Python预处理实例

链接：<https://cloud.tencent.com/developer/article/1555150>

**该数据预处理示例与数据获取实验的示例组合为一个完整的案例，使用的数据集为在数据获取实验中采用的数据集。**

在上一步获取数据之后，我们就可以使用pandas进行数据的分析工作。在正式的分析之前，数据预处理非常重要，它保证了数据的质量，也为后续的工作奠定了重要的基础。通常数据预处理在实际工作中都会占用比较多的时间。虽然我们这里的数据已经足够“结构化”，但仍然不可避免存在一些问题。下面我们就来一探究竟。

# **1.如何读取数据？**

使用pandas提供的read\_csv方法，该方法有很多可选的参数，例如指定索引，列名，编码等。对于本次数据，直接使用默认的即可。读取的ori\_data是dataframe类型，调用head方法可以输出前5行的样例数据。

ori\_data = pd.read\_csv('bjgj\_lines\_utf8.csv')

ori\_data.head()

# **2.如何查看每一列数据的唯一值的个数？(如何查看有多少条线路)**

可以使用dataframe的nunique方法，该方法输出每一列有几个唯一的值。

ori\_data.nunique()

输出结果如下：

line\_name 1986

line\_url 2002

line\_start 989

line\_stop 1123

line\_op\_time 560

line\_interval 4

line\_price 126

line\_company 82

line\_up\_times 650

line\_station\_up 1928

line\_station\_up\_len 80

line\_station\_down 1700

line\_station\_down\_len 80

dtype: int64

由于线路很多，我们在原始网页中很难发现是否会有重复的线路。但从上面观察line\_name和line\_url两个字段，line\_name有1986个唯一值，line\_url有2002个唯一值。说明line\_name存在重复：会有名称相同的线路对应不同的line\_url。所以接下来我们需要进行重复值的剔除。

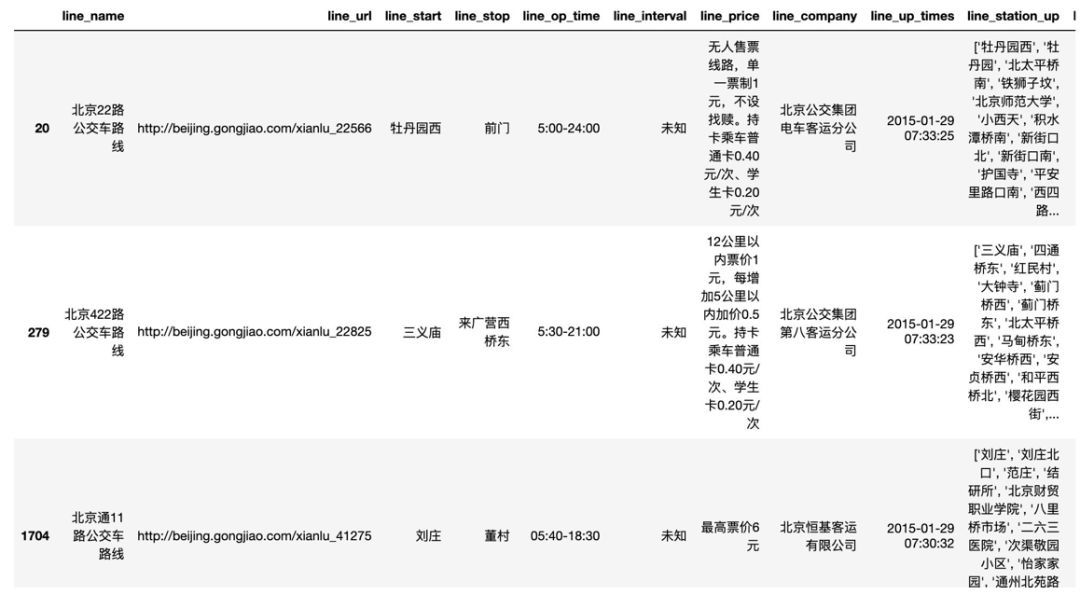
# **3.如何找出重复的值？**

出现了线路名称的重复，但却有不同的line\_url，究竟是确实是线路**“重名”**还是线路**“重复”**？我们需要看一下数据重复的具体情况。因此需要把重复的行都找出来看看。可以使用pandas的duplicated方法，它可以对dataframe的指定列查看是否重复，返回True和False，代码如下。

d = ori\_data.duplicated(subset=['line\_name'])

dup\_data = ori\_data[d]

dup\_data



这是所有重复出现过的line\_name值，但并不是所有重复的值(例如22路重复出现过，但22路在结果中只有一条，不便于观察除了名字之外是否还有其他字段的重复)。为了找出所有重复的值(例如输出所有22路的记录)，我们可以从原数据中取line\_name是这些值的所有行，代码和思路如下：

#首先定义一个列表，每找出一行line\_name在上面范围内的，

#就将这行加入列表，然后调用concat方法将列表拼接成#dataframe

dup\_lines = []

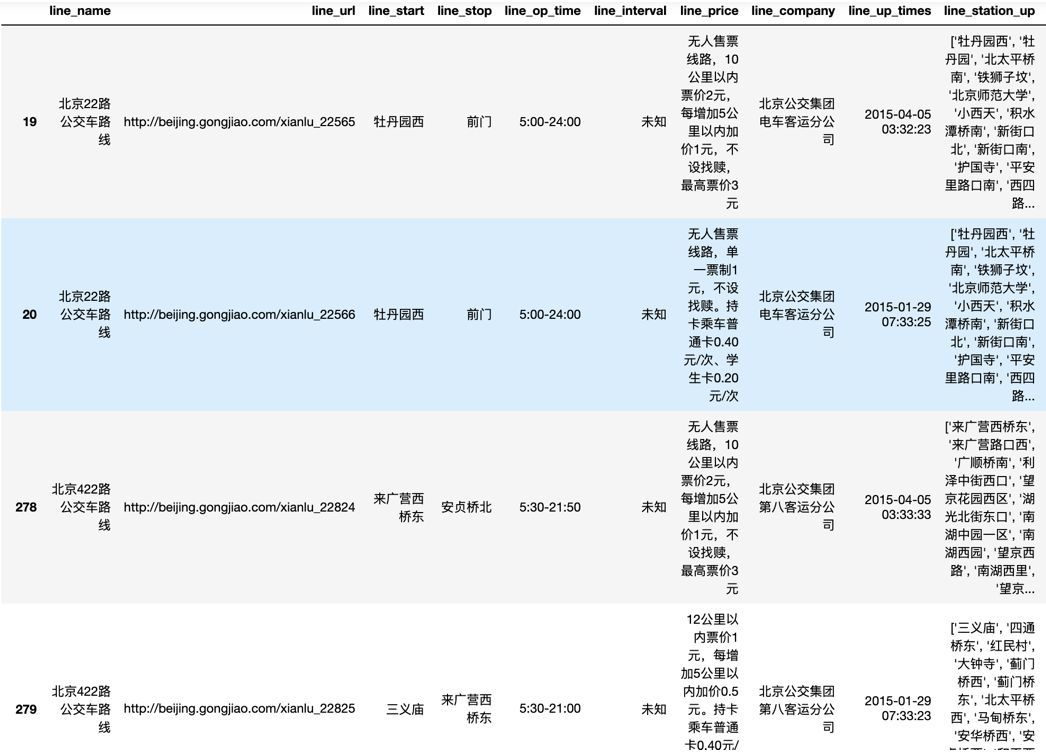
for name in dup\_data.line\_name:

tmp\_lines = ori\_data[ori\_data['line\_name'] == name]

dup\_lines.append(tmp\_lines)

dup\_data\_all = pd.concat(dup\_lines)

dup\_data\_all



观察dup\_data\_all，确实同一个线路名字存在重复的记录，而且其余信息也是几乎都相同的，这确认了我们认为的线路**”重名“**现象是不存在的。但同一条线路的信息具体以哪一个为准呢？注意到有更新时间line\_up\_time字段，因此我们可以以最新时间的信息为准。

# **4.如何对原数据剔除重复值？**

这里考虑两种思路。第一种，直接对原数据进行操作，当line\_name存在重复时，保留最近更新时间的记录。第二种，将原数据中的dup\_data\_all部分完全删除，拼接上dup\_data\_all去除重复的部分。两种思路都需要删除line\_name重复的记录，保留一个时间最新的。pandas本身有drop\_duplicates方法，使用keep=last或keep=first参数就可以指定保留的记录。但在这之前我们需要将line\_up\_time转换为pandas可以识别的时间类型，然后对其进行排序。下面来看代码：

#方法1

ori\_data['line\_up\_times'] = pd.to\_datetime(ori\_data['line\_up\_times'], format='%Y-%m-%d %H:%M:%S')#使用to\_datetime方法，指定format，将字符串转换为pandas的时间类型。

ori\_data.sort\_values(by=['line\_name', 'line\_up\_times'], ascending=[True, True], inplace=True)#使用sort\_values方法，对line\_name和line\_up\_time排序

drop\_dup\_line1 = ori\_data.drop\_duplicates(subset=['line\_name'], keep='last')#由于是升序排列，所以keep=last就可以保留最新事件的记录

len(drop\_dup\_line1)#结果是1986

方法2：

dup\_data\_all['line\_up\_times'] = pd.to\_datetime(dup\_data\_all['line\_up\_times'], format='%Y-%m-%d %H:%M:%S')#使用to\_datetime方法，指定format，将字符串转换为pandas的时间类型。

dup\_data\_all.sort\_values(by=['line\_name', 'line\_up\_times'], ascending=[True, True], inplace=True)#使用sort\_values方法，对line\_name和line\_up\_time排序

dup\_data\_all.drop\_duplicates(subset=['line\_name'], keep='last', inplace=True)#使用keep=last保留时间更新的记录

other\_data = ori\_data[~ori\_data['line\_name'].isin(dup\_data\_all.line\_name)]#获取原数据中剔除了重复线路的数据：取名字不在dup\_data\_all的line\_name集合中的记录

drop\_dup\_line2 = pd.concat([other\_data, dup\_data\_all]) #拼接两部分数据

len(drop\_dup\_line2)#结果是1986

**如何比较两种方法获得的结果线路是否一致？**我们可以用下面的代码进行。

drop\_dup\_line2.sort\_values(by=['line\_name', 'line\_up\_times'], ascending=[True, True], inplace=True)#由于drop\_dup\_line1排序过，我们也对drop\_dup\_line2进行相同规则的排序

res = drop\_dup\_line1['line\_name'].values.ravel() == drop\_dup\_line2['line\_name'].values.ravel()#ravel()方法将数组展开，res是一个布尔值组成的ndarray数组，结果为true表示对应元素相等

res = [1 for i in res.flat if i]

sum(res)#使用flat方法可以对ndarray进行遍历，sum看一下一共有多少个true，结果是1986，说明drop\_dup\_line1和drop\_dup\_line2对应每一个位置的元素都相同

这样对于重复数据的处理就结束了，我们使用drop\_dup\_line1来进行下面的分析。

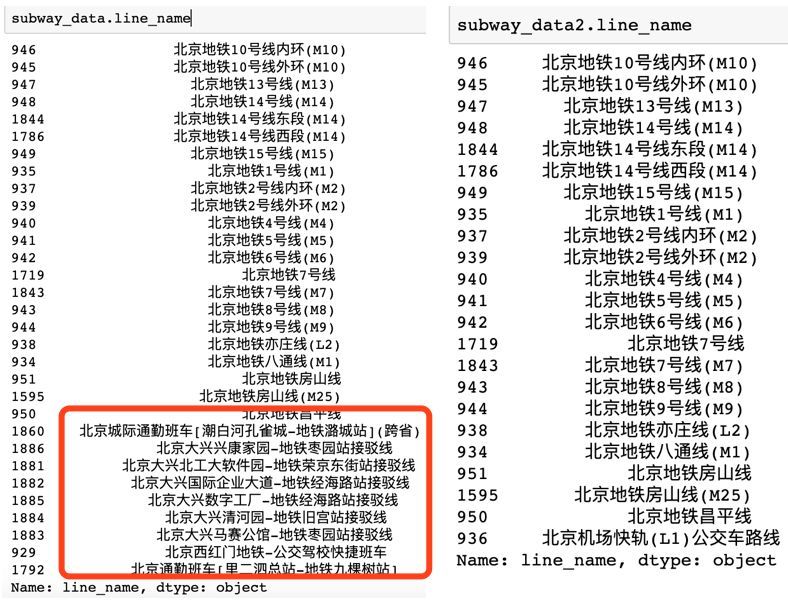
# **5.如何删除地铁线路？**

虽然我们爬取的是公交路线，但程序运行过程中我也发现了地铁的线路(其实地铁也是广义上的公交啦)。如果我们的目的是对纯粹的公交线路进行分析，就需要将地铁的线路删除。直观的思路是剔除线路名称中含有“地铁”的记录。

is\_subway = drop\_dup\_line1.line\_name.str.contains('地铁')#使用.str将其转换为字符串就可以使用字符串的contains方法。

subway\_data = drop\_dup\_line1[is\_subway]

subway\_data



从上图左侧可以看到subway\_data的结果不仅仅有地铁，还有一些地铁有关的通勤线路，其实是公交。因此不能直接删除line\_name中含有“地铁”的记录，我们使用line\_conpany中含有“地铁”来区分，效果更好。代码如下所示：

is\_subway2 = drop\_dup\_line1.line\_company.str.contains('地铁')

subway\_data2 = drop\_dup\_line1[is\_subway2]

subway\_data2

结果如上图右侧所示，虽然最后一条也有一条“公交车路线”，但观察整条记录就会发现它其实是特殊的机场线地铁。

到这里，你会不会想到根据线路名称中是否含有“公交车路线”将地铁线路剔除？我们可以试一试。但其实上面的图已经告诉了我们答案：有的公交线路是“接驳线”，并不含有“公交车路线”。

# **6.获取删除地铁数据之后的全部数据**

在drop\_dup\_line1的基础上，筛选出线路名称不在subway\_data2中的线路名称的记录即可：

clean\_data = drop\_dup\_line1[~drop\_dup\_line1['line\_name'].isin(subway\_data2.line\_name)]

len(clean\_data) #结果是1963，也就是北京的公交车一共有1963条线路

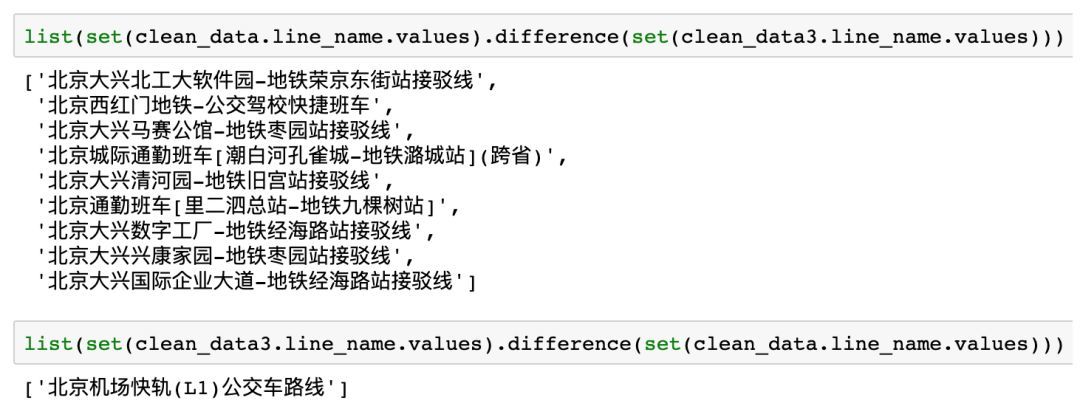
clean\_data3 = drop\_dup\_line1[drop\_dup\_line1.line\_name.str.contains("公交车路线")]

len(clean\_data3) #通过是否含有“公交车线路”进行筛选，结果是1955，应该就是少了那些“接驳线”

**如何比较clean\_data和clean\_data3**。这个问题其实是如何求两个dataframe差集的问题，我们转化为求列表的差集，代码和结果如下所示。

list(set(clean\_data.line\_name.values).difference(set(clean\_data3.line\_name.values))) #找出在clean\_data的line\_name中但是不在clean\_data3的line\_name中的数据

list(set(clean\_data3.line\_name.values).difference(set(clean\_data.line\_name.values))) #找出在clean\_data3的line\_name中但是不在clean\_data的line\_name中的数据



至此我们将重复数据进行了删除，并剔除了“地铁”线路。但其实我们的数据预处理工作还没有结束，我们还没有观察数据中是否含有缺失值。

# **7.如何查看数据集中的缺失值情况？**

可以使用isnull().sum()方法查看。发现票价有230个缺失值。参见后面的图片。对于缺失值我们需要在预处理阶段对其进行填充。考虑到票价数据本身不是纯粹的价格数据，而是一大串的文字描述，并且在公交的这种场景下，其实不同线路的票价差别不是很大，因此我们可以使用众数对缺失值进行填充。使用mode方法查看众数，使用fillna方法填补缺失值。

#查看众数的方法：

clean\_data.line\_price.mode()#使用mode()方法查看line\_price的众数

clean\_data.line\_price.value\_counts()#使用value\_counts()方法查看每一个取值出现的次数，第一个也是众数

clean\_data.line\_price.fillna(clean\_data.line\_price.mode()[0], inplace=True)

clean\_data.isnull().sum()



至此我们基本完成了重复值和缺失值的处理。

# ****8. 总结****

本文我们主要借助于北京公交数据的实例，学习了使用python进行数据获取和数据预处理的流程。内容虽然简单但不失完整性。数据获取部分主要使用requests模拟了get请求，使用lxml进行了网页解析并将数据存储到csv文件中。数据预处理部分我们进行了重复值和缺失值的处理，但应该说数据预处理并没有完成。(比如我们可以对运营时间拆分成两列，对站点名称进行清理等，如何进行预处理工作与后续的分析紧密相关)。文章的重点不在于例子的难度，而在于通过具体问题学习python中数据处理的方法。所处理的问题虽然有一定的特殊性，但也方便扩展到其他场景。