記事
Toshihiko Minamoto · 2021年9月23日 10m read

データジャングルの視覚化 -- パート1: グラフを作成しよう

この記事は、視覚化ツールと時系列データの分析を説明する連載の最初の記事です。
当然ながら、Caché製品ファミリーから収集できるパフォーマンス関連のデータを見ることに焦点を当てますが、説明の途中で、他の内容についても解説していきます。
まずは、Pythonとそのエコシステムで提供されているライブラリ/ツールを探りましょう。

この連載は、Murrayが投稿
したCachéのパフォーマンスと監視に関する優れた連載
（こちらから参照）、より具体的にはこちらの記事と密接に関係しています。

免責事項1: 確認しているデータの解釈について話すつもりですが、それを詳しく話すと実際の目標から外れてしまう可能性があります。
そのため、Murrayの連載を先に読んで、主題の基本的な理解を得ておくことを強くお勧めします。

免責事項2: 収集したデータを視覚化するために使用できるツールは山ほどあります。
その多くは、mgstatなどから得たデータを直接処理するが、必要最低限の調整だけで処理することができます。
この記事は「このソリューションがベストですよ」という投稿ではまったくなく、
あくまでも、「データを操作する上で便利で効果的な方法を見つけたよ」という記事です。

免責事項3: データの視覚化と分析は、詳しく見るほど非常にやみつきになる、刺激的で楽しい分野です。
これに没頭して自由な時間を失ってしまう可能性があります。 でも、警告しましたからね！

では、前置きはこれくらいにして、早速内容を見ていきましょう。

前提条件

始める前に、次のツールとライブラリを用意してください。
 * Jupyterノートブック
 * Python （3）
 * 作業中に使用するさまざまなPythonライブラリ

Python（3）：マシンに Python が必要です。 アーキテクチャごとにインストールの方法が様々です。
私はmacでhomebrewを使用しており、これを使って簡単にインストールできました。

```
brew install python3
```

Googleであなたのお気に入りのプラットフォームに使用できる手順を検索してください。

Jupyterノートブック:
厳密には必要ではありませんが、Jupyterノートブックがあれば、Pythonスクリプトでの作業が楽になります。
ブラウザウィンドウからインタラクティブにPythonスクリプトを実行して表示することができます。
また、スクリプトでの共同作業も可能です。
コードの実験と操作を非常に簡単に行えるようになるため、強くお勧めします。

```
pip3 install jupyter
```

（繰り返しになりますが、$search（検索）エンジンに尋ねてください）

Pythonライブラリ Pythonライブラリについては使用しながら説明します。 **import**
ステートメントでエラーが発生している場合は、まずは、ライブラリがインストールされていることを確認してください。

```
pip3 install matplotlib
```

はじめに

マシンにすべてがインストールされていれば、ライブラリから

```
jupyter notebook
```

を実行できるはずです。
このコードを実行すると自動的にブラウザウィンドウが開き、単純なUIが表示されます。 

早速メニューから新しいノートブックを作成し、最初のコードセルにインポートステートメントをいくつか追加します（新規 -> ノートブック -> Python3）。

```
import math
import pandas as pd
import mpl_toolkits.axisartist as AA
from mpl_toolkits.axes_grid1 import host_subplot
import matplotlib.pyplot as plt
from datetime import datetime
from matplotlib.dates import DateFormatter
```

インポートしているライブラリについて、いくつかのコメントがあります。
 * Pandas これは、Pythonプログラミング言語向けの高性能で使いやすいデータ構造とデータ解析ツールを提供するオープンソースのBSDラインセンスライブラリです。
これにより、ビッグデータセットを効率的に処理することができます。
pButtonsから取得するデータセットは、決して「ビッグデータ」ではありませんが、
一度にたくさんのデータを確認できるので安心です。
過去20年に渡って、24時間/2秒サンプリングでpButtonsを収集していたとしても、
それをグラフ化できるということです。
 * Matplotlib mataplotlibは、さまざまなハードコピー形式やプラットフォーム間でのインタラクティブ環境で出版グレードの図を生成するPythonの2Dグラフ作成ライブラリです。
この記事のメインのグラフ作成エンジンとしてとりあえず使用するのがこのライブラリです。

現在のコードセルの実行（ショートカット:
Ctrl+Enter）（ショートカット一覧
）でエラーが発生している場合は、上記のライブラリがインストールされていることを確認してください。

また、*Untitled*
ノートブックの名前も変更していることに気づくでしょう。名前を変更するには、タイトルをクリックしてください。

データの読み込み

基礎工事が終了したので、データをいくらか読み込むことにしましょう。
幸い、PandasにはCSVデータを簡単に読み込む方法があります。
都合よくcsv形式のmgstatデータのセットが手元にあるため、それを使うことにします。

```
mgstatfile = '/Users/kazamatzuri/work/proj/vis-articles/part1/mgstat.txt'
data = pd.read_csv(
    mgstatfile,
    header=1,
    parse_dates=[[0,1]]
   )
```

readcsv コマンドを使用して、mgstatデータを直接DataFrameに読み取っています。
オプションに関する総合的な概要は、ドキュメント全文をご覧ください。 つまり、読み取るファイルを渡して、2行目（1行目は0です！）にヘッダー名が含まれていることを伝えているだけです。 mgstatは日付と時刻のフィールドを2つのフィールドに分割するため、parse_datesパラメーターでそれらのフィールドを組み合わせることも必要です。

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25635 entries, 0 to 25634
Data columns (total 37 columns):
Date_       Time        25635 non-null datetime64[ns]
  Glorefs               25635 non-null int64
 RemGrefs               25635 non-null int64
 GRratio                25635 non-null int64
  PhyRds                25635 non-null int64
 Rdratio                25635 non-null float64
 Gloupds                25635 non-null int64
 RemGupds               25635 non-null int64
 Rourefs                25635 non-null int64
 RemRrefs               25635 non-null int64
  RouLaS                25635 non-null int64
 RemRLaS                25635 non-null int64
  PhyWrs                25635 non-null int64
   WDQsz                25635 non-null int64
  WDtmpq                25635 non-null int64
 WDphase                25635 non-null int64
  WIJwri                25635 non-null int64
  RouCMs                25635 non-null int64
 Jrnwrts                25635 non-null int64
   GblSz                25635 non-null int64
 pGblNsz                25635 non-null int64
 pGblAsz                25635 non-null float64
   ObjSz                25635 non-null int64
 pObjNsz                25635 non-null int64
 pObjAsz                25635 non-null int64
   BDBSz                25635 non-null int64
 pBDBNsz                25635 non-null int64
 pBDBAsz                25635 non-null float64
  ActECP                25635 non-null int64
  Addblk                25635 non-null int64
 PrgBufL                25635 non-null int64
 PrgSrvR                25635 non-null int64
  BytSnt                25635 non-null int64
```

```
  BytRcd                25635 non-null int64
  WDpass                25635 non-null int64
  IJUcnt                25635 non-null int64
 IJULock                25635 non-null int64
dtypes: datetime64[ns](1), float64(3), int64(33)
memory usage: 7.2 MB
```

上記は、収集されたDataFrameの概要を示します。

データの操作

一部のフィールド名にはスペースが含まれており、「Date_
Time」は扱いにくいため、文字列を削除して最初の列の名前を変更します。

```
data.columns=data.columns.str.strip()
data=data.rename(columns={'Date_       Time':'DateTime'})
```

DataFrameはデフォルトでRangeIndexになります。 これはこのデータを確認するにはあまり役に立ちません。
より実用的なDateTime列を使用できるため、それをインデックスとして設定することにします。

```
data.index=data.DateTime
```

これで、最初のバージョンのプロットを作成できるようになりました。
最初に確認するものの1つは必ずGlorefsであるため、Glorefsを使用してみましょう。

```
plt.figure(num=None, figsize=(16,5), dpi=80, facecolor='w', edgecolor='k')
plt.xticks(rotation=70)
plt.plot(data.DateTime,data.Glorefs)
plt.show()
```

まず、ライブラリにグラフのサイズを指示しています。
また、x軸のラベルを少し回転させて、重なり合わないようにしています。
最後に、DateTimeとGlorefsのプロットを作成し、グラフを表示しています。
これで、次のようなグラフが出来上がります。

Glorefsをほかの列に置き換えるだけで、何が起きているのかを大まかに捉えることができます。

グラフの合成

複数のグラフをまとめて表示すると非常に便利な場合があります。
そのため、当然、複数のプロットを1つのグラフに描こうと考えるでしょう。 matplotlib
だけでこれを行うのは非常に簡単です。

```
plt.plot(data.DateTime,data.Glorefs)
plt.plot(data.DateTime,data.PhyRds)
plt.show()
```

ところが、これでは前とほぼ同じグラフが作成されてしまいます。 問題はy軸です。 Glorefsは数百万に達するのに対し、PhyRdsはたいてい数百（から数千）であるため、PhyRdsはほぼ読み取れません。

これを解決するには、前にインポートしたaxisartistツールキットを使用する必要があります。

```
plt.gcf()
plt.figure(num=None, figsize=(16,5), dpi=80, facecolor='w', edgecolor='k')
host = host_subplot(111, axes_class=AA.Axes)
plt.subplots_adjust(right=0.75)

par1 = host.twinx()
par2 = host.twinx()
offset = 60
new_fixed_axis = par2.get_grid_helper().new_fixed_axis
par2.axis["right"] = new_fixed_axis(loc="right",axes=par2,offset=(offset, 0))
par2.axis["right"].toggle(all=True)

host.set_xlabel("time")
host.set_ylabel("Glorefs")
par1.set_ylabel("Rdratio")
par2.set_ylabel("PhyRds")

p1,=host.plot(data.Glorefs,label="Glorefs")
p2,=par1.plot(data.Rdratio,label="Rdratio")
p3,=par2.plot(data.PhyRds,label="PhyRds")

host.legend()

host.axis["left"].label.set_color(p1.get_color())
par1.axis["right"].label.set_color(p2.get_color())
par2.axis["right"].label.set_color(p3.get_color())

plt.draw()
plt.show()
```

簡単に要約すると、2つのy軸をプロットに追加し、それぞれに異なる尺度を設定するということになります。 最初の例では暗黙的にサブプロットを使用しましたが、この場合は、直接それにアクセスして、軸とラベルを追加する必要があるため、 2つのラベルと色を設定しています。
凡例を追加して、様々なプロットに色を繋げたら、次のような画像になります。

最後に

これでデータをプロットするための非常に強力なツールを手に入れました。
mgstatデータを読み込んでいくつかの基本的なプロットを作成する方法を探りました。
次のパートでは、ほかのグラフの出力形式を試し、さらに多くのデータを取得することにします。

コメントやご質問をお受け付けしております！ あなたの体験をシェアしてください！

-Fab

追伸: これに使用するノートブックはこちらにあります。

#Python #オブジェクトデータモデル #ツール #ビッグデータ #視覚化 #Caché

ソースURL:
https://jp.community.intersystems.com/post/%E3%83%87%E3%83%BC%E3%82%BF%E3%82%B8%E3%83%A3%E3%83%B3%E3%82%B0%E3%83%AB%E3%81%AE%E8%A6%96%E8%A6%9A%E5%8C%96-%E3%83%91%E3%83%BC%E3%83%881-%E3%82%B0%E3%83%A9%E3%83%95%E3%82%92%E4%BD%9C%E6%88%90%E3%81%97%E3%82%88%E3%81%86