第8回
データの集計と可視化1

pandasを用いた単純集計、クロス集計

目次

事例:アンケートデータの集計

- pandasモジュールを使った、簡単なアンケートデータ(質的変数、量的変数)の集計について試します。
- デスクトップ上に用意したデータファイルを使用します。
  - 2つのファイル「python.txt」「python.xlsx」を開いて内容を確認します。前者はテキスト形式、後者はExcel形式のファイルで内容は同一です。
  - 2つのファイルは下記のPython講座のサイトにも掲載しています。
    http://pana4405.u-shizuoka-ken.ac.jp/pp202102

	A	B	C	D	E
1	性別	年代	職業	満足度	消費金額
2	女性	20代	学生	満足	¥7,400
3	女性	20代	学生	とても満足	¥4,000
4	男性	60代	無職	ふつう	¥9,300
5	女性	50代	会社員	やや満足	¥5,900
6	女性	20代	会社員	ふつう	¥9,400
7	女性	20代	パート・ア	やや満足	¥6,900
8	女性	40代	会社員	とても満足	¥6,500

質的変数

量的変数

python.xlsxの例

変数と尺度

- データ集計で扱う変数は、4つの尺度に分けて考えることができる。

変数	尺度	大小	差	比	説明	例
質的変数	名義尺度	×	×	×	単に分類するためのラベル名	性別
	順序尺度	○	×	×	分類の大小関係や順序のみに意味を持つ変数	順位、満足度
量的変数	間隔尺度	○	○	×	データの間隔や差に意味はあるが、0が相対的な量のため2が1の2倍とは言えない変数	摂氏
	比例尺度	○	○	○	0が無い状態を表しており、差だけでなく比も意味を持つ変数	長さ、年齢

	A	B	C	D	E
1	性別	年代	職業	満足度	消費金額
2	女性	20代	学生	満足	¥7,400
3	女性	20代	学生	とても満足	¥4,000
4	男性	60代	無職	ふつう	¥9,300
5	女性	50代	会社員	やや満足	¥5,900
6	女性	20代	会社員	ふつう	¥9,400
7	女性	20代	パート・ア	やや満足	¥6,900
8	女性	40代	会社員	とても満足	¥6,500

※年代などの幅のある区間として扱われる変数＝順序尺度
※満足度は、とても満足＝7、満足＝6、やや満足＝5、ふつう＝4・・・、などと数値に置き換えたとしても、各評定値が等間隔とは限らないので、順序尺度
※順序尺度を便宜的に「等間隔」であるとみなし、間隔尺度として分析する場合もある

データの取り込み(テキストファイル)

- データをテキストファイル(csv, tsv)から取り込む方法について試します。
- 手順
  - Spyderを起動し、[新規]-[新規ファイル]を作成します。
  - pandasによってテキストファイルを取り込むには、read_csvメソッドを使います。その際、区切り文字を引数sepで指定します。
    - csv(カンマ区切り)ならば","
    - tsv(タブ区切り)ならば"¥t"(または、"\t")
  - ファイル名にURLを記述すれば、ネット上のファイルも取り込めます。
  - 取り込んだデータは、DataFrame型という行列形式のデータ型として管理されます。また、1行目は自動的にラベルとして認識されます。

DataFrame型

pandasのインポート

```
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams["font.family"] = "IPAexGothic"

data = pd.read_csv("~/Desktop/python.txt", sep="\t")
print(data)
```

「~」は自分のホームフォルダという意味

データの取り込み(Excelファイル)

- pandasでは、テキストファイルだけでなくExcelファイルを直接読み込むことができます。
  - ExcelFileメソッドでExcelファイルを開く。
  - 開いたファイルのparseメソッドでシート名を指定して取り込む。

```
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams["font.family"] = "IPAexGothic"

file = pd.ExcelFile("~/Desktop/python.xlsx")
data = file.parse("アンケート")
print(data)
```

「アンケート」シートの取り込み

取り込んだデータには自動的に連番のindexが割り当てられます。

	性別	年代	職業	満足度	消費金額
0	女性	20代	学生	満足	7400
1	女性	20代	学生	とても満足	4000
2	男性	60代	無職	ふつう	9300
3	女性	50代	会社員	やや満足	5900

特定の行や列を取り出す

- pandasのDataFrameでは特定の行や列を取り出すことができます。

0行目
1行目

	性別	年代	職業	満足度	消費金額
0	女性	20代	学生	満足	7400
1	女性	20代	学生	とても満足	4000

列名

  - 特定の列を取り出す
    - データフレーム.列名
  - インデックス名やラベル名を指定して取り出す
    - データフレーム.loc[行インデックス名, 列ラベル名]
  - 番号を指定して取り出す
    - データフレーム.iloc[行番号, 列番号]

取り出した特定の行や列は、Series型となります

行指定の0を「:」とすると、全行を取り出せます。

```
#性別の列を取り出し
print(data.性別)
#0行目の性別を取り出し
print(data.loc[0, "性別"])
#0行目の性別と満足度を取り出し(リストで指定)
print(data.loc[0, ["性別","満足度"]])
#0行0列を取り出し
print(data.iloc[0, 0])
#0行の0列から3列までを取り出し(スライスで指定)
print(data.iloc[0, 0:4])
#0行の1列と3列を取り出し
print(data.iloc[0, [1, 3]])
#性別が男性の行だけ取り出し
print(data[data.性別 == "男性"])
```


質的変数の単純集計(1)

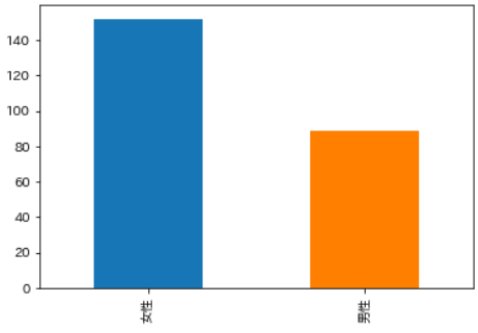
- value_countsメソッドにより、質的変数の項目ごとに単純集計を実行できます。
  - 構文　データフレーム.value_counts()
- 集計により新たなDataFrame(Series)が生成され、そのDataFrame(Series)を元に縦棒グラフを描画します。
  - 構文　データフレーム.plot.bar()

集計結果は新たなデータフレームとなる

```
sei = data.性別.value_counts()
print(sei)
sei.plot.bar()
```

```
女性    152
男性     89
```

pandasでは内部的にmatplotlibを利用してグラフが描画できる



質的変数の単純集計(2)

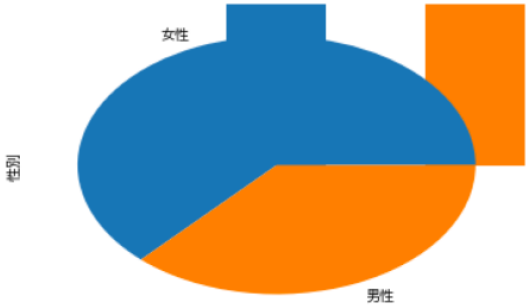
- 棒グラフbar以外にも、円グラフpieなど様々な種類のグラフを描くことができます。
  - bar（縦棒グラフ）、barh（横棒グラフ）、pie（円グラフ）、scatter（散布図）、line（折れ線グラフ）など

- 複数のグラフを一度に描きたい場合は、単純にplotメソッドを並べただけでは、グラフが重なって描かれてしまいます。
- 重ならなくするには、plt.show()で描画を一旦リセットします。

```
sei.plot.bar()
sei.plot.pie()
```

修正

```
sei.plot.bar()
plt.show()
sei.plot.pie()
plt.show()
```

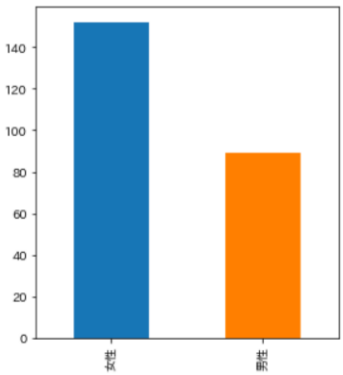


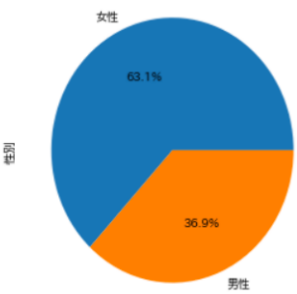
質的変数の単純集計(3)

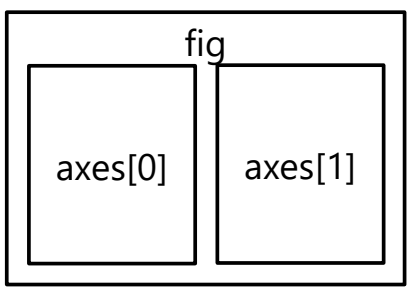
- 複数のグラフを並べたい場合は、あらかじめmatplotlibのpyplotで、subplotsメソッドを使いグラフをいくつ並べるか設定しておき、軸（axes）を指定してグラフを描きます。

```
sei = data.性別.value_counts()
print(sei)
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
sei.plot.bar(ax=axes[0])
sei.plot.pie(ax=axes[1], autopct="%.1f%%")
```

グラフの行数
グラフの列数
グラフ全体のサイズ（インチ）
軸の指定
円グラフに割合を表示（小数点以下1位まで%つきで表示）







質的変数の単純集計(4)

- その他、様々なグラフの調整が可能です。

文字サイズ

タイトルや配色の指定（r, g, b, c, m, y, k, wや、#ff0000など16進数のカラーコード）が可能

```
plt.rcParams["font.size"] = 15
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
fig.tight_layout()
sei.plot.bar(ax=axes[0], title="性別", color=["r", "b"])
sei.plot.pie(ax=axes[1], autopct="%.1f%%")
plt.sca(axes[0])
plt.xticks(rotation=0, color="red")
plt.ylabel("人")
plt.legend()
plt.show()
```

現在の設定対象をaxes[0]に指定（グラフが1つだけの場合は不要）

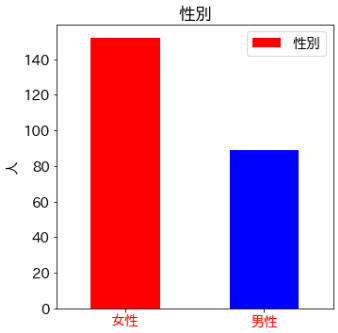
X軸の文字の調整

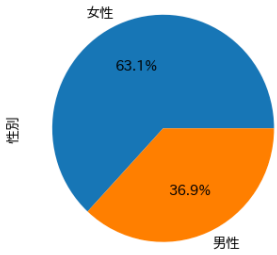
Y軸の軸ラベルの指定

凡例の表示

設定の反映

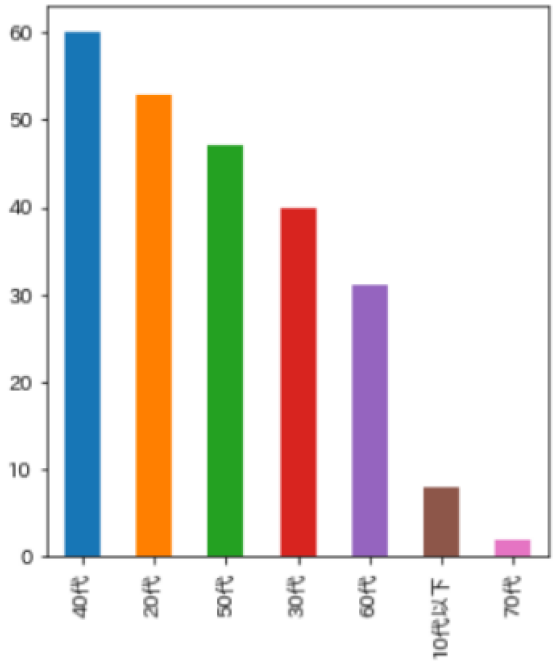
グラフ間の余白を詰める

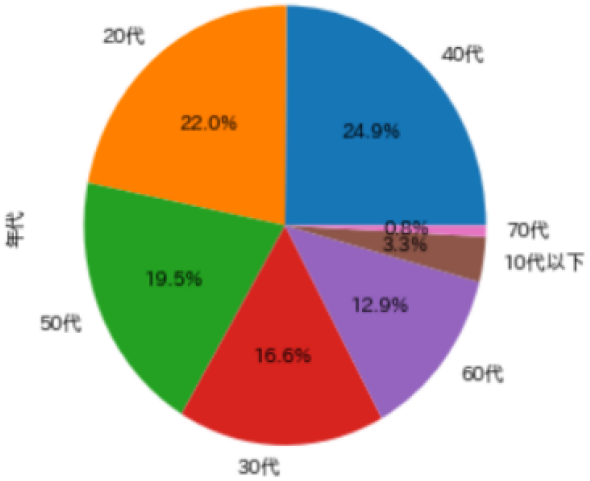




課題8

- 年代、職業、満足度についてもコードを修正してグラフを描画してください。





関数による効率化(1)

- グラフを描く処理を関数にしておくと、関数を呼び出すだけでいつも同じフォーマットのグラフが描けて便利です。
- 円グラフを描くpie関数の例
  - デスクトップ上のpie.txtファイルに、記述例を用意しましたので、ファイルを開いてコードを全て選択＆コピーし、データファイルを読み込むコードの下に貼ってください。

pie.txtの内容を貼る

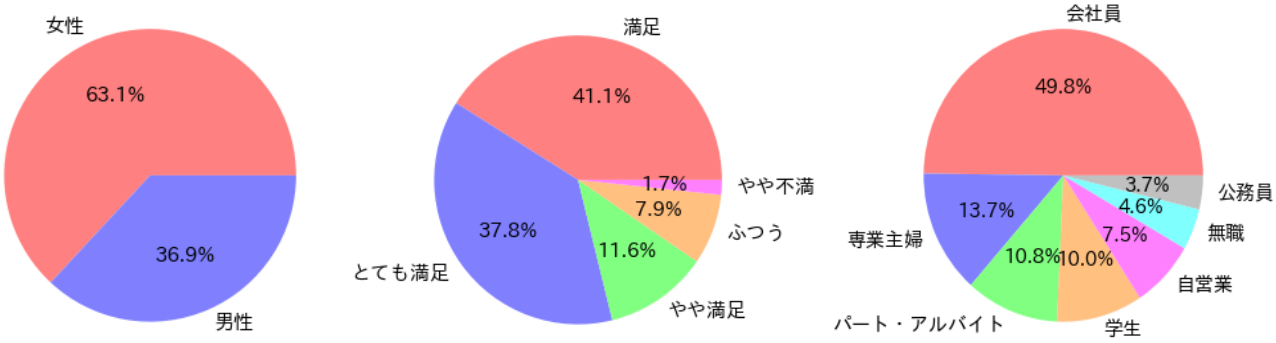
```
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams["font.family"] = "IPAexGothic"

file = pd.ExcelFile("~/Desktop/python.xlsx")
data = file.parse("アンケート")
print(data)
```

```
#円グラフ
def pie(data, column, fontsize=15, w=5, h=5):
    plt.rcParams['font.size'] = fontsize
    ret = data[column].value_counts()
    ret.plot.pie(autopct="%.1f%%", figsize=(w, h), wedgeprops={'linewidth': 0}, \
    label="", \
    colors=("#ff8080", "#8080ff", "#80ff80", "#ffc080", "#ff80ff", "#80ffff", "#c0c0c0",
    plt.show()
    kensu = data[column].count()
    ret.loc["総計"] = kensu
    final = pd.DataFrame(ret)
    final["割合"] = final[column].map(lambda x: '{:.01f}'.format(x / kensu * 100) + "%")
    print(final)
```


関数による効率化(2)

- pie関数の呼び出し方
  - pie(データフレーム, 列名, フォントサイズ, グラフの幅, グラフの高さ)
    - 最後の3つの引数はオプション

```
pie(data, "性別")
pie(data, "満足度")
pie(data, "職業")
```



量的変数の単純集計

- 量的変数については、和（sum）、平均値（mean）、最小値（min）、最大値（max）などの集計が可能です。
- 基本統計量の一括計算がdescribeメソッドで可能です。
- 量的変数に対してヒストグラムの描画がplot.histメソッドで可能です。

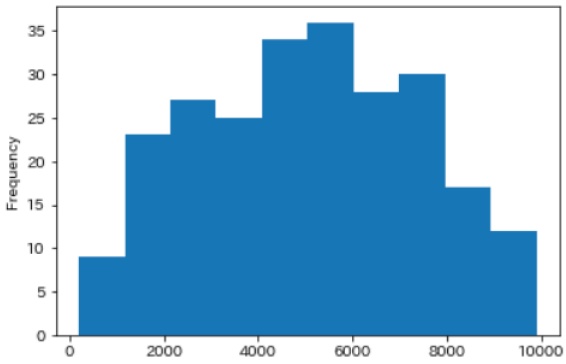
```
print(data.消費金額.sum())
print(data.消費金額.mean())
print(data.消費金額.min())
print(data.消費金額.max())
print(data.消費金額.describe())
data.消費金額.plot.hist(bins=10)
plt.show()
```

和や平均値の計算

基本統計量

binsでヒストグラムの区分数を指定

```
1211530
5027.095435684647
200
9900
count     241.000000
mean     5027.095436
std      2351.373328
min       200.000000
25%      3200.000000
50%      5100.000000
75%      6900.000000
max      9900.000000
```



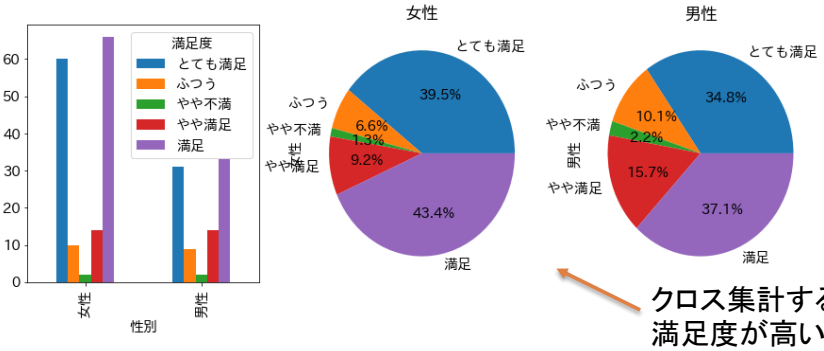
クロス集計

- 2つの項目を行と列に割り当てて集計するクロス集計は、pandasのcrosstabメソッドで簡単に実行できます。

```
cross = pd.crosstab(data.性別, data.満足度)
print(cross)
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))
cross.plot.bar(ax=axes[0])
cross.loc["女性"].plot.pie(ax=axes[1], autopct="%.1f%%", title="女性")
cross.loc["男性"].plot.pie(ax=axes[2], autopct="%.1f%%", title="男性")
plt.show()
```

stacked=Trueを追記すると積み上げ型に

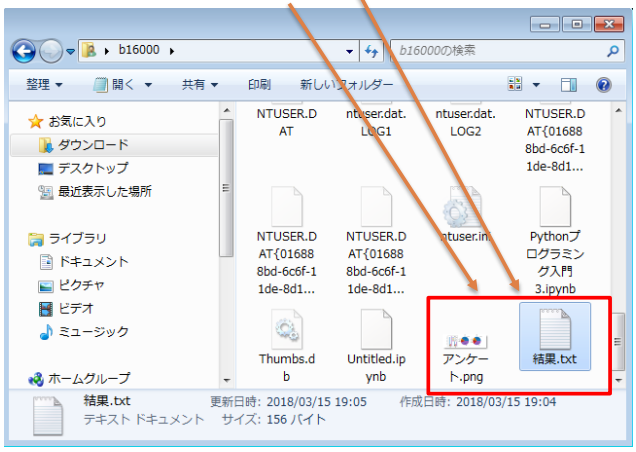
```
満足度  とても満足  ふつう  やや不満  やや満足  満足
性別
女性        60    10     2     14    66
男性        31     9     2     14    33
```



クロス集計することで、女性の方が満足度が高いことがわかる。

結果の保存

- matplotlibで描いた図は、plt.savefigメソッドで画像として保存できます。
  - ただし、plt.showメソッドの前に書く必要あります。
  - 例：`plt.savefig("アンケート.png")`
- pandasのDataFrameは、to_csvメソッドでcsvファイルとして保存できます。
  - 例：`cross.to_csv("結果.txt")`



(補足)クロス集計(1)

- pandasのcrosstabメソッド以外に、DataFrameのpivot_tableメソッドでクロス集計することもできます。下記の例では、消費金額の合計値を集計しています。

集計項目　行　列

```
cross = data.pivot_table(values="消費金額", index="性別", columns="年代",
aggfunc="sum")
print(cross)
```

```
年代     10代以下      20代      30代      40代      50代      60代     70代
性別
女性  27300.0  174910.0  125400.0  220300.0  148000.0   54500.0      NaN
男性  14000.0   81600.0   78700.0   72500.0   94100.0  116220.0  4000.0
```

  - 値が存在しない項目にはNaNが表示されますが、fill_value=0を設定することで、強制的に0で埋めることができます。

```
cross = data.pivot_table(values="消費金額", index="性別", columns="年代",
aggfunc="sum", fill_value=0)
print(cross)
```

NaNを0で埋める

```
年代  10代以下   20代   30代   40代   50代   60代  70代
性別
女性  27300  174910  125400  220300  148000  54500     0
男性  14000   81600   78700   72500   94100  116220  4000
```

(補足)クロス集計(2)

- crosstabやpivot_table以外に、同じ値を持つ行をグループ化するgroupbyメソッドによっても項目ごとの集計が可能です。
  - groupbyの場合には、列に並ぶ項目は無く、すべて行方向に並ぶ形になります。

性別と年代でグループ化

```
print(data.groupby(["性別", "年代"]).消費金額.sum())
```

消費金額の和を計算

```
性別  年代
女性  10代以下     27300
      20代       174910
      30代       125400
      40代       220300
      50代       148000
      60代        54500
男性  10代以下     14000
      20代        81600
      30代        78700
      40代        72500
      50代        94100
      60代       116220
      70代         4000
```

下記のpivot_tableでも同じ結果が得られます。

```
print(data.pivot_table(values="消費金額",
index=["性別", "年代"], aggfunc="sum",
fill_value=0))
```

列ではなく全て行に項目を指定

(補足)Seriesの各要素の値の加工(1)

- DataFrameの各行や各列はSeries型となる
- Series型の各要素の値を、あるルールで加工したい場合は？
  - 例:消費金額の各要素に、消費税8%分を加えたい
  - mapメソッド、applyメソッドで各要素に関数を適用すれば実現できる

```
def plus8(x):
    return int(x * 1.08)

print(data.消費金額.map(plus8))
```

各要素にplus8関数が適用される

applyでもOK

```
0    3000
1    2500
2    2400
3    5900
4    5100
5    6900
6    6500
```

8%up

```
0    3240
1    2700
2    2592
3    6372
4    5508
5    7452
6    7020
```

- ラムダ式を使うと関数を定義しなくても同じことができる(無名関数)
  - 構文・・・ lambda 各要素の値をとる変数 : 変数に対する処理

```
print(data.消費金額.map(lambda x: int(x * 1.08)))
```

- mapやapplyを実行しても元のSeriesは変更されないが、上書きすれば変更できる

```
data.消費金額 = data.消費金額.map(lambda x: int(x * 1.08))
print(data.消費金額)
```

(補足)Seriesの各要素の値の加工(2)

- 要素が文字列の場合は、str.文字列メソッドが使用できる
  - 例:満足度の「ふつう」を「普通」と漢字に置換する

```
print(data.満足度.str.replace("ふつう", "普通"))
```

  - その他
    - str.strip() ・・・空白の削除
    - str.lower()・・・小文字に
    - str.upper()・・・大文字に
    - ・・・

- mapでディクショナリを引数として指定すると、複数種類の置換が可能(質的変数の量的変数への置き換えに便利)
  - 例:男性を0、女性を1に変換

```
print(data.性別.map({"男性": 0, "女性": 1}))
```