zhuanlan.zhihu.com

10套练习，教你如何用Pandas做数据分析【1-5】

26-32 minutes

Pandas是入门Python做数据分析所必须要掌握的一个库，本文精选了十套练习题，帮助读者上手Python代码，完成数据集探索。

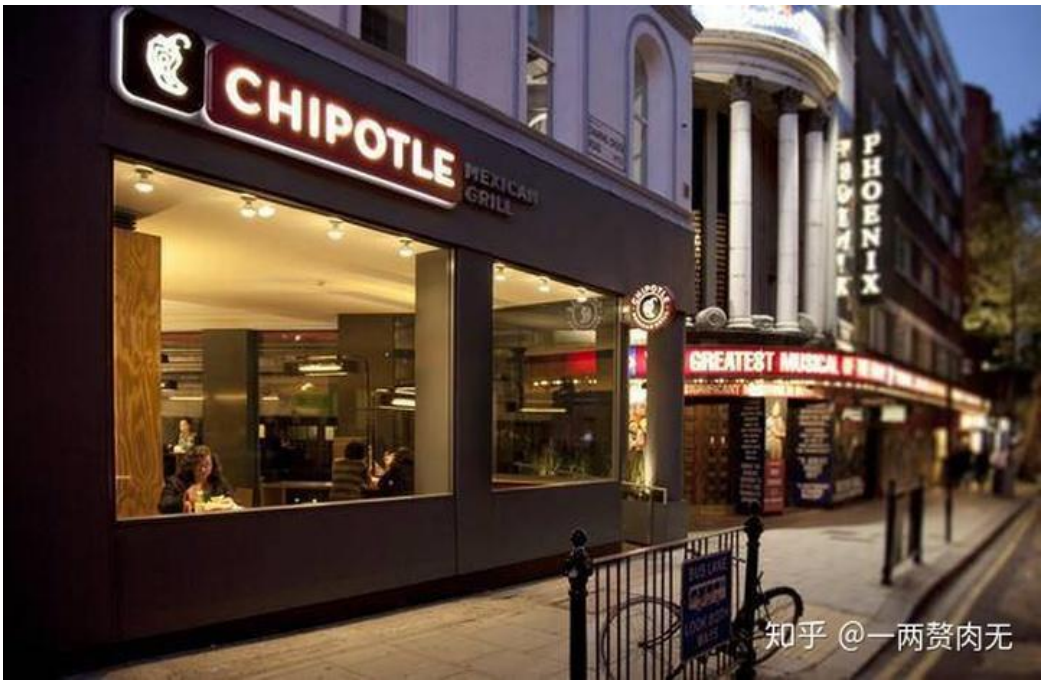
本文内容由和鲸社区翻译整理自Github，建议读者完成科赛网 从零上手Python关键代码 和 Pandas基础命令速查表 教程学习的之后，再对本教程代码进行调试学习。

【小提示：本文所使用的数据集下载地址：DATA | TRAIN 练习数据集】

练习1-开始了解你的数据

探索Chipotle快餐数据

相应数据集：chipotle.tsv



步骤1 导入必要的库

```
# 运行以下代码
import pandas as pd
```

步骤2 从如下地址导入数据集

```
# 运行以下代码
path1 = "../input/pandas_exercise/exercise_data/chipotle.tsv"    # chipotle.tsv
```

步骤3 将数据集存入一个名为chipo的数据框内

```
# 运行以下代码
chipo = pd.read_csv(path1, sep = '\t')
```

步骤4 查看前10行内容

out[235]:

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98
5	3	1	Chicken Bowl	[Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou...	$10.98
6	3	1	Side of Chips	NaN	$1.69
7	4	1	Steak Burrito	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	$11.75
8	4	1	Steak Soft Tacos	[Tomatillo Green Chili Salsa, [Pinto Beans, Ch...	$9.25
9	5	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Pinto...	$9.25

步骤6 数据集中有多少个列(columns)

out[236]:

5

步骤7 打印出全部的列名称

out[237]:

Index(['order_id', 'quantity', 'item_name', 'choice_description', 'item_price'], dtype='object')

步骤8 数据集的索引是怎样的

out[238]:

RangeIndex(start=0, stop=4622, step=1)

步骤9 被下单数最多商品(item)是什么?

```
# 运行以下代码，做了修正
c =
```

```
chipo[['item_name','quantity']].groupby(['item_name'],as_index=False).agg({'quantity':sum})
```

```
c.sort_values(['quantity'],ascending=False,inplace=True)
c.head()
```

out[239]:

	item_name	quantity
17	Chicken Bowl	761
18	Chicken Burrito	591
25	Chips and Guacamole	506
39	Steak Burrito	386
10	Canned Soft Drink	351

步骤10 在item_name这一列中，一共有多少种商品被下单?

```
# 运行以下代码
chipo['item_name'].nunique()
```

out[240]:

50

步骤11 在choice_description中，下单次数最多的商品是什么?

```
# 运行以下代码，存在一些小问题
chipo['choice_description'].value_counts().head()
```

out[241]:

[Diet Coke] 134

[Coke] 123

[Sprite] 77

[Fresh Tomato Salsa, [Rice, Black Beans, Cheese, Sour Cream, Lettuce]] 42

[Fresh Tomato Salsa, [Rice, Black Beans, Cheese, Sour Cream, Guacamole, Lettuce]] 40

Name: choice_description, dtype: int64

步骤12 一共有多少商品被下单?

```
# 运行以下代码
total_items_orders = chipo['quantity'].sum()
total_items_orders
```

out[242]:

4972

步骤13 将item_price转换为浮点数

```
# 运行以下代码
dollarizer = lambda x: float(x[1:-1])
chipo['item_price'] = chipo['item_price'].apply(dollarizer)
```

步骤14 在该数据集对应的时期内，收入(revenue)是多少

```
# 运行以下代码,已经做更正
chipo['sub_total'] = round(chipo['item_price'] * chipo['quantity'],2)
chipo['sub_total'].sum()
```

out[244]:

39237.02

步骤15 在该数据集对应的时期内，一共有多少订单?

```
# 运行以下代码
chipo['order_id'].nunique()
```

out[245]:

1834

步骤16 每一单(order)对应的平均总价是多少?

```
# 运行以下代码，已经做过更正
chipo[['order_id','sub_total']].groupby(by=['order_id']
).agg({'sub_total':'sum'})['sub_total'].mean()
```

out[246]:

21.39423118865867

步骤17 一共有多少种不同的商品被售出?

```
# 运行以下代码
chipo['item_name'].nunique()
```

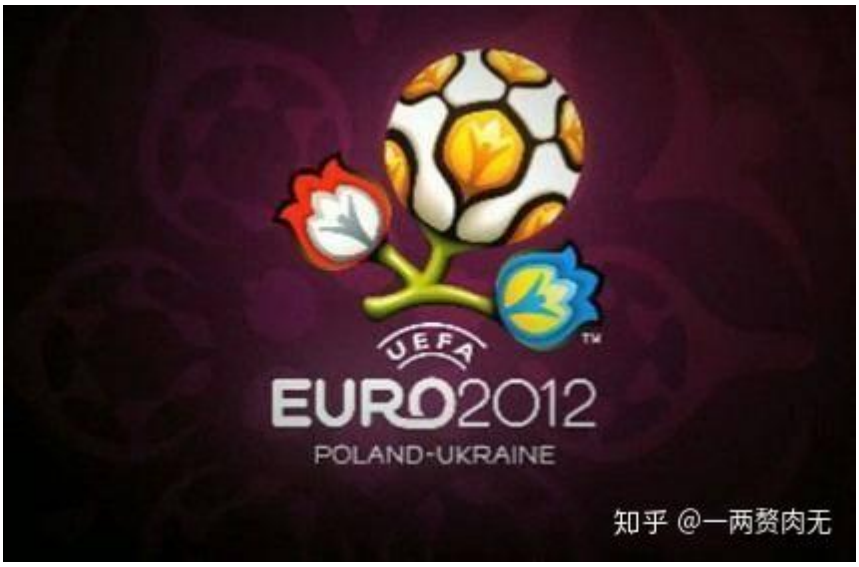
out[247]:

50

练习2-数据过滤与排序

探索2012欧洲杯数据

相应数据集：Euro2012_stats.csv



步骤1 - 导入必要的库

```
# 运行以下代码
import pandas as pd
```

步骤2 - 从以下地址导入数据集

```
# 运行以下代码
path2 = "../input/pandas_exercise/exercise_data/Euro2012_stats.csv"      #
Euro2012_stats.csv
```

步骤3 - 将数据集命名为euro12

```
# 运行以下代码
euro12 = pd.read_csv(path2)
euro12
```

out[250]:

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)	Hit Woodwork	Penalty goals	Penalties not scored	...	Saves made	Saves-to-shots ratio	Fouls Won	Fouls Conceded	Offsides	Yellow Cards	C
0	Croatia	4	13	12	51.9%	16.0%	32	0	0	0	...	13	81.3%	41	62	2	9	0
1	Czech Republic	4	13	18	41.9%	12.9%	39	0	0	0	...	9	60.1%	53	73	8	7	0
2	Denmark	4	10	10	50.0%	20.0%	27	1	0	0	...	10	66.7%	25	38	8	4	0
3	England	5	11	18	50.0%	17.2%	40	0	0	0	...	22	88.1%	43	45	6	5	0
4	France	3	22	24	37.9%	6.5%	65	1	0	0	...	6	54.6%	36	51	5	6	0
5	Germany	10	32	32	47.8%	15.6%	80	2	1	0	...	10	62.6%	63	49	12	4	0
6	Greece	5	8	18	30.7%	19.2%	32	1	1	1	...	13	65.1%	67	48	12	9	1
7	Italy	6	34	45	43.0%	7.5%	110	2	0	0	...	20	74.1%	101	89	16	16	0
8	Netherlands	2	12	36	25.0%	4.1%	60	2	0	0	...	12	70.6%	35	30	3	5	0
9	Poland	2	15	23	39.4%	5.2%	48	0	0	0	...	6	66.7%	48	56	3	7	1
10	Portugal	6	22	42	34.3%	9.3%	82	6	0	0	...	10	71.5%	73	90	10	12	0
11	Republic of Ireland	1	7	12	36.8%	5.2%	28	0	0	0	...	17	65.4%	43	51	11	6	1
12	Russia	5	9	31	22.5%	12.5%	59	2	0	0	...	10	77.0%	34	43	4	6	0
13	Spain	12	42	33	55.9%	16.0%	100	0	1	0	...	15	93.8%	102	83	19	11	0
14	Sweden	5	17	19	47.2%	13.8%	39	3	0	0	...	8	61.6%	35	51	7	7	0
15	Ukraine	2	7	26	21.2%	6.0%	38	0	0	0	...	13	76.5%	48	31	7	9	0

16 rows × 35 columns

步骤4 只选取 Goals 这一列

out[251]:

0 4

1 4

2 4

3 5

4 3

5 10

6 5

7 6

8 2

9 2

10 6

11 1

12 5

13 12

14 5

15 2

Name: Goals, dtype: int64

步骤5 有多少球队参与了2012欧洲杯?

out[252]:

16

步骤6 该数据集中一共有多少列(columns)?

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16 entries, 0 to 15
Data columns (total 35 columns):
Team                            16 non-null object
Goals                           16 non-null int64
Shots on target                 16 non-null int64
Shots off target                16 non-null int64
Shooting Accuracy               16 non-null object
% Goals-to-shots                16 non-null object
Total shots (inc. Blocked)      16 non-null int64
Hit Woodwork                    16 non-null int64
Penalty goals                   16 non-null int64
Penalties not scored            16 non-null int64
Headed goals                    16 non-null int64
Passes                          16 non-null int64
Passes completed                16 non-null int64
Passing Accuracy                16 non-null object
Touches                         16 non-null int64
Crosses                         16 non-null int64
Dribbles                        16 non-null int64
Corners Taken                   16 non-null int64
Tackles                         16 non-null int64
Clearances                      16 non-null int64
Interceptions                   16 non-null int64
Clearances off line             15 non-null float64
Clean Sheets                    16 non-null int64
Blocks                          16 non-null int64
Goals conceded                  16 non-null int64
Saves made                      16 non-null int64
Saves-to-shots ratio            16 non-null object
Fouls Won                       16 non-null int64
Fouls Conceded                  16 non-null int64
Offsides                        16 non-null int64
Yellow Cards                    16 non-null int64
Red Cards                       16 non-null int64
Subs on                         16 non-null int64
Subs off                        16 non-null int64
Players Used                    16 non-null int64
dtypes: float64(1), int64(29), object(5)
memory usage: 4.5+ KB
```

步骤7 将数据集中的列Team, Yellow Cards和Red Cards单独存为一个名叫discipline的数据框

```
# 运行以下代码
discipline = euro12[['Team', 'Yellow Cards', 'Red Cards']]
discipline
```

out[254]:

	Team	Yellow Cards	Red Cards
0	Croatia	9	0
1	Czech Republic	7	0
2	Denmark	4	0
3	England	5	0
4	France	6	0
5	Germany	4	0
6	Greece	9	1
7	Italy	16	0
8	Netherlands	5	0
9	Poland	7	1
10	Portugal	12	0
11	Republic of Ireland	6	1
12	Russia	6	0
13	Spain	11	0
14	Sweden	7	0
15	Ukraine	5	0

步骤8 对数据框discipline按照先Red Cards再Yellow Cards进行排序

```
# 运行以下代码
discipline.sort_values(['Red Cards', 'Yellow Cards'], ascending = False)
```

out[255]:

	Team	Yellow Cards	Red Cards
6	Greece	9	1
9	Poland	7	1
11	Republic of Ireland	6	1
7	Italy	16	0
10	Portugal	12	0
13	Spain	11	0
0	Croatia	9	0
1	Czech Republic	7	0
14	Sweden	7	0
4	France	6	0
12	Russia	6	0
3	England	5	0
8	Netherlands	5	0
15	Ukraine	5	0
2	Denmark	4	0
5	Germany	4	0

步骤9 计算每个球队拿到的黄牌数的平均值

```
# 运行以下代码
round(discipline['Yellow Cards'].mean())
```

out[256]:

7.0

步骤10 找到进球数Goals超过6的球队数据

```
# 运行以下代码
euro12[euro12.Goals > 6]
```

out[257]:

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)	Hit Woodwork	Penalty goals	Penalties not scored	...	Saves made	Saves-to-shots ratio	Fouls Won	Fouls Conceded	Offsides	Yellow Cards	Re Card
5	Germany	10	32	32	47.8%	15.6%	80	2	1	0	...	10	62.6%	63	49	12	4	0
13	Spain	12	42	33	55.9%	16.0%	100	0	1	0	...	15	93.8%	[illegible]	[illegible]	19	[illegible]	[illegible]

2 rows × 35 columns

步骤11 选取以字母G开头的球队数据

```
# 运行以下代码
euro12[euro12.Team.str.startswith('G')]
```

out[258]:

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)	Hit Woodwork	Penalty goals	Penalties not scored	...	Saves made	Saves-to-shots ratio	Fouls Won	Fouls Conceded	Offsides	Yellow Cards	Red Cards
5	Germany	10	32	32	47.8%	15.6%	80	2	1	0	...	10	62.6%	63	49	12	4	0
6	Greece	5	8	18	30.7%	19.2%	32	1	1	1	...	13	65.1%	67	48	12	9	1

2 rows × 35 columns

步骤12 选取前7列

```
# 运行以下代码
euro12.iloc[: , 0:7]
```

out[259]:

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)
0	Croatia	4	13	12	51.9%	16.0%	32
1	Czech Republic	4	13	18	41.9%	12.9%	39
2	Denmark	4	10	10	50.0%	20.0%	27
3	England	5	11	18	50.0%	17.2%	40
4	France	3	22	24	37.9%	6.5%	65
5	Germany	10	32	32	47.8%	15.6%	80
6	Greece	5	8	18	30.7%	19.2%	32
7	Italy	6	34	45	43.0%	7.5%	110
8	Netherlands	2	12	36	25.0%	4.1%	60
9	Poland	2	15	23	39.4%	5.2%	48
10	Portugal	6	22	42	34.3%	9.3%	82
11	Republic of Ireland	1	7	12	36.8%	5.2%	28
12	Russia	5	9	31	22.5%	12.5%	59
13	Spain	12	42	33	55.9%	16.0%	100
14	Sweden	5	17	19	47.2%	13.8%	39
15	Ukraine	2	7	26	21.2%	6.0%	38

步骤13 选取除了最后3列之外的全部列

```
# 运行以下代码
euro12.iloc[: , :-3]
```

out[260]:

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)	Hit Woodwork	Penalty goals	Penalties not scored	...	Clean Sheets	Blocks	Goals conceded	Saves made	Saves-to-shots ratio	Fouls Won	Co
0	Croatia	4	13	12	51.9%	16.0%	32	0	0	0	...	0	10	3	13	81.3%	41	62
1	Czech Republic	4	13	18	41.9%	12.9%	39	0	0	0	...	1	10	6	9	60.1%	53	73
2	Denmark	4	10	10	50.0%	20.0%	27	1	0	0	...	1	10	5	10	66.7%	25	38
3	England	5	11	18	50.0%	17.2%	40	0	0	0	...	2	29	3	22	88.1%	43	45
4	France	3	22	24	37.9%	6.5%	65	1	0	0	...	1	7	5	6	54.6%	36	51
5	Germany	10	32	32	47.8%	15.6%	80	2	1	0	...	1	11	6	10	62.6%	63	49
6	Greece	5	8	18	30.7%	19.2%	32	1	1	1	...	1	23	7	13	65.1%	67	48
7	Italy	6	34	45	43.0%	7.5%	110	2	0	0	...	2	18	7	20	74.1%	101	89
8	Netherlands	2	12	36	25.0%	4.1%	60	2	0	0	...	0	9	5	12	70.6%	35	30
9	Poland	2	15	23	39.4%	5.2%	48	0	0	0	...	0	8	3	6	66.7%	48	56
10	Portugal	6	22	42	34.3%	9.3%	82	6	0	0	...	2	11	4	10	71.5%	73	90
11	Republic of Ireland	1	7	12	36.8%	5.2%	28	0	0	0	...	0	23	9	17	65.4%	43	51
12	Russia	5	9	31	22.5%	12.5%	59	2	0	0	...	0	8	3	10	77.0%	34	43
13	Spain	12	42	33	55.9%	16.0%	100	0	1	0	...	5	8	1	15	93.8%	102	83
14	Sweden	5	17	19	47.2%	13.8%	39	3	0	0	...	1	12	5	8	61.6%	35	51
15	Ukraine	2	7	26	21.2%	6.0%	38	0	0	0	...	0	4	4	13	76.5%	48	31

16 rows × 32 columns

步骤14 找到英格兰(England)、意大利(Italy)和俄罗斯(Russia)的射正率(Shooting Accuracy)

```
# 运行以下代码
euro12.loc[euro12.Team.isin(['England', 'Italy', 'Russia']), ['Team','Shooting Accuracy']]
```

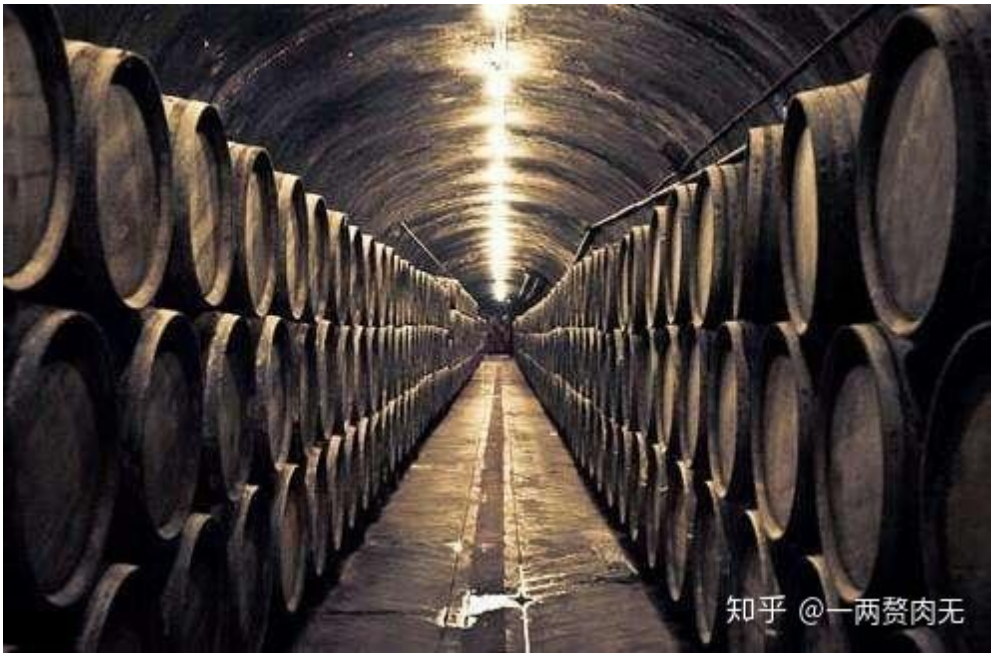
out[261]:

	Team	Shooting Accuracy
3	England	50.0%
7	Italy	43.0%
12	Russia	22.5%

练习3-数据分组

探索酒类消费数据

相应数据集：drinks.csv



步骤1 导入必要的库

```
# 运行以下代码
import pandas as pd
```

步骤2 从以下地址导入数据

```
# 运行以下代码
path3 ='../input/pandas_exercise/exercise_data/drinks.csv'    #'drinks.csv'
```

步骤3 将数据框命名为drinks

```
# 运行以下代码
drinks = pd.read_csv(path3)
drinks.head()
```

out[264]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	AS
1	Albania	89	132	54	4.9	EU
2	Algeria	25	0	14	0.7	AF
3	Andorra	245	138	312	12.4	EU
4	Angola	217	57	45	5.9	AF

步骤4 哪个大陆(continent)平均消耗的啤酒(beer)更多?

```
# 运行以下代码
drinks.groupby('continent').beer_servings.mean()
```

out[265]:

```
continent
AF      61.471698
AS      37.045455
EU     193.777778
OC      89.687500
SA     175.083333
Name: beer_servings, dtype: float64
```


步骤5 打印出每个大陆(continent)的红酒消耗(wine_servings)的描述性统计值

```
# 运行以下代码
drinks.groupby('continent').wine_servings.describe()
```

out[266]:

	count	mean	std	min	25%	50%	75%	max
continent								
AF	53.0	16.264151	38.846419	0.0	1.0	2.0	13.00	233.0
AS	44.0	9.068182	21.667034	0.0	0.0	1.0	8.00	123.0
EU	45.0	142.222222	97.421738	0.0	59.0	128.0	195.00	370.0
OC	16.0	35.625000	64.555790	0.0	1.0	8.5	23.25	212.0
SA	12.0	62.416667	88.620189	1.0	3.0	12.0	98.50	221.0

步骤6 打印出每个大陆每种酒类别的消耗平均值

```
# 运行以下代码
drinks.groupby('continent').mean()
```

out[267]:

	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol
continent				
AF	61.471698	16.339623	16.264151	3.007547
AS	37.045455	60.840909	9.068182	2.170455
EU	193.777778	132.555556	142.222222	8.617778
OC	89.687500	58.437500	35.625000	3.381250
SA	175.083333	114.750000	62.416667	6.308333

步骤7 打印出每个大陆每种酒类别的消耗中位数

```
# 运行以下代码
drinks.groupby('continent').median()
```

out[268]:

	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol
continent				
AF	32.0	3.0	2.0	2.30
AS	17.5	16.0	1.0	1.20
EU	219.0	122.0	128.0	10.00
OC	52.5	37.0	8.5	1.75
SA	162.5	108.5	12.0	6.85

步骤8 打印出每个大陆对spirit饮品消耗的平均值，最大值和最小值

```
# 运行以下代码
drinks.groupby('continent').spirit_servings.agg(['mean', 'min', 'max'])
```

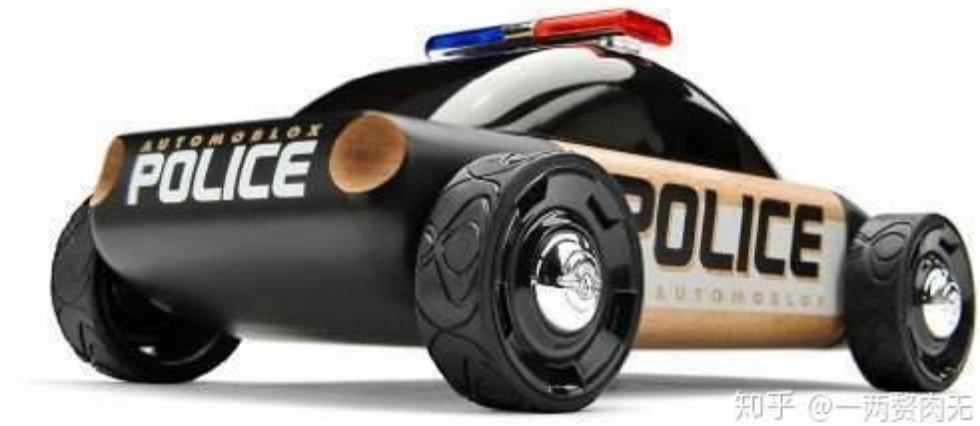
out[269]:

	mean	min	max
continent			
AF	16.339623	0	152
AS	60.840909	0	326
EU	132.555556	0	373
OC	58.437500	0	254
SA	114.750000	25	302

练习4-Apply函数

探索1960-2014美国犯罪数据

相应数据集：US_Crime_Rates_1960_2014.csv



步骤1 导入必要的库

```
# 运行以下代码
import numpy as np
import pandas as pd
```

步骤2 从以下地址导入数据集

```
# 运行以下代码
path4 = '../input/pandas_exercise/exercise_data/US_Crime_Rates_1960_2014.csv'
# "US_Crime_Rates_1960_2014.csv"
```

步骤3 将数据框命名为crime

```
# 运行以下代码
crime = pd.read_csv(path4)
crime.head()
```

out[272]:

	Year	Population	Total	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assault	Burglary	Larceny_Theft	Vehicle_Theft
0	1960	179323175	3384200	288460	3095700	9110	17190	107840	154320	912100	1855400	328200
1	1961	182992000	3488000	289390	3198600	8740	17220	106670	156760	949600	1913000	336000
2	1962	185771000	3752200	301510	3450700	8530	17550	110860	164570	994300	2089600	366800
3	1963	188483000	4109500	316970	3792500	8640	17650	116470	174210	1086400	2297800	408300
4	1964	191141000	4564600	364220	4200400	9360	21420	130390	203050	1213200	2514400	472800

步骤4 每一列(column)的数据类型是什么样的?

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55 entries, 0 to 54
Data columns (total 12 columns):
Year                  55 non-null int64
Population            55 non-null int64
Total                 55 non-null int64
Violent               55 non-null int64
Property              55 non-null int64
Murder                55 non-null int64
Forcible_Rape         55 non-null int64
Robbery               55 non-null int64
Aggravated_assault    55 non-null int64
Burglary              55 non-null int64
Larceny_Theft         55 non-null int64
Vehicle_Theft         55 non-null int64
dtypes: int64(12)
memory usage: 5.2 KB
```


注意到了吗，Year的数据类型为int64，但是pandas有一个不同的数据类型去处理时间序列(time series)，我们现在来看看。

步骤5 将Year的数据类型转换为 datetime64

```
# 运行以下代码
crime.Year = pd.to_datetime(crime.Year, format='%Y')
crime.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55 entries, 0 to 54
Data columns (total 12 columns):
Year                 55 non-null datetime64[ns]
Population           55 non-null int64
Total                55 non-null int64
Violent              55 non-null int64
Property             55 non-null int64
Murder               55 non-null int64
Forcible_Rape        55 non-null int64
Robbery              55 non-null int64
Aggravated_assault   55 non-null int64
Burglary             55 non-null int64
Larceny_Theft        55 non-null int64
Vehicle_Theft        55 non-null int64
dtypes: datetime64[ns](1), int64(11)
memory usage: 5.2 KB
```


步骤6 将列Year设置为数据框的索引

```
# 运行以下代码
crime = crime.set_index('Year', drop = True)
crime.head()
```

out[275]:

Year	Population	Total	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assault	Burglary	Larceny_Theft	Vehicle_Theft
1960-01-01	179323175	3384200	288460	3095700	9110	17190	107840	154320	912100	1855400	328200
1961-01-01	182992000	3488000	289390	3198600	8740	17220	106670	156760	949600	1913000	336000
1962-01-01	185771000	3752200	301510	3450700	8530	17550	110860	164570	994300	2089600	366800
1963-01-01	188483000	4109500	316970	3792500	8640	17650	116470	174210	1086400	2297800	408300
1964-01-01	191141000	4564600	364220	4200400	9360	21420	130390	203050	1213200	2514400	472800

步骤7 删除名为Total的列

```
# 运行以下代码
del crime['Total']
crime.head()
```

out[276]:

	Population	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assault	Burglary	Larceny_Theft	Vehicle_Theft
Year										
1960-01-01	179323175	288460	3095700	9110	17190	107840	154320	912100	1855400	328200
1961-01-01	182992000	289390	3198600	8740	17220	106670	156760	949600	1913000	336000
1962-01-01	185771000	301510	3450700	8530	17550	110860	164570	994300	2089600	366800
1963-01-01	188483000	316970	3792500	8640	17650	116470	174210	1086400	2297800	408300
1964-01-01	191141000	364220	4200400	9360	21420	130390	203050	1213200	2514400	472800


```
crime.resample('10AS').sum()
```

out[277]:

	Population	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assault	Burglary	Larceny_Theft	Vehicle_Theft
Year										
1960-01-01	1915053175	4134930	45160900	106180	236720	1633510	2158520	13321100	26547700	5292100
1970-01-01	2121193298	9607930	91383800	192230	554570	4159020	4702120	28486000	53157800	9739900
1980-01-01	2371370069	14074328	117048900	206439	865639	5383109	7619130	33073494	72040253	11935411
1990-01-01	2612825258	17527048	119053499	211664	998827	5748930	10568963	26750015	77679366	14624418
2000-01-01	2947969117	13968056	100944369	163068	922499	4230366	8652124	21565176	67970291	11412834
2010-01-01	1570146307	6072017	44095950	72867	421059	1749809	3764142	10125170	30461698	3569230
2020-01-01	0	0	0	0	0	0	0	0	0	0

步骤8 按照Year对数据框进行分组并求和

*注意Population这一列，若直接对其求和，是不正确的**

```
# 更多关于 .resample 的介绍
# (https://pandas.pydata.org/pandas-
docs/stable/generated/pandas.DataFrame.resample.html)
# 更多关于 Offset Aliases的介绍
# (http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases)
# 运行以下代码
crimes = crime.resample('10AS').sum() # resample a time series per decades



# 用resample去得到“Population”列的最大值
population = crime['Population'].resample('10AS').max()

# 更新 "Population"
crimes['Population'] = population

crimes
```

out[278]:

	Population	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assault	Burglary	Larceny_Theft	Vehicle_Theft
Year										
1960-01-01	201385000.0	4134930	45160900	106180	236720	1633510	2158520	13321100	26547700	5292100
1970-01-01	220099000.0	9607930	91383800	192230	554570	4159020	4702120	28486000	53157800	9739900
1980-01-01	248239000.0	14074328	117048900	206439	865639	5383109	7619130	33073494	72040253	11935411
1990-01-01	272690813.0	17527048	119053499	211664	998827	5748930	10568963	26750015	77679366	14624418
2000-01-01	307006550.0	13968056	100944369	163068	922499	4230366	8652124	21565176	67970291	11412834
2010-01-01	318857056.0	6072017	44095950	72867	421059	1749809	3764142	10125170	30401698	3569080
2020-01-01	NaN	0	0	0	0	0	0	0	0	0

步骤9 何时是美国历史上生存最危险的年代?

out[279]:

```
Population            2014-01-01
Violent               1992-01-01
Property              1991-01-01
Murder                1991-01-01
Forcible_Rape         1992-01-01
Robbery               1991-01-01
Aggravated_assault    1993-01-01
Burglary              1980-01-01
Larceny_Theft         1991-01-01
Vehicle_Theft         1991-01-01
dtype: datetime64[ns]
```


练习5-合并

探索虚拟姓名数据

相应数据集：练习中手动内置的数据

步骤1 导入必要的库

```
# 运行以下代码
import numpy as np
import pandas as pd
```

步骤2 按照如下的元数据内容创建数据框

```
# 运行以下代码
raw_data_1 = {
        'subject_id': ['1', '2', '3', '4', '5'],
        'first_name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
        'last_name': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}
```

```
raw_data_2 = {
        'subject_id': ['4', '5', '6', '7', '8'],
        'first_name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
        'last_name': ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']}


raw_data_3 = {
        'subject_id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
        'test_id': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]}
```

步骤3 将上述的数据框分别命名为data1，data2，data3

```
# 运行以下代码
data1 = pd.DataFrame(raw_data_1, columns = ['subject_id', 'first_name',
'last_name'])
data2 = pd.DataFrame(raw_data_2, columns = ['subject_id', 'first_name',
'last_name'])
data3 = pd.DataFrame(raw_data_3, columns = ['subject_id','test_id'])
```

步骤4 将data1和data2两个数据框按照行的维度进行合并，命名为all_data

```
# 运行以下代码
all_data = pd.concat([data1, data2])
all_data
```

out[283]:

	subject_id	first_name	last_name
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali
3	4	Alice	Aoni
4	5	Ayoung	Atiches
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

步骤5 将data1和data2两个数据框按照列的维度进行合并，命名为all_data_col

```
# 运行以下代码
all_data_col = pd.concat([data1, data2], axis = 1)
all_data_col
```

out[284]:

	subject_id	first_name	last_name	subject_id	first_name	last_name
0	1	Alex	Anderson	4	Billy	Bonder
1	2	Amy	Ackerman	5	Brian	Black
2	3	Allen	Ali	6	Bran	Balwner
3	4	Alice	Aoni	7	Bryce	Brice
4	5	Ayoung	Atiches	8	Betty	Btisan

步骤6 打印data3

out[285]:

	subject_id	test_id
0	1	51
1	2	15
2	3	15
3	4	61
4	5	16
5	7	14
6	8	15
7	9	1
8	10	61
9	11	16

步骤7 按照subject_id的值对all_data和data3作合并

```
# 运行以下代码
pd.merge(all_data, data3, on='subject_id')
```

out[286]:

	subject_id	first_name	last_name	test_id
0	1	Alex	Anderson	51
1	2	Amy	Ackerman	15
2	3	Allen	Ali	15
3	4	Alice	Aoni	61
4	4	Billy	Bonder	61
5	5	Ayoung	Atiches	16
6	5	Brian	Black	16
7	7	Bryce	Brice	14
8	8	Betty	Btisan	15

步骤8 对data1和data2按照subject_id作连接

```
# 运行以下代码
pd.merge(data1, data2, on='subject_id', how='inner')
```

out[287]:

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	4	Alice	Aoni	Billy	Bonder
1	5	Ayoung	Atiches	Brian	Black

步骤9 找到 `data1` 和 `data2` 合并之后的所有匹配结果

```
# 运行以下代码
pd.merge(data1, data2, on='subject_id', how='outer')
```

out[288]:

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	1	Alex	Anderson	NaN	NaN
1	2	Amy	Ackerman	NaN	NaN
2	3	Allen	Ali	NaN	NaN
3	4	Alice	Aoni	Billy	Bonder
4	5	Ayoung	Atiches	Brian	Black
5	6	NaN	NaN	Bran	Balwner
6	7	NaN	NaN	Bryce	Brice
7	8	NaN	NaN	Betty	Btisan

由于内容过多，以免导致篇幅过大而导致读者阅读不顺畅，练习5至练习10将会放在下期发布。