Artículo
Kurro Lopez · 30 jul, 2020 Lectura de 9 min

Visualizando la jungla de datos - Parte I. Hagamos un gráfico

Este es el primer artículo de una serie que se sumerge en herramientas de visualización y análisis de datos de series temporales. Obviamente, estamos más interesados en analizar los datos relacionados con el rendimiento que podemos recopilar de la familia de productos Caché. Sin embargo, como veremos más adelante, no estamos limitados a eso. Por ahora estamos explorando Python y las bibliotecas/herramientas disponibles dentro de ese ecosistema.

La serie está estrechamente vinculada a la excelente serie de Murray sobre el rendimiento y la supervisión de Caché. (ver aquí) y mas especificamente este artículo.

Descargo de responsabilidad I: Si bien hablaré de pasada sobre la interpretación de los datos que estamos viendo, hablar de eso en detalle distraería demasiado del objetivo real. Recomiendo encarecidamente la serie de Murray para comenzar, para obtener una comprensión básica del tema.

Descargo de responsabilidad II: Existen miles de millones de herramientas que le permiten visualizar los datos recopilados. Muchos de ellos trabajan directamente con los datos que obtiene de mgstat y sus amigos, o solo necesitan un ajuste mínimo. Esta no es una publicación de 'esta solución es la mejor'. Es solo una de las formas en que he encontrado útil y eficiente trabajar con los datos.

Descargo de responsabilidad III: La visualización y el análisis de datos es un campo altamente adictivo, emocionante y divertido para sumergirse. Puede perder algo de tiempo libre por esto. ¡Usted ha sido advertido!

Entonces, sin más preámbulos, profundicemos en ello.

Prerequisitos

Para comenzar, necesitará algunas herramientas y bibliotecas:
* Jupyter notebooks
* Python (3)
* varias bibliotecas de Python que usaremos en el futuro

Python (3) Necesitará Python en su máquina. Hay numerosas formas de instalarlo en varias arquitecturas. Yo uso homebrew en mi mac, lo que lo hizo fácil:

```
brew install python3
```

Solicite a google instrucciones para su plataforma favorita.

Jupyter notebooks: Si bien no es técnicamente necesario, el Jupyter notebooks hace que trabajar en scripts de python sea muy fácil. Le permite ejecutar interactivamente y mostrar scripts de Python desde una ventana del navegador. También permite trabajar en colaboración en scripts. Como hace que sea muy fácil experimentar y jugar con código, es muy recomendable.

```
pip3 install jupyter
```

(de nuevo, habla con $search-engine ;)

Librerías Python Mencioné las diferentes bibliotecas de Python mientras las usamos en el futuro. Si obtiene un error en una declaración **import**, una buena primera aproximación es siempre asegurarse de tener instalada la biblioteca:

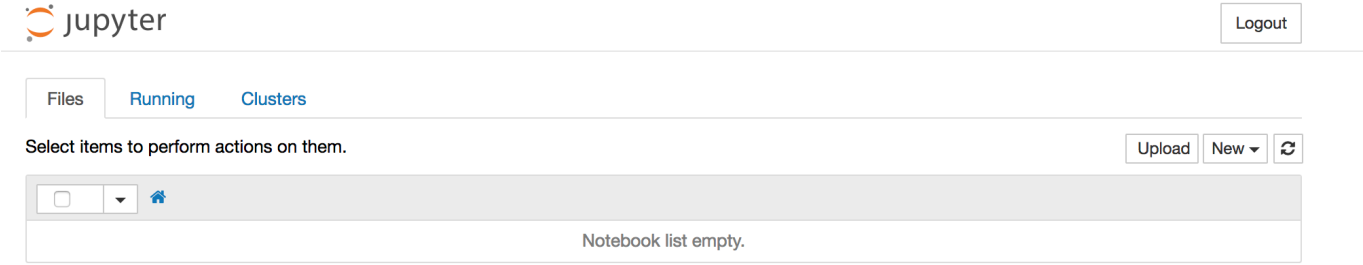
```
pip3 install matplotlib
```

Empezando

Suponiendo que tiene todo instalado en su máquina, debería poder ejecutar

```
jupyter notebook
```

de un directorio.
Esto debería abrir automáticamente una ventana del navegador con una interfaz de usuario simple.



Continuaremos y crearemos un nuevo cuaderno a través del menú y agregaremos un par de declaraciones de importación a nuestra primera celda de código (New -> Notebooks -> 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
```

En cuanto a las bibliotecas que estamos importando, solo quiero mencionar algunas:
 * Pandas "es una biblioteca de código abierto con licencia BSD que proporciona estructuras de datos y herramientas de análisis de datos de alto rendimiento y fáciles de usar para el lenguaje de programación Python". Lo que permite trabajar eficientemente con grandes conjuntos de datos. Si bien los conjuntos de datos que obtenemos de pButtons, de ninguna manera son 'big data'. Sin embargo, nos consolaremos con el hecho de que podríamos ver muchos datos a la vez. Imagine que ha estado recolectando pButtons con muestreo de 24 h/2 segundos durante los últimos 20 años en su sistema. Podríamos graficar eso.
 * Matplotlib "matplotlib es una biblioteca de trazado 2D de python que produce cifras de calidad de publicación en una variedad de formatos impresos y entornos interactivos en todas las plataformas". Este será el principal motor de gráficos que vamos a utilizar (por ahora).

Si recibe un error al ejecutar la celda de código actual (shortcut: Ctrl+Enter) (lista de atajos), asegúrese de verificar que los tenga instalados.

También notará que cambié el nombre del cuaderno *Sin título*, para hacerlo, simplemente puede hacer clic en el título.

Cargando algunos datos

Ahora que pusimos un poco de terreno, es hora de obtener algunos datos. Por suerte, Pandas proporciona una manera fácil de cargar datos CSV. Ya que tenemos un conjunto de datos de mgstat por ahí in csv-format, we'll just use that.

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

Nosotras estamos utilizando el readcsv comando para leer directamente los datos de mgstat en un DataFrame. Consulte la documentación completa para una descripción completa de las opciones. En resumen: simplemente estamos pasando el archivo para leerlo y decirle que la segunda línea (¡basada en 0!) Contiene los nombres de los encabezados.
Dado que mgstat divide los campos de fecha y hora en dos campos, también necesitamos combinarlos con el parámetro parsedates.

```
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
```

nos da una buena visión general del DataFrame recopilado.

Trabajando con los datos

Dado que algunos nombres de campo contienen espacios y guión bajo ("Date_Time") es bastante difícil de manejar, seguiremos adelante y eiminamos espacios y cambiaremos el nombre de la primera columna:

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

El Marco de datos predeterminado es un RangeIndex. Esto no es muy útil para ver nuestros datos. Como tenemos disponible una columna DateTime bastante práctica, seguiremos adelante y la configuraremos como índice:

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

Ahora estamos listos para crear una versión inicial de nuestra trama. Como esta es siempre una de las primeras cosas a tener en cuenta, usemos Glorefs para esto:

```
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()
```

Primero le decimos a la biblioteca en qué tamaño queremos el gráfico. También queremos que las etiquetas del eje x giren un poco, para que no se superpongan.
Finalmente graficamos DateTime vs Glorefs y mostramos el gráfico. Esto nos da algo como el siguiente gráfico.

Podemos reemplazar fácilmente Glorefs con cualquiera de las otras columnas para tener una idea general de lo que está sucediendo.

Combinando gráficos

En algún momento es bastante útil mirar múltiples gráficos a la vez. Entonces, la idea de dibujar varias parcelas en una sola gráfica parece natural.

Si bien es muy sencillo hacerlo solo con matplotlib:

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

Esto nos dará más o menos la misma gráfica que antes. El problema, por supuesto, es la escala y. Dado que Glorefs sube a millones, mientras que los PhyRds generalmente están en los 100 (a miles), no los vemos.

Para resolver esto, necesitaremos usar el kit de herramientas axisartist previamente importado.

```
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()
```

El breve resumen es: agregaremos dos ejes y al diagrama, que tendrán su propia escala. Si bien utilizamos implícitamente la subtrama en nuestro primer ejemplo, en este caso necesitamos acceder a ella directamente para poder agregar el eje y las etiquetas.
Establecemos un par de etiquetas y los colores. Después de agregar una leyenda y conectar los colores a las diferentes parcelas, terminamos con una imagen como esta:

Comentarios finales

Esto ya nos da un par de herramientas muy poderosas para trazar nuestros datos. Exploramos cómo cargar datos de mgstat y crear algunos gráficos básicos. En la siguiente parte, jugaremos con diferentes formatos de salida para nuestros gráficos y obtendremos más datos.

Comentarios y preguntas son alentados! ¡Comparte tus experiencias!

-Fab

ps. el cuaderno para esto está disponible aquí

#Big Data #Herramientas #Modelo de datos de objetos #Python #Visualización #Caché

URL de fuente:https://es.community.intersystems.com/post/visualizando-la-jungla-de-datos-parte-i-hagamos-un-gr%C3%A1fico