected_col:
                selected_col.append(int(row[1]['col_id']))
    for i in range(len(col)):
        if int(col.iloc[i,0]) in selected_col:
            col.iloc[i,4] = 1
    cmap = matplotlib.colors.ListedColormap(['#B3B3B3', color])
    boundaries = [-1,0,1]
    norm = matplotlib.colors.BoundaryNorm(boundaries, cmap.N, clip=True)
    print("Processing data...")
    clear_output(wait = True)
    f, ax = plt.subplots(1, figsize=(14, 10))
    ax = col.plot(column ='color',cmap = cmap ,ax=ax,linestyle='solid')
    plt.show()
    return
```

Examples

1. Processing and visualizing time series of marine and non-marine paleoenvironmental coverages in Cretaceous

Macrostrat database is pretty big. Depending on your computer's capability and the specific variable you request in the API, the bootstrapping sampling could take several minutes.

In [7]:
```python
# Retriving the non-marine data via the Macrostrat API
api_non_marine = 'https://macrostrat.org/api/units?environ_class=non-marine&project_id=1&res
ponse=long'
# Processing the API and get the raw column data
df_non_marine = col_json_proc(api_non_marine)
# Converting the raw column data into the time series data, with 5 iterations for bootstrapp
ing
ts_non_marine = get_time_series(df_non_marine, 5)
# Preview the time series data
ts_non_marine
```

Out[7]:

	time	col_mean	col_std	package_mean	package_std	area_mean	area_std
0	1	356.4	77.306145	485.4	7.939773	8.674132e+06	1.917438e+06
1	2	324.2	69.332244	434.8	10.851728	7.438156e+06	1.721470e+06
2	3	115.2	25.245990	156.6	5.571355	2.300513e+06	5.636276e+05
3	4	126.4	29.513387	166.6	8.138796	2.726739e+06	7.441721e+05
4	5	112.0	25.392912	149.8	6.852737	2.485676e+06	6.493807e+05
...	...	...	...	...	...	...	...
535	536	0.8	0.400000	0.8	0.400000	4.481883e+04	2.240941e+04
536	537	0.8	0.400000	0.8	0.400000	4.481883e+04	2.240941e+04
537	538	2.4	0.489898	2.8	0.400000	5.086686e+04	2.172516e+04
538	539	2.4	0.489898	2.8	0.400000	5.086686e+04	2.172516e+04
539	540	2.4	0.489898	2.8	0.400000	5.086686e+04	2.172516e+04

540 rows × 7 columns

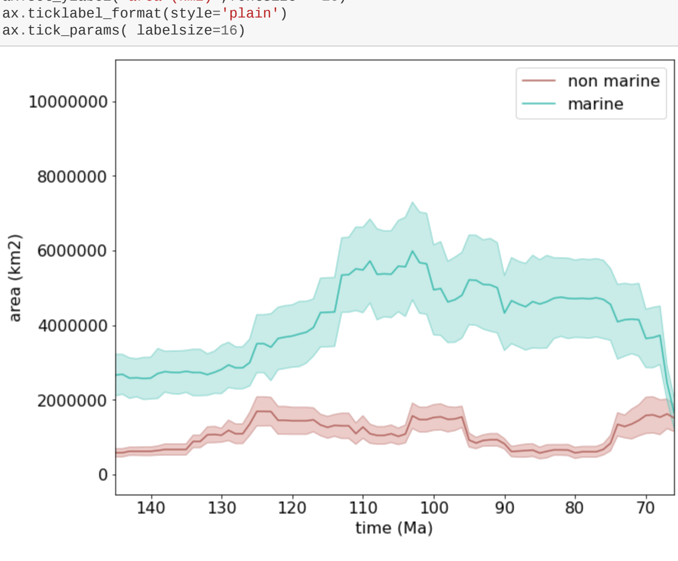
In [8]:
```python
# Retriving the marine data via the Macrostrat API
api_marine = 'https://macrostrat.org/api/units?environ_class=marine&project_id=1&response=lo
ng'
# Processing the API and get the raw column data
df_marine = col_json_proc(api_marine)
# Converting the raw column data into the time series data, with 5 iterations for bootstrapp
ing
ts_marine = get_time_series(df_marine, 5)
# Preview the time series data
ts_marine
```

Out[8]:

	time	col_mean	col_std	package_mean	package_std	area_mean	area_std
0	1	46.8	11.124747	62.6	4.454211	1.056107e+06	286446.908762
1	2	63.8	14.246403	86.0	5.253570	1.290220e+06	313506.674752
2	3	58.2	12.155657	78.4	4.176123	1.096084e+06	353161.905305
3	4	73.0	17.366635	97.8	6.112283	1.262094e+06	337212.367144
4	5	73.0	17.227884	98.6	3.773592	1.541647e+06	333918.612819
...	...	...	...	...	...	...	...
535	536	87.2	20.023986	119.8	9.987993	1.438988e+06	418616.544025
536	537	83.6	20.224737	113.4	9.178235	1.360760e+06	416632.913117
537	538	77.4	18.693314	106.8	6.337192	1.252502e+06	372771.486354
538	539	77.8	18.334667	107.2	5.114685	1.266707e+06	361419.781189
539	540	77.8	18.334667	107.2	5.114685	1.266707e+06	361419.781189

540 rows × 7 columns

In [9]:
```python
# Plot the time series data for the spatial coverage of both marine and non-marine environme
nt
fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(ts_non_marine['time'], ts_non_marine['area_mean'], label = 'non marine',color='#b366
60')
ax.plot(ts_marine['time'], ts_marine['area_mean'], label = 'marine',color='#40bdb0')
ax.fill_between(ts_non_marine['time'], ts_non_marine['area_mean']-ts_non_marine['area_std'],
ts_non_marine['area_mean']+ts_non_marine['area_std'],
    alpha=0.5, edgecolor='#b36660', facecolor='#db9a95')
ax.fill_between(ts_marine['time'], ts_marine['area_mean']-ts_marine['area_std'], ts_marine[
'area_mean']+ts_marine['area_std'],
    alpha=0.5, edgecolor='#40bdb0', facecolor='#95dbd4')
ax.set_xlim(145,66)
ax.legend(fontsize = 16)
ax.set_xlabel("time (Ma)",fontsize = 16)
ax.set_ylabel("area (km2)",fontsize = 16)
ax.ticklabel_format(style='plain')
ax.tick_params( labelsize=16)
```



2. Processing and visualizing time series of siliciclastic and carbonate coverages in Phanerozoic

Macrostrat database is pretty big. Depending on your computer's capability and the specific variable you request in the API, the bootstrapping sampling could take several minutes.

In [10]:
```python
# Retriving the siliciclastic data via the Macrostrat API
api_sil = 'https://macrostrat.org/api/units?lith_type=siliciclastic&project_id=1&response=lo
ng'
# Processing the API and get the raw column data
df_sil = col_json_proc(api_sil)
# Converting the raw column data into the time series data, with 5 iterations for bootstrapp
ing
ts_sil = get_time_series(df_sil, 5)
# Preview the time series data
ts_sil
```

Out[10]:

	time	col_mean	col_std	package_mean	package_std	area_mean	area_std
0	1	377.4	85.614485	517.2	13.044539	9.169546e+06	2.075346e+06
1	2	356.6	80.911309	492.6	18.216476	8.302722e+06	1.825732e+06
2	3	144.8	33.623801	196.6	9.748846	3.002594e+06	7.205028e+05
3	4	170.0	40.059955	233.0	12.132601	3.442324e+06	8.412622e+05
4	5	159.4	35.802793	221.4	13.062925	3.471035e+06	8.164358e+05
...	...	...	...	...	...	...	...
535	536	62.8	14.218298	86.2	1.833030	9.988984e+05	2.690538e+05
536	537	61.4	13.690873	83.4	2.576820	9.870236e+05	2.546009e+05
537	538	58.0	14.113823	80.2	2.227106	8.911496e+05	2.644663e+05
538	539	56.8	14.274453	79.0	3.286335	8.852188e+05	2.647451e+05
539	540	56.8	14.274453	79.0	3.286335	8.852188e+05	2.647451e+05

540 rows × 7 columns

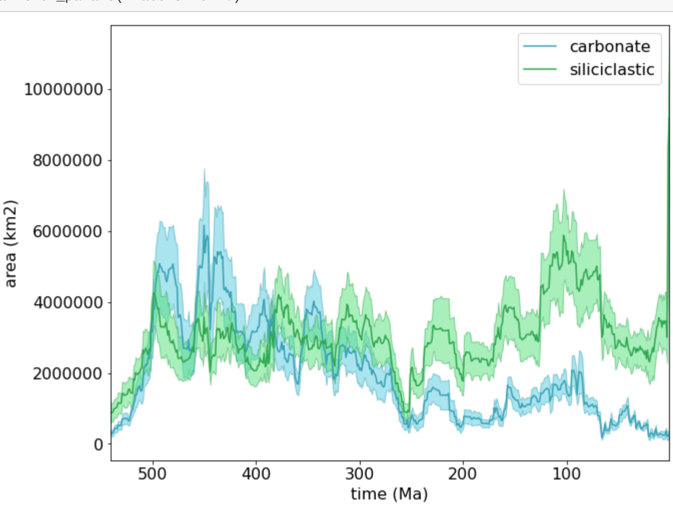
In [11]:
```python
# Retriving the carbonate data via the Macrostrat API
api_carb = 'https://macrostrat.org/api/units?lith_type=carbonate&project_id=1&response=long'
# Processing the API and get the raw column data
df_carb = col_json_proc(api_carb)
# Converting the raw column data into the time series data, with 5 iterations for bootstrapp
ing
ts_carb = get_time_series(df_carb, 5)
# Preview the time series data
ts_carb
```

Out[11]:

	time	col_mean	col_std	package_mean	package_std	area_mean	area_std
0	1	6.4	2.870540	9.4	3.382307	220259.414667	103350.235877
1	2	7.0	2.756810	10.0	0.632456	255688.481167	113511.637960
2	3	5.2	3.187475	6.6	2.332381	213648.496600	116031.082841
3	4	12.0	3.521363	16.2	2.400000	363114.364367	93117.889238
4	5	8.2	2.481935	11.4	2.939388	258648.677000	113385.923366
...	...	...	...	...	...	...	...
535	536	26.0	5.932959	34.0	4.857983	435041.073833	109938.841951
536	537	24.4	5.851496	31.8	5.192302	389713.932817	106125.729813
537	538	20.4	4.923413	26.6	3.555278	305567.231850	87783.986111
538	539	20.0	4.289522	26.2	2.925748	306796.149100	85508.550340
539	540	20.0	4.289522	26.2	2.925748	306796.149100	85508.550340

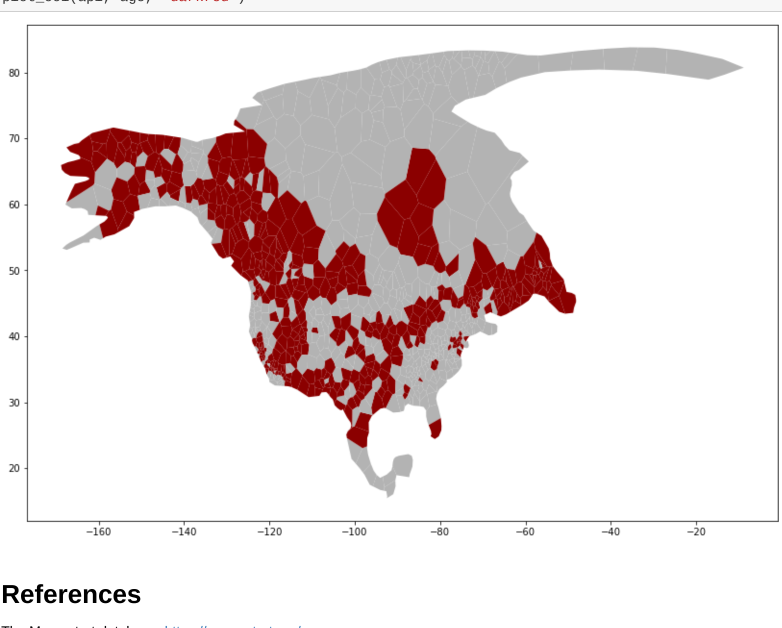
540 rows × 7 columns

In [12]:
```python
# Plot the time series data for the spatial coverage of both siliciclastic and carbonate
fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(ts_carb['time'], ts_carb['area_mean'], label = 'carbonate',color='#3ea3b5')
ax.plot(ts_sil['time'], ts_sil['area_mean'], label = 'siliciclastic',color='#32a852')
ax.fill_between(ts_carb['time'], ts_carb['area_mean']-ts_carb['area_std'], ts_carb['area_mea
n']+ts_carb['area_std'],
    alpha=0.5, edgecolor='#3ea3b5', facecolor='#55cbe0')
ax.fill_between(ts_sil['time'], ts_sil['area_mean']-ts_sil['area_std'], ts_sil['area_mean']+
ts_sil['area_std'],
    alpha=0.5, edgecolor='#32a852', facecolor='#55e07b')
ax.set_xlim(540,1)
ax.legend(fontsize = 16)
ax.set_xlabel("time (Ma)",fontsize = 16)
ax.set_ylabel("area (km2)",fontsize = 16)
ax.ticklabel_format(style='plain')
ax.tick_params( labelsize=16)
```



3. Mapping the spatial coverage of non-marine environment at 1 Ma

In [13]:
```python
api = 'https://macrostrat.org/api/units?environ_class=non-marine&project_id=1&response=long'
age = 1
plot_col(api, age, "darkred")
```



References

The Macrostrat database: https://macrostrat.org/

Peters, Shanan E., Jon M. Husson, and John Czaplewski. "Macrostrat: a platform for geological data integration and deep-time Earth crust research." Geochemistry, Geophysics, Geosystems 19.4 (2018): 1393-1409.