prog_datasci_5_api

August 7, 2019

1 Programación para *Data Science*

1.1 Unidad 5: Adquisición de datos en Python

1.2 Instrucciones de uso

A continuación se presentarán explicaciones y ejemplos de adquisición de datos en Python. Recordad que podéis ir ejecutando los ejemplos para obtener sus resultados.

1.3 Introducción

Los procesos de adquisición de datos son muy diversos. En esta unidad, veremos ejemplos de adquisición de datos de Internet con tres métodos diferentes:

- descarga directa
- petición a APIs de terceros
- *web crawling*

Por lo que respeta a la interacción con APIs de terceros, repasaremos dos alternativas, la construcción manual de las peticiones HTTP y el uso de librerías Python.

Con relación al *web crawling*, veremos cómo utilizar la librería Scrapy para construir un pequeño *web crawler* que capture datos de nuestro interés.

1.3.1 Primeros pasos

En esta unidad trabajaremos en varias ocasiones con datos en formato JSON (recordad que ya hemos introducido el formato JSON en la xwiki).

La librería json de Python nos ofrece algunas funciones muy útiles para trabajar en este formato. Por ejemplo, podemos obtener la representación JSON de objetos Python o crear objetos Python a partir de su representación en JSON.

[19]:
```
# Construimos un diccionario de ejemplo y mostramos el tipo de datos y el␣
 ↪contenido de la variable.
diccionario_ejemplo = {"nombre": "Yann", "apellidos": {"apellido1": "LeCun",␣
 ↪"apellido2": "-"}, "edad": 56}
print(type(diccionario_ejemplo))
print(diccionario_ejemplo)
```

```
# Construimos una lista de ejemplo y mostramos el tipo de datos y el contenido␣
 ↪de la variable.
lista_ejemplo = [1, 2, 3]
print(type(lista_ejemplo))
print(lista_ejemplo)
```

```
<class 'dict'>
{'nombre': 'Yann', 'apellidos': {'apellido1': 'LeCun', 'apellido2': '-'},
'edad': 56}
<class 'list'>
[1, 2, 3]
```

[2]:
```
# Importamos la librería json.
import json

# Mostramos la representación json del diccionario.
json_dict = json.dumps(diccionario_ejemplo)
print(type(json_dict))
print(json_dict)

# Mostramos la representación json de la lista.
json_list = json.dumps(lista_ejemplo)
print(type(json_list))
print(json_list)
```

```
<class 'str'>
{"nombre": "Yann", "apellidos": {"apellido1": "LeCun", "apellido2": "-"},
"edad": 56}
<class 'str'>
[1, 2, 3]
```

Fijaos que, en ambos casos, obtenemos una cadena de caracteres que nos representa, en formato JSON, los objetos Python. Este proceso se conoce como **serializar** el objeto.

También podemos realizar el proceso inverso (conocido como **deserializar**), creando objetos Python (por ejemplo, listas o diccionarios) a partir de cadenas de texto en formato JSON.

[3]:
```
# Deserializamos la cadena json_dict.
diccionario_ejemplo2 = json.loads(json_dict)
print(type(diccionario_ejemplo2))
print(diccionario_ejemplo2)

# Deserializamos la cadena json_list.
lista_ejemplo2 = json.loads(json_list)
print(type(lista_ejemplo2))
print(lista_ejemplo2)
```

```
<class 'dict'>
{'nombre': 'Yann', 'apellidos': {'apellido1': 'LeCun', 'apellido2': '-'},
```

```
'edad': 56}
<class 'list'>
[1, 2, 3]
```

Para mejorar la legibilidad de los datos que obtendremos de las API, definiremos una función que mostrará cadenas JSON por pantalla formateadas para mejorar la lectura. La función aceptará tanto cadenas de carácteres con contenido JSON como objetos Python, y mostrará el contenido por pantalla.

Además, la función recibirá un parámetro opcional que nos permitirá indicar el número máximo de líneas que hay que mostrar. Así, podremos usar la función para visualizar las primeras líneas de un JSON largo, sin tener que mostrar el JSON completo por pantalla.

[4]:
```
# Define la función 'json_print', que tiene un parámetro obligatorio␣
 ↪'json_data' y un parámetro opcional limit
# y no devuelve ningún valor.
# La función muestra por pantalla el contenido de la variable 'json_data' en␣
 ↪formato JSON, limitando el número
# de líneas a mostrar si se incluye el parámetro limit.
def json_print(json_data, limit=None):
    if isinstance(json_data, (str)):
        json_data = json.loads(json_data)
    nice = json.dumps(json_data, sort_keys=True, indent=3, separators=(',', ':␣
 ↪'))
    print("\n".join(nice.split("\n")[0:limit]))
    if limit is not None:
        print("[...]")
```

Veamos un ejemplo del resultado de utilizar la función que acabamos de definir.

[5]:
```
# Muestra el valor de la variable 'json_ejemplo' con la función 'print'.
json_ejemplo = '{"nombre": "Yann", "apellidos": {"apellido1": "LeCun",␣
 ↪"apellido2": "-"}, "edad": 56}'
print(json_ejemplo)
```

```
{"nombre": "Yann", "apellidos": {"apellido1": "LeCun", "apellido2": "-"},
"edad": 56}
```

[6]:
```
# Muestra el valor de la variable 'json_ejemplo' con la función 'json_print'␣
 ↪que acabamos de definir.
json_print(json_ejemplo)
```

```
{
   "apellidos": {
      "apellido1": "LeCun",
      "apellido2": "-"
   },
   "edad": 56,
   "nombre": "Yann"
}
```

[7]:
```
# Mostramos únicamente las tres primeras líneas.
json_print(json_ejemplo, 3)
```

```
{
   "apellidos": {
      "apellido1": "LeCun",
[...]
```

1.4 Descarga directa de datos

La descarga directa del conjunto de datos es quizás el método más sencillo de adquisición de datos y consiste en descargar un fichero con los datos de interés ya recopilados por algún otro analista. De hecho, en la unidad anterior ya hemos usado este método para adquirir el fichero con los datos sobre los personajes de cómic de Marvel. Una vez descargado el fichero, el procedimiento para cargarlo en Python dependerá del formato concreto (ya hemos visto un ejemplo de carga de datos desde un fichero .csv).

Algunos de los sitios web donde podéis encontrar conjuntos de datos para analizar son: - Open Data gencat, el portal de datos abiertos de la Generalitat. - datos.gov.es, el catálogo de conjuntos de datos del Gobierno de España. - European Data Sources, el portal de datos abiertos de la Unión Europea. - Mark Newman network datasets, conjuntos de datos en forma de red recopilados por Mark Newman. - Stanford Large Network Dataset Collection, otra recopilación de conjuntos de datos en forma de red, en este caso creado por Jure Leskovec. - SecRepo.com, datos relacionados con la seguridad. - AWS Public Datasets, conjuntos de datos recopilados y hospedados por Amazon. - UC Irvine Machine Learning Repository, datos recopilados por un grupo de investigación de la Universidad de California en Irvine. - El repositorio de Five Thirty Eight, que recoge datos utilizados en artículos de la publicación y que ya hemos visto en la unidad anterior.

1.5 Uso de API de terceros

1.5.1 Acceso a API manualmente

Podemos utilizar la librería de Python Requests para realizar peticiones a web API de manera manual. Para ello, tendremos que acceder a la documentación de la API con la que queramos actuar, construir manualmente las peticiones para obtener la información deseada y procesar también manualmente la respuesta recibida.

Veamos un ejemplo de petición HTTP a una API pública. El sitio http://postcodes.io/ ofrece una API de geolocalización sobre códigos postales en el Reino Unido. Leyendo la documentación, podemos ver que tiene un método GET con la URL http://api.postcodes.io/postcodes/:código-postal que nos retorna información del código postal especificado.

[8]:
```
# Importamos la librería.
import requests

# Realizamos una petición get a la API, preguntando sobre el código postal "E98
 ↪1TT"
# Notad que el carácter espacio se codifica como %20 en la URL.
response = requests.get('http://api.postcodes.io/postcodes/E98%201TT')
```

```python
# Mostramos la respuesta recibida.
print("Código de estado de la respuesta: ", response.status_code, "\n")
print("Cabecera de la respuesta: ")
json_print(dict(response.headers))
print("\nCuerpo de la respuesta: ")
json_print(str(response.text))
```

```
Código de estado de la respuesta:  200

Cabecera de la respuesta:
{
   "Access-Control-Allow-Origin": "*",
   "Connection": "keep-alive",
   "Content-Length": "805",
   "Content-Type": "application/json; charset=utf-8",
   "Date": "Mon, 05 Aug 2019 07:45:30 GMT",
   "ETag": "W/\"325-s87vTpVPeXA7mcWcF8hfLWqOGL8\"",
   "Server": "nginx/1.14.0",
   "X-GNU": "Michael J Blanchard"
}

Cuerpo de la respuesta:
{
   "result": {
      "admin_county": null,
      "admin_district": "Tower Hamlets",
      "admin_ward": "St Katharine's & Wapping",
      "ccg": "NHS Tower Hamlets",
      "ced": null,
      "codes": {
         "admin_county": "E99999999",
         "admin_district": "E09000030",
         "admin_ward": "E05009330",
         "ccg": "E38000186",
         "ced": "E99999999",
         "nuts": "UKI42",
         "parish": "E43000220",
         "parliamentary_constituency": "E14000882"
      },
      "country": "England",
      "eastings": 534427,
      "european_electoral_region": "London",
      "incode": "1TT",
      "latitude": 51.508024,
      "longitude": -0.064393,
      "lsoa": "Tower Hamlets 026B",
      "msoa": "Tower Hamlets 026",
```

```
        "nhs_ha": "London",
        "northings": 180564,
        "nuts": "Tower Hamlets",
        "outcode": "E98",
        "parish": "Tower Hamlets, unparished area",
        "parliamentary_constituency": "Poplar and Limehouse",
        "postcode": "E98 1TT",
        "primary_care_trust": "Tower Hamlets",
        "quality": 1,
        "region": "London"
    },
    "status": 200
}
```


Como podemos ver, el estado de la respuesta es 200, lo que nos indica que la petición se ha procesado correctamente. Entre otros campos, la cabecera de la respuesta incluye el tipo de contenido que encontraremos en el cuerpo, que será un texto en formato JSON. Por último, el cuerpo de la respuesta incluye datos sobre el código postal consultado. Por ejemplo, podemos ver que corresponde a Inglaterra (concretamente, a la ciudad de Londres).

Notad que podemos visualizar también la respuesta accediendo a la misma URL con un navegador web. En este caso, se pueden instalar extensiones específicas que gestionen la visualización mejorada del JSON retornado (por ejemplo, JSONView para Chrome o Firefox).

1.5.2 Acceso a API con librerías de Python

Aunque podríamos usar este método para interactuar con cualquier API HTTP, lo cierto es que cuando la complejidad de las funciones disponibles incrementa (por ejemplo, al incluir autenticación) puede no resultar muy práctico. Cuando queramos acceder a APIs populares, normalmente encontraremos que ya existen librerías de Python diseñadas para interactuar con las estas APIs, de manera que podremos obtener datos sin necesidad de manejar las peticiones HTTP manualmente.

Por ejemplo, Twitter, la famosa plataforma de envío de mensajes cortos, ofrece varias APIs que permiten obtener datos de la red. Disponemos de varias librerías de Python que permiten interactuar con la API de Twitter. En este notebook, veremos cómo obtener datos de Twitter usando Tweepy.

Autenticación con la API de Twitter Twitter requiere autenticación para poder utilizar su API. Por este motivo, el primer paso a realizar para poder obtener datos de Twitter a través de su API es conseguir unas credenciales adecuadas. En esta sección, describiremos cómo obtener credenciales para acceder a la API de Twitter.

Para empezar, es necesario disponer de una cuenta en Twitter. Para poder ejecutar los ejemplos del notebook, necesitaréis por lo tanto tener una cuenta de Twitter. Podéis utilizar vuestra cuenta personal, si ya disponéis de ella, para solicitar los permisos de desarrollador que nos permitirán interactuar con la API. En caso contrario (o si preferís no usar vuestra cuenta personal), podéis crearos una cuenta de Twitter nueva. El proceso es muy sencillo: 1. Acceder a Twitter. 2. Pulsar sobre *Sign up for Twitter* y seguir las indicaciones para completar el registro.

Después, habrá que solicitar convertir la cuenta recién creada (o vuestra cuenta personal), en una cuenta de desarrollador. Para hacerlo, hay que seguir los siguientes pasos: 1. Acceder al panel

de desarolladores de Twitter. 2. Clickar sobre *Apply*. 3. Clickar sobre *Apply for a developer account*. 3. Pulsar *Continue*. 4. Indicar porqué queréis disponer de una cuenta de desarrollador.

Para poder realizar este proceso satisfactoriamente, necesitaréis que vuestra cuenta disponga de un número de teléfono asociado verificado. En caso contrario, veréis que os aparecerá un mensaje para que verifiquéis vuestro teléfono.

Finalmente, una vez ya disponemos de una cuenta en Twitter, será necesario registrar una nueva aplicación. Para hacerlo, es necesario seguir los siguientes pasos: 1. Acceder al panel de desarolladores de Twitter. 2. Pulsar sobre *Create new app*. 3. Rellenar el formulario con los detalles de la aplicación. En concreto, necesitaréis proporcionar como mínimo los campos: * *App name* * *Application description* * *Website URL* * *Tell us how this app will be used*

El campo Website debe contener una URL válida (por ejemplo, el enlace a vuestro perfil de Twitter).

Una vez creada la aplicación, podéis acceder a la pestaña *Keys and access tokens*. Allí se encuentran las credenciales recién creadas para vuestra aplicación, que utilizaremos para autenticarnos y poder utilizar la API de Twitter. Veréis que ya tenéis las claves *Consumer API keys* disponibles. Además, será necesario pulsar sobre *Create* en la sección *Access token & access token secret* para obtener también ambos tokens. Los cuatro valores serán usados para autenticar nuestra aplicación: * API / Consumer Key * API / Consumer Secret * Access Token * Access Token Secret

La librería Tweepy Tweepy nos permite interactuar con la API de Twitter de una manera sencilla, ya que encapsula los métodos HTTP de la API en métodos de Python, que pueden ser llamados directamente. Encontraréis la documentación de la librería en el siguiente enlace.

[13]:
```
# Importamos la librería tweepy
import tweepy

# IMPORTANTE: Es necesario incluir las credenciales de acceso que hayáis␣
 ↪obtenido al crear vuestra App
# para ejecutar el ejemplo.
consumer_key = ''
consumer_secret = ''
access_token = ''
access_secret = ''

# Inicializamos la interacción con la API
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)
api = tweepy.API(auth)

# Obtenemos datos del usuario "twitter" usando la librería tweepy
user = api.get_user('twitter')

print("El tipo de datos de la variable user es: {}".format(type(user)))
print("El nombre de usuario es: {}".format(user.screen_name))
print("El id de usuario es: {}".format(user.id))
```

```
El tipo de datos de la variable user es: <class 'tweepy.models.User'>
El nombre de usuario es: Twitter
```