前言

现在越来越多的数据以json的格式进行存储，例如通过网络爬虫时，那些异步存储的数据往往都是json类型的；再如企业数据库中的日志数据，也会以json的格式存放。前不久，一位网友就碰到了这个问题，手中Excel存储的数据并不是标准化的结构数据，而是以json格式存储在Excel的每个单元格。那今天我们就来聊聊如何利用Python将半结构化的json数据转换成结构化数据。

简单的json格式

其实json的格式与Python中的字典非常类似，数据放在大括号（{}）内，每一个元素都是键值对，元素之间以逗号隔开。我们都知道，在Python中，是可以将一个字典对象转换成数据框的，接下来我们就通过一个简单的例子慢慢进入复杂的环境。

```
# 加载第三方包
import pandas as pd # 数据处理包
import numpy as np # 数值计算包
import json # json文件转换包

# 一个简单的json格式字符串
string1 = '{"name":"Sim","gender":"Male","age":28,"province":"江苏"}'
string2 = '{"name":"Lily","gender":"Feale","age":25,"province":"湖北"}'

# 查看数据类型
type(string)

# 将json格式转换为字典
dict1 = json.loads(string1)
dict2 = json.loads(string2)

type(dict1)
```

上面构造的json数据实际上是字典型的字符串，可以直接通过json包中的loads函数完成由字符型到字典型的转化。那如何根据这两个字典，组装成一个2行3列的数据框呢？只需借助于pandas模块中的DataFrame函数即可：

```
# 将字典数据转换为数据框
pd.DataFrame([dict1,dict2])
```

这里需要注意的是，上面的字典，是一个键仅对应一个值的情况，如果直接将dict1传递给DataFrame函数是会出错的，除非你指定索引值。所以，当你有两个及以上的这种字典时，你是可以传递给DataFrame函数的，但必须以列表或元组的形式。还有一种字典，是一个键对应多个值，如果是这样的字典，就可以直接将字典扔给DataFrame函数形成数据框了：

```
string =  '{"name":["Sim","Lily"],"gender":["Male","Feale"],\
           "age":[28,25],"province":["江苏","湖北"]}'
# 转换为数据框
pd.DataFrame(json.loads(string))
```

尽管这样完成了一个字典到数据框的转换，但千万注意，如果一个字典的键包含多个值，那一定要保证所有键对应的值个数一致！OK，了解了这个基础知识点后，我们来两个案例，加深一下对知识点的理解。

经典案例一

先来看一下Excel表中存储的数据格式，现在的问题是，如何将表中UserBasic一列拆解出来，即所有键值对转换成变量名和观测值。

```
data1 = pd.read_excel(r'C:\Users\Administrator\Desktop\data1.xlsx')
data1.head()

data1.UserBasic[0]
```

从上面的反馈结果来看，表中UserBasic字段的单元格存储的json字符串都是一个键仅对应一个值，这跟上面介绍的string1和string2是一致的，故如果需要转换成数据框的话，需要将这些转换的字典存放到列表中。具体操作如下：

```
# UserBasic列中的信息拆分到各个变量中
basic = []
for i in data1.UserBasic:
    basic.append(json.loads(i))

UserBasic = pd.DataFrame(basic)
UserBasic.head()
```

上面通过循环的方式将UserBasic字段的每一行解析成字典，并保存到列表中，最后通过DataFrame函数完成数据框的转换。接下来需要将拆分出来的这列，与原始表中的Id变量，Mobile变量整合到一起。

```
# 数据整合到一起
final_data = pd.concat([data1[['Id','Mobile']],UserBasic], axis = 1)
final_data.head()
```

效果呈现还是蛮好的，但是有一点不好的是，通过for循环来完成毕竟不是高效的，如果数据量特别大，上百万行的话，那就得循环执行上百万次，会耗很多时间。这里我们借助于apply方法，避免显式的循环：

```
# 避免循环的方式
trans_data = pd.DataFrame(data1.UserBasic.apply(json.loads))

# 数据整合到一起
final_data = pd.concat([data1[['Id','Mobile']],trans_data], axis = 1)
```

经典案例二

我们接着看第二个例子，原始数据如下图所示，现在的问题是在解析字段CellBehaviorData的同时，还要做一次聚合操作（每个用户ID近3个月的消费平均水平）。

```
# 读取数据
data2 = pd.read_excel(r'C:\Users\Administrator\Desktop\data2.xlsx')

# 查看字段CellBehaviorData第一行的信息
data2.CellBehaviorData[0]
```

细心的你一定发现了个问题，这个字符串的起始和结尾并不是大括号（{}），而是中括号（[]），故接下来要做的第一件事就是去除这两个中括号；另一方面，behavior键对应的值是列表，而且列表中还有多个相同的键，如sms_cnt、cell_phone_num等。这样的json最后形成的数据框一定是多行的，即表中一个单元格会就可以转换成多行的数据框。不妨，我们先来看一下变量CellBehaviorData第一行形成是数据框张啥样：

```
# 通过切片的方式去除首尾的中括号
s = data2.CellBehaviorData[0][1:-1]
# 将字符串转换成字典，并取出behavior键
d = json.loads(s)['behavior']
# 将字典转换为数据框
df = pd.DataFrame(d)
df
```

这就是一行观测产生的多行数据框，现在的问题是如何将多行数据框与每一个Id配对上。我们发现字典中除了behavior键，还有phone_num键，而且该键的值是唯一的，正好与上面数据框的cell_phone_num变量匹配。所以，待会做数据关联的时候，就使用phone_num变量和cell_phone_num变量。

```
# 取出phone_num
phone_num = [i['phone_num'] for i in data2.CellBehaviorData.str[1:-1].apply(json.loads)]

# 取出CellBehaviorData字段，并解析为数据框
df = pd.concat([pd.DataFrame(j) for j in [i['behavior'] for i in
data2.CellBehaviorData.str[1:-1].apply(json.loads)]])
# 将Id与手机号捆绑
user = pd.concat([pd.Series(phone_num,name = 'phone_num'), data2.Id], axis = 1)

# 以手机号作为数据的关联关联
final_data = pd.merge(df, user, left_on = 'cell_phone_num', right_on='phone_num')
```

```
final_data.head()
# 查看数据类型
final_data.dtypes
```

为了速度的提升，上面使用了apply技术和列表解析式的技巧将json数据拆解成数据框，同时，发现所有变量类型都是字符串型，需要对数值变量进行转换，因为接下来还要做聚合操作：

```
# 挑选需要转换类型的变量名称
vars =
['call_cnt','call_in_cnt','call_in_time','call_out_cnt','call_out_time','net_flow','sms_cnt','total_amount']
# 对以上变量进行数据类型转换
df_convert = final_data[vars].apply(lambda x: x.astype('float'))

# 从新完成数据合并
final_data2 = pd.concat([df_convert, final_data.loc[:,~final_data.columns.isin(vars)]], axis = 1)

# 对每个id计算近三个月的平均指标值
stats = final_data2.loc[final_data2.cell_mth.isin(['2017-08','2017-07','2017-
06']),:].groupby('Id').aggregate(np.mean)
stats
```

大功告成，这种类型的数据我们就可以游刃有余的完成转换。但如果CellBehaviorData字段不含有phone_num（键）变量的话，如何实现数据关联呢？这里把解决方案的代码呈现出来：

```
# 构造空列表，存放CellBehaviorData变量每一行形成的数据框
final_data = []
# 使用zip函数捆绑两列，并使用for循环
for Id,CellBehaviorData in zip(data2.Id, data2.CellBehaviorData):
    # 组装数据框
    mydf = pd.DataFrame(json.loads(CellBehaviorData[1:-1])['behavior'])
    # 将数据框与变量Id组装起来
    final_data.append(pd.concat([pd.Series(np.repeat(Id,mydf.shape[0]), name = 'Id'),mydf], axis = 1))

# 构造最终的数据框
final_data = pd.concat(final_data)

# 数据类型转换
vars =
['call_cnt','call_in_cnt','call_in_time','call_out_cnt','call_out_time','net_flow','sms_cnt','total_amount']
# 对以上变量进行数据类型转换
df_convert = final_data[vars].apply(lambda x: x.astype('float'))

# 从新完成数据合并
final_data2 = pd.concat([df_convert, final_data.loc[:,~final_data.columns.isin(vars)]], axis = 1)
final_data2
# 对每个id计算近三个月的平均指标值
stats = final_data2.loc[final_data2.cell_mth.isin(['2017-08','2017-07','2017-
06']),:].groupby('Id').aggregate(np.mean)
stats
```

虽然通过上面的方法可以实现数据的关联和匹配，但个人觉得并不是很理想，因为这里毕竟使用了for循环，一旦数据量大的话，执行起来会比较缓慢，如果高人，还请指点。

结语

OK，今天关于半结构化的json数据转数据框的分享就介绍到到这里，希望本篇文章对各位网友有一定的帮助。如果你有任何问题，欢迎在公众号的留言区域表达你的疑问。欢迎各位朋友继续转发与分享文中的内容，让更多的朋友学习和进步。有关文中的脚本和数据可至下方的链接获取，再次感谢网友对我的关注和支持。

关注“**每天进步一点点2015**”

相关材料下载链接

链接: https://pan.baidu.com/s/1kVzNnFp 密码: tdkm