

3.9（重要）Pandas分组数据处理

1.1 需求分析及groupby()方法描述

任务：需求分析

任务：groupby() 方法描述

任务：groupby() 返回的是 DataFrameGroupBy 对象

任务：DataFrameGroupBy 分组对象调用数据分析函数

2 分组数据处理流程分析

任务：分组数据处理基本流程

任务：数据分组步骤

任务：通过list()函数显示分组对象内容

任务：数据处理+结果组合步骤

任务：串联调用方式

任务：图解“数据分割+数据处理+结果组合”过程

3 agg()、transform()、apply()数据处理操作

任务：基本groupby()原理

任务：agg()聚合操作

任务：transform()变换操作

任务：apply()应用自定义操作

4 案例：美国警方致命枪击案数据分析

任务：导入数据

任务：选择某列进行数据处理操作

任务：离散字段的数据处理

任务：可视化不同种族的逃逸方式分布

任务：可视化不同逃逸类型的年龄分布

任务：不同字段采取不同的数据处理操作

任务：同个字段采取多种的数据处理操作

任务：不同字段采用多种数据处理操作

任务：总结

3.9（重要）Pandas分组数据处理

1.1 需求分析及groupby()方法描述

任务：需求分析

知识点：

- 之前学习了如何通过聚合方法来分析 Dataframe 数据集，例如 sum() 求和、mean() 求均值、count() 计数等函数。
- 但是，上述的聚合方法过于简单，只能针对整个数据集。然而，在日常的数据分析中，经常需要根据某个或多个字段(列)将数据划分
- 为不同的群体

进行分析。

- 分别统计男生和女生的平均身高。
- 电商领域将全国的总销售额根据省份进行划分，分析各省销售额的变化情况。
- 社交领域将用户根据画像（性别、年龄）进行细分，研究用户的使用情况和偏好等。
- Pandas为 `Dataframe` 提供了 `groupby()` 方法，实现对数据集的**分组**功能。

在使用pandas进行数据分析时，`groupby()` 将会是一个数据分析辅助的利器。本节将介绍 `groupby()` 的基本原理及对应的 `agg()`、`transform()` 和 `apply()` 操作。

任务： `groupby()` 方法描述

```
1 dataframeobj.groupby(by=None, axis=0, level=None,  
2                       as_index=True, sort=True,  
3                       group_keys=True, squeeze=False, **kwargs)
```

- `by`：指定按哪个**关键列**对数据进行分组。例如，`df.groupby(['A'])`，根据 '`A`' 列进行分组。
- `axis`：指定沿 `axis=0` 或 `axis=1` 方向分组，默认是 `axis=0`。
- 其他参数都可以缺省，不常用。
- 返回：一个 `DataFrameGroupBy` 对象，储存了按关键列分组后的**子 DataFrame 集合**。
- Pandas后续将会对 `DataFrameGroupBy` 对象中的每个子 `DataFrame` 进行数据处理。

任务： `groupby()` 返回的是 `DataFrameGroupBy` 对象

[68]:

```
1 import numpy as np  
2 import pandas as pd  
3  
4 # 绘图导包  
5 %matplotlib inline  
6 %config InlineBackend.figure_format = 'retina'  
7 import matplotlib.pyplot as plt  
8 plt.style.use('ggplot')
```

[]:

```
1 df = pd.DataFrame({'A': ['a', 'c', 'b', 'c', 'a'],  
2                   'B': [0, 1, 3, 2, 3],  
3                   'C': ['T', 'F', 'F', 'T', 'F']})  
4 print(df)  
5 #    A  B  C  
6 # 0  a  0  T  
7 # 1  c  1  F  
8 # 2  b  3  F  
9 # 3  c  2  T  
10 # 4  a  3  F  
11  
12 # 返回的是一个DataFrameGroupBy对象  
13 # 选定A列作为分组关键列。  
14 g_df = df.groupby('A')
```

```
15 print(g_df)
16 # pandas.core.groupby.generic.DataFrameGroupBy
```

 image-20200502111924766

任务： DataFrameGroupBy 分组对象调用数据分析函数

知识点：

- `groupby()` 方法的目的是为了对分组数据进行后续的数据处理。
- 如果只分组，而不进行数据处理操作。那么分组将会变得毫无意义。
- Pandas为

```
1 DataFrameGroupBy
```

分组对象提供了三类数据处理操作：

- 聚合操作： `agg()` 或直接调用聚类函数(求和、均值、最大、最小、方差等)。
- 变换操作： `transform()`
- 应用自定义操作： `apply()` 。

知识点：

- 数据处理步骤：Pandas为每个分组的子 `DataFrame` 执行数据处理(聚合)操作。
- 结果合并步骤：再将数据处理的结果合并为一个 `DataFrame` 。
- Pandas将数据处理步骤和结果合并步骤整合为一个步骤。

下面将通过一个为 `DataFrameGroupBy` 分组对象，调用聚合函数来演示数据处理功能。

[]：

```
1 # 对DataFrameGroupBy对象调用均值聚合函数
2 # 对于非数值类型的，直接忽略
3 print(g_df.mean())
4 #      B
5 # A
6 # a  1.5
7 # b  3.0
8 # c  1.5
```

2 分组数据处理流程分析

任务：分组数据处理基本流程

知识点：理论上，分组数据处理分为如3步骤，数据分组、数据处理和结果组合：

1. 数据分组：根据某些条件将数据分组。
2. 数据处理：对每组独立应用数据处理函数。
3. 结果组合：将结果合并组合，返回一个新的 `Dataframe` 。

知识点：Pandas程序实现上，将数据处理和结果组合合二为一，如下：

- 步骤一(数据分组): 调用 `groupby()` 方法, 传递分组关键列, 得到分组对象 `DataFrameGroupBy` 。
- 步骤二(数据处理+结果组合): 对 `DataFrameGroupBy` 调用 `agg()`、`transform()` 和 `apply()` 操作, 得到分组处理结果。

下面通过例子, 描述分组数据分析的基本流程。

[]:

```
1 df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],
2                     'data': range(6)},
3                     columns=['key', 'data'])
4 print(df)
5 #    key  data
6 # 0    A     0
7 # 1    B     1
8 # 2    C     2
9 # 3    A     3
10 # 4    B     4
11 # 5    C     5
```

任务: 数据分组步骤

数据分组: 调用 `DataFrame` 的 `groupby()` 方法, 选择 `key` 列作为分组关键列。

[]:

```
1 # 返回DataFrameGroupBy对象
2 g_df = df.groupby('key')
3 print(g_df)
```

知识点: 返回值并非是一个 `DataFrame` 对象, 而是一个 `DataFrameGroupBy` 对象。

- `DataFrameGroupBy` 对象是分组后的子 `DataFrame` 集合。
- 注意: 在应用数据处理函数前, 不会返回具体的 `DataFrame` 数据。
- 这种方式被称为“**延迟计算**”(lazy evaluation), 使得数据处理更高效。

任务: 通过list()函数显示分组对象内容

那个生成的 `DataFrameGroupBy` 是啥呢? 对 `DataFrame` 进行了 `groupby()` 后发生了什么? Jupyter所返回的结果是对象的内存地址。这并不利于直观地理解, 为了看看这个 `DataFrameGroupBy` 内部究竟是什么, 可以把 `DataFrameGroupBy` 转换成 `list` 对象的形式来查看。

知识点:

- 通过 `list()` 函数, 将 `DataFrameGroupBy` 对象转换为列表来显示其中的内容。
- 转化为列表的

```
1 DataFrameGroupBy
```

对象

形式上

是

元组嵌套列表

。多个分组子

```
1 DataFrame
```

嵌套在集合中，具体如下：

- 每个元组对应分组子 `DataFrame`，嵌套在列表中。
- 整个 `DataFrameGroupBy` 对象，分组子 `DataFrame` 集合。
- `getsizeof()` 函数可以观察，`DataFrameGroupBy` 对象相对原 `DataFrame` 对象内存小很多。这说明pandas对分组操作的内存使用上进行了优化。

[]:

```
1 import sys
2 # 原DataFrame的内存大小
3 print(sys.getsizeof(df)) # 572
4 # 分组后的DataFrameGroupBy对象的内存大小
5 print(sys.getsizeof(g_df)) # 56
6
7 print(list(g_df))
8 # [('A', key data
9 # 0 A 0
10 # 3 A 3), ('B', key data
11 # 1 B 1
12 # 4 B 4), ('C', key data
13 # 2 C 2
14 # 5 C 5)]
```

任务：数据处理+结果组合步骤

数据处理+结果组合：对 `DataFrameGroupBy` 对象应用数据处理函数，并将结果合并为一个新的 `DataFrame` 对象。

[]:

```
1 print(g_df.sum())
2 # data
3 # key
4 # A 3
5 # B 5
6 # C 7
```

任务：串联调用方式

Pandas提供了简洁的**串联调用**方式，可将这些步骤整合为一个语句，实现**数据分割+数据处理+结果组合**操作。

[]:

```
1 # 串联调用形式(操作从左到右)
2 # df.groupby('key')返回对象，再调用sum()
3 print(df.groupby('key').sum())
4 #      data
5 # key
6 # A      3
7 # B      5
8 # C      7
```

任务：图解“数据分割+数据处理+结果组合”过程

下图给出了**数据分割+数据处理+结果组合**操作的可视化过程，其中数据处理调用了求和函数。

- 数据分割：将 `DataFrame` 按指定 `by` 参数的关键列，进行**分组**。
- 数据处理：对每个分组**应用**数据处理函数，通常是聚合操作 `agg()`、变换操作 `transform()` 和自定义操作 `apply()`。
- 结果组合：将每组的结果**组合**成一个新的 `DataFrame` 结果数据。



3 agg()、transform()、apply()数据处理操作

在本小节，将详细介绍 `groupby()` 方法的基本使用以及三类数据处理操作：

- 聚合操作：`agg()` 或直接调用聚类函数(求和、均值、最大、最小、方差等)。
- 变换操作：`transform()`
- 应用自定义操作：`apply()`。

任务：基本groupby()原理

创建 `DataFrame` 数据集，3个字段(列)包括company、salary、age，10条数据(行)。


[]:

```
1 company=["A","B","C"]
2
3 data=pd.DataFrame({
4     "company":[company[x] for x in np.random.randint(0,len(company),10)],
5     "salary":np.random.randint(5,50,10),
6     "age":np.random.randint(15,50,10)
7 })
8 )
```

在pandas中，实现分组操作的代码很简单，仅需一行代码，在这里，将上面的数据集按照 `company` 字段进行划分：

[]:

```
1 group = data.groupby("company")
2 list(group)
```

转换成列表的形式后，可以看到，列表由三个元组组成，每个元组中，第一个元素是group（这里是按照 `company` 进行分组，所以最后分为了 A ， B ， C ），第二个元素的是对应组别下的 `DataFrame` 数据，整个过程可以图解如下： 

分析：

- `groupby()` 的过程就是将原有的 `DataFrame` 按照 `groupby()` 的字段（这里是 `company` ），划分为若干个 `分组DataFrame`，被分为多少个组就有多少个 `子 DataFrame`。
- 所以说，在 `groupby` 之后的一系列操作（如 `agg`、`apply` 等），均是基于 `子 DataFrame` 的操作。
- 理解了这点，也就基本摸清了Pandas中 `groupby()` 操作的主要原理。

下面来讲讲 `groupby` 之后的常见操作。

任务：agg()聚合操作

聚合操作是 `groupby()` 之后非常常见的操作。聚合操作可以用来求和、均值、最大值、最小值等，下面的表格列出了Pandas中常见的聚合操作。

指标	描述
<code>count()</code>	计数项
<code>first()</code> 、 <code>last()</code>	第一项与最后一项
<code>mean()</code> 、 <code>median()</code>	均值与中位数
<code>min()</code> 、 <code>max()</code>	最大值与最小值
<code>std()</code> 、 <code>var()</code>	标准差与方差
<code>mad()</code>	均值绝对偏差(mean absolute deviation)
<code>prod()</code>	所有项乘积
<code>sum()</code>	所有项求和

针对之前的数据集，如果想求不同公司员工的平均年龄和平均薪水，可以先对 `company` 字段进行分组，然后再调用 `agg('mean')` 或 `mean()` 聚合方法。

[]：

```
1 g_df = data.groupby("company")
2 # 两种等价的应用聚合方式
3 print(g_df.agg('mean'))
4 print(g_df.mean())
```

如果希望针对 **不同的列求执行不同的聚合函数**，比如要计算不同公司员工的平均年龄以及薪水的中位数，可以利用字典进行聚合操作的指定。

[]：

```

1 # 通过字典，不同列执行不用聚合函数
2 result = data.groupby('company').agg({'salary':'median','age':'mean'})
3 print(result)

```

`agg()` 聚合过程可以图解如下，程序数据是随机的，因此数据上不一定一致。



任务：transform()变换操作

`transform` 是一种什么数据操作？和 `agg` 有什么区别呢？为了更好地理解 `transform` 和 `agg` 的不同，下面从实际的应用场景出发进行对比。

- 在上面的 `agg()` 中，学会了如何求不同公司员工的平均薪水。
- 如果现在需要在原数据集中新增一列 `avg_salary`，代表**员工所在的公司的平均薪水（相同公司的员工具有一样的平均薪水）**，该怎么实现呢？

如果按照正常的步骤来计算，需要先求得不同公司的平均薪水，然后按照员工和公司的对应关系填充到对应的位置，不用 `transform()` 的话，实现代码如下：

[]:

```

1 # 要学会看串联代码，从左到右，左边的返回调用右边的方法
2 # 根据'company'进行分组，索引'salary'列元素，再对该'salary'元素求mean()
3 avg_df = data.groupby('company')['salary'].mean()
4 print(avg_df)
5 # company
6 # A    22.2
7 # B    40.0
8 # C    39.0
9 # Name: salary, dtype: float64
10
11 # 将结果通过to_dict()转换为字典
12 avg_salary_dict = avg_df.to_dict()
13 print(avg_salary_dict)
14 # {'A': 22.2, 'B': 40.0, 'C': 39.0}
15
16 # 根据'company'，将字典avg_salary_dict，
17 # 通过map()映射函数，解封分组数据，填充到'avg_salary'列的对应的位置
18 data['avg_salary'] = data.groupby('company')['salary'].transform('mean')
19 print(data)

```

如果使用`transform`的话，仅需要一行代码：

[]:

```

1 # 对data关于'company'列分组，并索引'salary'列元素
2 # 通过transform('mean')方法，参数是聚合操作函数名(字符串)
3 # 将返回的'mean'聚合结果，并解封填充到'avg_salary'列中
4 result = data['avg_salary'] = data.groupby('company')['salary'].transform('mean')
5 print(result)

```


还是以图解的方式来看看进行 `groupby()` 后 `transform()` 的实现过程。为了更直观展示，图中加入了 `company` 列，实际按照上面的代码只有 `salary` 列。



图中的大方框是 `transform()` 和 `agg()` 所不一样的地方。

- 对 `agg()` 而言，会计算得到 A , B , C 公司对应的均值并直接返回。
- 但对 `transform()` 而言，则会**对每一条数据求得相应的结果，同一组内的样本会有相同的值**，组内求完均值后会**按照原索引的顺序**返回结果。
- 如果有不理解的可以拿这张图和 `agg()` 那张对比一下。

任务：apply()应用自定义操作

知识点：

- `apply()` 相比 `agg()` 和 `transform()` 而言更加灵活，能够传入任意自定义的函数，实现复杂的数据操作。
- 那在 `groupby()` 后使用 `apply()` 和之前所介绍的有什么区别呢？区别是有的，但是整个实现原理是基本一致的。
- 两者的区别在于，对于 `groupby()` 后的 `apply()`，以分组后的 **子DataFrame** 作为参数传入**自定义函数**的。

假设现在需要获取各个公司年龄最大的员工的数据，该怎么实现呢？可以用以下代码实现：

[]:

```
1 def get_oldest_staff(x):
2     df = x.sort_values(by = 'age',ascending=True)
3     return df.iloc[-1,:]
4
5 oldest_staff = data.groupby('company',as_index=False).apply(get_oldest_staff)
6
```

这样便得到了每个公司年龄最大的员工的数据，整个流程图解如下：



知识点：关于 `apply()` 的使用有个建议

- 虽然说 `apply()` 拥有更大的灵活性，但 `apply()` 的运行效率会比 `agg()` 和 `transform()` 更慢。
- 所以，`groupby()` 之后能用 `agg()` 和 `transform()` 解决的问题还是优先使用这两个方法，实在解决不了了才考虑使用 `apply()` 进行自定义函数操作。

4 案例：美国警方致命枪击案数据分析

任务：导入数据

调用 `read_csv()` 读取 `PoliceKillingsUS.csv` 数据集，这里需要指定 `path` 路径，`r` 字符串前缀代表**不转义字符**。

[69]:

```

1 path = r'input\PoliceKillingsUS.csv'
2 data = pd.read_csv(path, encoding='utf-8')
3 data.head()

```

[69]:

	name	date	race	age	signs_of_mental_illness	flee
0	Tim Elliot	02/01/15	A	53.0	True	Not fleeing
1	Lewis Lee Lembke	02/01/15	W	47.0	False	Not fleeing
2	John Paul Quintero	03/01/15	H	23.0	False	Not fleeing
3	Matthew Hoffman	04/01/15	W	32.0	True	Not fleeing
4	Michael Rodriguez	04/01/15	H	39.0	False	Not fleeing

任务：选择某列进行数据处理操作

通常，会选择数据中的某列进行分组数据处理操作。

例如，统计被不同种族 `race` 内被击毙人员年龄 `age` 的均值 `mean()`。

[]:

```

1 # 串联操作，选择age列进行聚合操作
2 # 返回Series对象
3 result = data.groupby('race')['age'].mean()
4 print(type(result))
5 print(result)

```

程序分析：

- 首先，通过 `groupby()` 得到 `DataFrameGroupBy` 对象，比如 `data.groupby('race')`。
- 然后，对 `DataFrameGroupBy` 对象通过索引来选择特定的列，比如 `['age']`。返回一个 `SeriesGroupBy` 对象，它代表每组对应一个 `Series`。
- 对 `SeriesGroupBy` 对象进行操作。比如 `.mean()`，相当于对每个组的 `Series` 求均值。

注意：如果不选列，那么第三步的操作会遍历所有列，Pandas会对能成功操作的列进行操作，最后返回的一个由操作成功的列组成的 `DataFrame`。

[]:

```

1 # 对`DataframeGroupBy`对象通过索引来选择age列
2 # 返回SeriesGroupBy对象
3 print(data.groupby('race')['age'])

```

任务：离散字段的数据处理

知识点：数值类型的字段(列)分为连续型和离散型数据。

- **连续型**：可以对该字段执行连续的数值聚合操作(求均值、求和等)，如 `age` 列。
- **离散型**：在数据分析中，常遇到**离散型**字段，比如性别，人数等。如果采用连续操作会得到无意义结果，这时就需要使用离散操作。

例如，在不同种族 `race` 内，统计是否有精神异常迹象 `signs_of_mental_illness` 的人数。这里使用 `value_counts()` 来统计。

[]:

```
1 # 串联操作，选择signs_of_mental_illness列进行聚合操作
2 result = data.groupby('race')['signs_of_mental_illness'].value_counts()
3 print(type(result))
4 print(result)
```

此时，结果是一个 `Series` 对象，可以用 `unstack()` 将它展开。

[]:

```
1 # data.groupby('race')['signs_of_mental_illness'].value_counts().unstack()
2 print(result.unstack())
```

任务：可视化不同种族的逃逸方式分布

可视化功能是 `Groupby()` 最吸引人地方，它能够很轻松地分组画图，免去手动分组遍历每个组，并自动为每个分组分颜色。绘图需要调用 `matplotlib` 模块，下面将给出调用函数的详细注释。

下面，先给出传统可视化步骤：

- 遍历每个组。
- 然后筛选不同组的数据。
- 逐个子集画条形图。

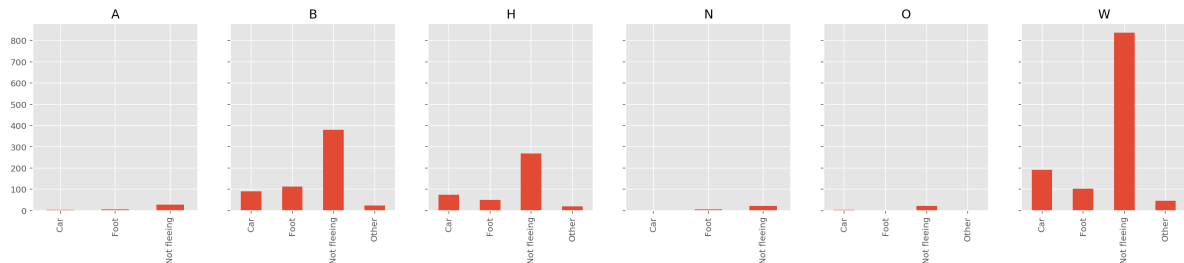
[70]:

```
1 # 需要导包matplotlib
2 # data['race'].dropna().unique() 关于race列去缺失值，并取唯一值。
3 # 然后，关于字母升序排序，返回给races列表。
4 races = np.sort(data['race'].dropna().unique())
5
6 # plt.subplots(1, len(races), figsize=(24, 4), sharey=True)
7 # 第1个参数和第2参数：共绘制1行，len(races)列个子图。
8 # figsize=(24, 4)：图的尺寸。
9 # sharey=True：子图间共享y轴坐标
10 fig, axes = plt.subplots(1, len(races), figsize=(24, 4), sharey=True)
11
12 # 遍历races每个种族
13 for ax, race in zip(axes, races):
14     # 串联调用，从左到右一个个分解
15     # data[data['race']==race]：通过掩码索引，选择值为race的行数据，返回(Dataframe)
```

```

16 # 对data[data['race']==race]进行分组数据处理, 对flee统计个数
17 res = data[data['race']==race]['flee'].value_counts()
18
19 # res.sort_index(): 对res的行标签按升序排序
20 # plot()参数: kind='bar'为柱状图;
21 # ax=ax绘图的位置(实参ax是subplots返回的子图对象)
22 # title=race 设置子图标题
23 res.sort_index().plot(kind='bar', ax=ax, title=race)

```



[]:

```

1 # 与上面一样的非注释程序
2 races = np.sort(data['race'].dropna().unique())
3 fig, axes = plt.subplots(1, len(races), figsize=(24, 4), sharey=True)
4 for ax, race in zip(axes, races):
5     # 对所有操作进行串联
6     data[data['race']==race]['flee'].value_counts().sort_index().plot(kind='bar',
7     ax=ax, title=race)

```

使用 `groupby()` 方式绘图, 避免了遍历。注意, 一定要对分组数据处理返回的 `DataFrame` 使用 `unstack()` 展开操作, 才能调用 `plot()` 绘图函数绘制每个分组的结果。

[71]:

```

1 # 输出分组数据处理结果
2 # print(data.groupby('race')['flee'].value_counts())
3 data.groupby('race')['flee'].value_counts().unstack().plot(kind='bar', figsize=
4 (20, 4))

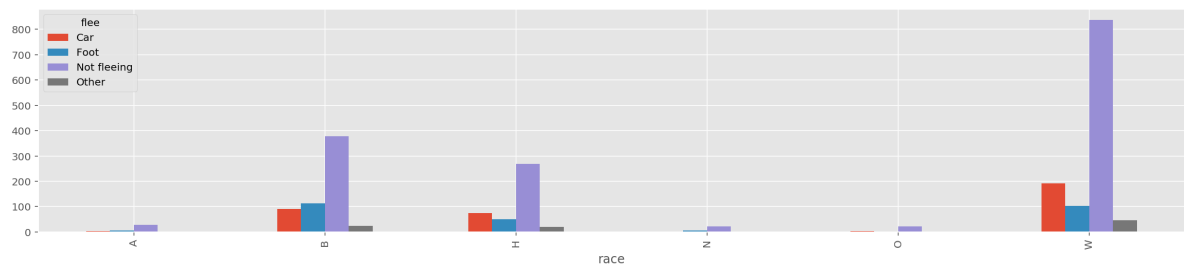
```

[71]:

```

1 <matplotlib.axes._subplots.AxesSubplot at 0x1d5aac27ef0>

```



程序分析:

- 首先, 得到分组操作后的结果 `data.groupby('race')['flee'].value_counts()` 。
- 这里有一个 `.unstack()` 操作, 返回一个 `DataFrame` 。

- 串联调用 `plot()` 绘图函数，Pandas的底层会遍历并绘制每个分组（'race'）下的 'flee' 柱状图。即 `unstack()` 后的每一行数据。
- `plot()` 参数 `kind='bar'` 绘制柱状图。

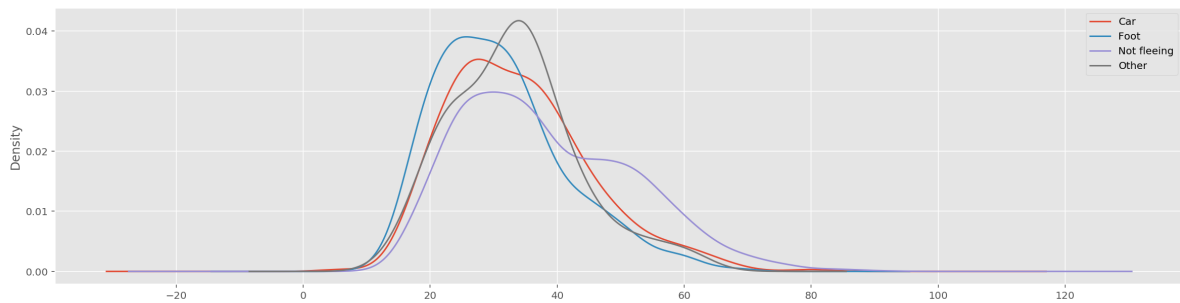
任务：可视化不同逃逸类型的年龄分布

[72]:

```
1 data.groupby('flee')['age'].plot(kind='kde', legend=True, figsize=(20, 5))
```

[72]:

```
1 flee
2 Car      AxesSubplot(0.125,0.125;0.775x0.755)
3 Foot     AxesSubplot(0.125,0.125;0.775x0.755)
4 Not fleeing AxesSubplot(0.125,0.125;0.775x0.755)
5 Other     AxesSubplot(0.125,0.125;0.775x0.755)
6 Name: age, dtype: object
```



程序分析：

- 首先，`data.groupby('flee')['age']` 返回 `SeriesGroupby` 对象。每组对应一个 `Series`。
- 依据 'flee' 逃逸类型字段进行分组，每个组包含相应的年龄数据。
- 串联调用 `plot()`，Pandas的底层会遍历并绘制每个分组(逃逸类型)下的 'age' 分布图。
- `plot()` 参数 `kind='kde'` 是连续变量的概率密度函数。

任务：不同字段采取不同的数据处理操作

例如，按 `flee` 分组，对分组中的 `age` 字段求中位数，`signs_of_mental_illness` 字段求占的比例。

注意：`signs_of_mental_illness` 字段是 `True` 和 `False` 的布尔值，平均函数将 `True` 作为1、`False` 作为0，求平均的结果是值为 `True` 的比例。

[]:

```
1 # 通过agg()聚合函数实现不同字段不同数据处理
2 opt = {'age': np.median, 'signs_of_mental_illness': np.mean}
3 data.groupby('race').agg(opt)
```

程序分析：

- `data.groupby('race')` 是依据 'race' 字段进行分组，返回 `DataFrameGroupby` 对象，每个分组对应一个 `DataFrame`。

- `agg()` 方法传入字典 `{'age': np.median, 'signs_of_mental_illness': np.mean}` , 给出了字段以及相应的聚合操作。
- `agg()` 数据处理的结果返回 `DataFrame` 对象, 每行对应一个组的结果, 每列是原 `DataFrame` 对应指定列(`'age'` 和 `'sign...illness'`)的结果。
- 每行给出每个分组中的聚合结果。例如第2行, 黑人被击毙者中, 年龄的中位数是30, 有精神疾病表现的占15%。

任务：同个字段采取多种的数据处理操作

例如, 依据 `flee` 字段分组, 对 `age` 字段求均值, 中位数, 方差。

`data.groupby('flee')['age']` 返回的是 `SeriesGroupby` 对象, 每组对应一个 `age` 的 `Series` 。

[]:

```
1 opt = [np.mean, np.median, np.std]
2 data.groupby('flee')['age'].agg(opt)
3 # 相当于分别调用下面的分组数据处理
4 # data.groupby('flee')['age'].mean()
5 # data.groupby('flee')['age'].median()
6 # data.groupby('flee')['age'].std()
```

任务：不同字段采用多种数据处理操作

综合上述两个任务的操作, 即对不同字段的采用多种数据处理。 `agg()` 参数传递列表嵌套字典。

[]:

```
1 opt = {'age': [np.median, np.mean], 'signs_of_mental_illness': np.mean}
2 data.groupby('flee').agg(opt)
```

但是返回的 `Dataframe` 中存在多个层级的列。可以通过重名列来解决此问题。

[]:

```
1 opt = {'age': [np.median, np.mean], 'signs_of_mental_illness': np.mean}
2 agg_df = data.groupby('flee').agg(opt)
3
4 # 遍历列名, 发现是以元组形式储存多级列名
5 # 因此, 可以对元组元素进行拼接合并
6 print([col for col in agg_df.columns.values])
7 # [('age', 'median'), ('age', 'mean'), ('signs_of_mental_illness', 'mean')]
8
9 # '_' .join(col).strip(), 列名元组进行拼接, 以_为连接符
10 # 重命名列名
11 agg_df.columns = ['_' .join(col).strip() for col in agg_df.columns.values]
12 agg_df
```

任务：总结

知识点：分组数据处理可总结为如下3个步骤：

1. 数据分组：将 `Dataframe` 数据按某个字段进行分组，得到 `DataFrameGroupby` 或 `SeriesGroupby` 对象。
2. 数据处理：对每组**独立**应用数据处理函数。
3. 结果组合：将结果**合并组合**，返回一个新的 `Dataframe` 或 `Series` 或可视化绘图。