第3回
pandasによるデータの集計と可視化2

目次

関数によるグラフ作成の自動化

- Pythonのpandasで集計して結果をもとに報告書用のグラフを描きたい

課題

- クオリティの高いグラフを描くためには、多くのコードを書く必要があり面倒では？（Excelの方が簡単では？）

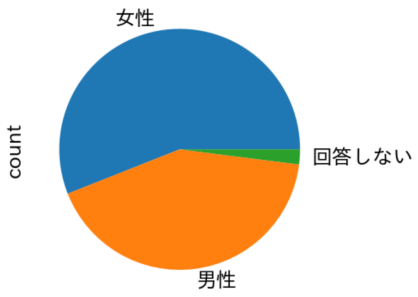
解決策

- グラフを描く処理を関数にしておくと、関数を呼び出せばいつも同じフォーマットのグラフが描けて便利

質的変数（単数回答）集計の関数化（1）

- 質的変数（単数回答）を集計し、円グラフを描く関数を定義します。
- 手順
  - 新しいコードセルを追加します。
  - 下記のpie関数を定義します。
  - 定義したpie関数を呼び出してみます。

```
#pie関数（データフレーム変数，集計したい列名，フォントサイズ）
def pie(data, col, fsize=18):
  plt.rcParams["font.size"] = fsize #フォントサイズの設定

  ret = data[col].value_counts() #集計
  ret.plot.pie() #円グラフを描く
  plt.show() #グラフ描画の確定

#関数を呼び出す
pie(data, "あなたの性別は？")
pie(data, "あなたの年代は？")
```



質的変数(単数回答)集計の関数化(2)

- 円グラフを描く際に、パーセント表示や配色の設定を追記してみましょう。

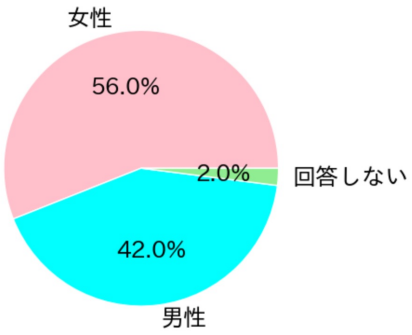
```
[23]  #pie関数（データフレーム変数，集計したい列名，フォントサイズ）
      def pie(data, col, fsize=18):
        plt.rcParams["font.size"] = fsize #フォントサイズの設定

        ret = data[col].value_counts() #集計
        ret.plot.pie(autopct="%.1f%%",
                     wedgeprops={"linewidth": 1, "edgecolor": "white"},
                     label = "",
                     colors=("pink", "cyan", "lightgreen", "yellow", "orange", "lightgray", "gray"))
        plt.show() #グラフ描画の確定

      #関数を呼び出す
      pie(data, "あなたの性別は？")
      pie(data, "あなたの年代は？")
```

パーセント表示の追加

境界線の太さと色

グラフ左のcountを消す

円グラフの色分け



質的変数(単数回答)集計の関数化(3)

- 集計表も出力できるように関数に追記しましょう。
  - 追記部分は赤枠

```
[25]  #pie関数（データフレーム変数，集計したい列名，フォントサイズ）
      def pie(data, col, fsize=18):
        plt.rcParams["font.size"] = fsize #フォントサイズの設定

        ret = data[col].value_counts() #集計
        ret.plot.pie(autopct="%.1f%%",
                     wedgeprops={"linewidth": 1, "edgecolor": "white"},
                     label = "",
                     colors=("pink", "cyan", "lightgreen", "yellow", "orange", "lightgray", "gray"))
        plt.show() #グラフ描画の確定

        kensu = data[col].count() #回答件数
        ret["総計"] = kensu         #総計行を追加
        final = pd.DataFrame(ret)     #集計結果をデータフレームに変換
        #割合の列を追加して、mapメソッドのラムダ式で割合を計算
        final["割合"] = final["count"].map(lambda x: '{:.01f}'.format(x / kensu * 100) + "%")
        display(final) #集計表をdisplay関数で整形して出力

      #関数を呼び出す
      pie(data, "あなたの性別は？")
      pie(data, "あなたの年代は？")
```

コードの解説(1)

```
kensu = data[col].count() #回答件数
```

- value_countsメソッドは、回答内容別に件数をカウントするメソッド(男性が何人、女性が何人など)でしたが、countメソッドは単純に有効回答数をカウントします(今回のデータでは50)。

```
ret["総計"] = kensu          #総計行を追加
```

- retはSeries型の集計結果の変数です。Series型の変数に存在しない項目名を指定して代入すると、その項目名が新規追加されます。

あなたの年代は？	
30代	14
20代	12
40代	9
50代	7
10代以下	5
60代	2
70代以上	1

ret(総計行を追加前)

あなたの年代は？	
30代	14
20代	12
40代	9
50代	7
10代以下	5
60代	2
70代以上	1
総計	50

ret(総計行を追加後)

コードの解説(2)

```
final = pd.DataFrame(ret)      #集計結果をデータフレームに変換
```

- Series型の変数であるretを、DataFrame型の変数であるfinalに変換しています。
- なぜ変換するのかというと、新たに「割合」の列を追加したいからです。Series型は1次元形式なので列が追加できず、2次元形式のDataFrame型への変換が必要です。

あなたの年代は？	
30代	14
20代	12
40代	9
50代	7
10代以下	5
60代	2
70代以上	1
総計	50

ret
Series型

あなたの年代は？	count
30代	14
20代	12
40代	9
50代	7
10代以下	5
60代	2
70代以上	1
総計	50

final
DataFrame型

「count」列の右に「割合」の列を追加したい

コードの解説(3)

```
#割合の列を追加して、mapメソッドのラムダ式で割合を計算
final["割合"] = final["count"].map(lambda x: '{:.01f}'.format(x / kensu * 100) + "%")
```

- final変数の「count」列の値を元に、「割合」列の値を計算しています。
- DataFrame型の変数に存在しない列名を指定して代入すると、その列名の列が新規追加されます。
- 割合の計算にはmapメソッドを使用しています(解説は次ページ)。

あなたの年代は？	count
30代	14
20代	12
40代	9
50代	7
10代以下	5
60代	2
70代以上	1
総計	50

final(割合列を追加前)

あなたの年代は？	count	割合
30代	14	28.0%
20代	12	24.0%
40代	9	18.0%
50代	7	14.0%
10代以下	5	10.0%
60代	2	4.0%
70代以上	1	2.0%
総計	50	100.0%

final(割合列を追加後)

mapとapply

- Series型やDataFrame型変数の各列(各行)のすべてに変換処理を適用して変換後の結果を取得したい場合にmapやapplyメソッドが利用できます。
  - 例:Series型の各要素に関数やラムダ式を適用

```
#各要素に関数を適用(map)
def test(x):
  return x + "です"
ret = data["あなたの性別は?"].map(test)
display(ret)

#各要素にラムダ式を適用(map)
ret = data["あなたの性別は?"].map(lambda x: x + "です")
display(ret)

#各要素にラムダ式を適用(apply)
ret = data["あなたの性別は?"].apply(lambda x: x + "です")
display(ret)
```

- mapとapplyの違い
  - mapは常にSeries型を返しますが、applyは必要な場合はDataframe型を返します。
- displayメソッド:printメソッドに似ていますが、結果をより見やすく表示してくれます。

文字列のフォーマット

- 下記の赤枠の部分では、mapメソッドとラムダ式を使用して割合を計算した結果から、文字列のformatメソッドにより小数点以下１桁のパーセント表記に変換しています。

```
#割合の列を追加して、mapメソッドのラムダ式で割合を計算
final["割合"] = final["count"].map(lambda x: '{:.01f}'.format(x / kensu * 100) + "%")
```

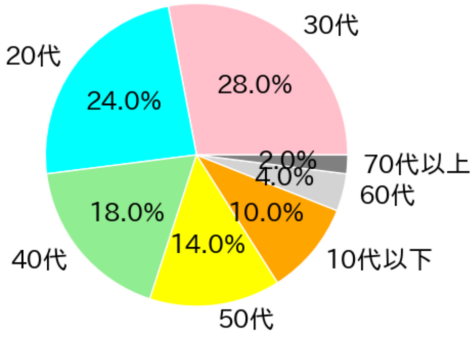
- formatメソッド
  - 文字列に値を当てはめて整形するためのメソッド
  - 構文

    ‘{値のインデックス番号:.小数点以下の桁数 型}…{…}’.format（値1, 値2, …）

    ※値が1つの場合は、インデックス番号は省略できます。
    ※型としては、整数の場合はd、実数の場合はf、文字列の場合はs（省略可）を指定します。
  - 例：
    - コード・・・'{0}は{1:.01f}です'.format("円周率", 3.141592)
    - 変換結果・・・円周率は3.1です

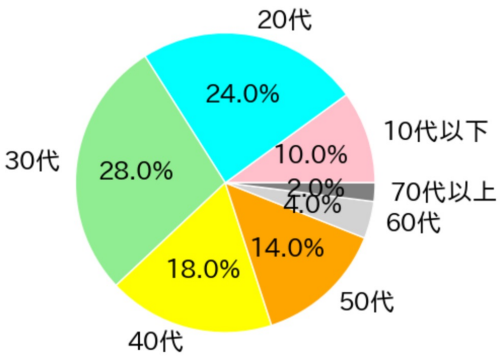
（補足）さらにブラッシュアップ（1）

- 年代の集計結果では、件数の多い順にソートされていて見づらい
  - 項目名（10代、20代など）でソートできないか？



あなたの年代は？	count	割合
30代	14	28.0%
20代	12	24.0%
40代	9	18.0%
50代	7	14.0%
10代以下	5	10.0%
60代	2	4.0%
70代以上	1	2.0%
総計	50	100.0%

件数順



あなたの年代は？	count	割合
10代以下	5	10.0%
20代	12	24.0%
30代	14	28.0%
40代	9	18.0%
50代	7	14.0%
60代	2	4.0%
70代以上	1	2.0%
総計	50	100.0%

項目名順

(補足)さらにブラッシュアップ(2)

- 項目名で昇順にソートできるように関数を修正しましょう。
  - 追記部分は赤枠

```
#pie関数（データフレーム変数，集計したい列名，フォントサイズ）
def pie(data, col, fsize=18, sort_index=False):
  plt.rcParams["font.size"] = fsize #フォントサイズの設定

  ret = data[col].value_counts() #集計

  #項目名でソートするか否か
  if sort_index == True:
    ret = ret.sort_index()

  ret.plot.pie(autopct="%.1f%%",
               wedgeprops={"linewidth": 1, "edgecolor": "white"},
               label = "",
               colors=("pink", "cyan", "lightgreen", "yellow", "orange", "lightgray", "gray"))
  plt.show() #グラフ描画の確定
```

sort_indexメソッドで、項目名を元に行をソートしています

  - 関数呼び出し時に項目名でのソートを指定

```
#関数を呼び出す
pie(data, "あなたの性別は？")
pie(data, "あなたの年代は？", sort_index=True)
```


質的変数(複数回答)の単純集計(1)

- 複数回答の場合は、一つのセルに回答項目がカンマ区切りで入力されている

今回の静岡観光の目的は？（複数選択可）
温泉, まち歩き
グルメ, ショッピング, 温泉
グルメ, ショッピング, まち歩き, 美術館・博物館等
温泉, ドライブ・ツーリング
グルメ, ショッピング, 名所旧跡の観光, テーマパーク

- このままでは集計しづらいので、下記のような横方向に項目名が並び、選択した場合は1、しなかった場合は0の表に変換する

温泉	まち歩き	グルメ	ショッピ	温泉	美術館・	ドライブ	名所旧跡	テーマパーク
1	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0
0	1	1	1	0	1	0	0	0
1	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	1	1
1	0	0	0	0	1	0	0	0

質的変数(複数回答)の単純集計(2)

- 新しいコードセルを追加して、下記を実行しましょう。

```
df = data["今回の静岡観光の目的は?(複数選択可)"].str.split(',\s*').apply(lambda x: pd.Series(1,index=x))
df = df.fillna(0).astype(int)
display(df)
```

- 結果

	温泉	まち歩き	グルメ	ショッピング	美術館・博物館等	ドライブ・ツーリング	名所旧跡の観光	テーマパーク	自然鑑賞	行祭事・イベント	各種体験
0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0
2	0	1	1	1	1	0	0	0	0	0	0
3	1	0	0	0	0	1	0	0	0	0	0
4	0	0	1	1	0	0	1	1	0	0	0
5	1	0	0	0	1	0	0	0	1	1	0
6	1	0	1	1	1	0	0	0	0	0	0
7	1	1	1	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0	1	0
9	0	1	1	1	0	0	0	0	1	0	0

コードの解説(1)

```
df = data["今回の静岡観光の目的は?(複数選択可)"].str.split(',\s*').apply(lambda x: pd.Series(1,index=x))
```

- 複数回答を上の行から1つずつ取り出して、strアクセサによって文字列型に変換し、splitメソッドでカンマごとにリストの要素に分割する
  - strアクセサを使うと、データの各要素を文字列に変換して文字列メソッドが適用できる
  - 今回は、splitメソッドで文字列をカンマと任意文字数のスペース(＼s*)で区切ってリストに変換している
  - 「＼s*」は正規表現と呼ばれる記法で、「＼s」はスペース(空白)、*は0個以上並んだ文字列を表している

変換前

今回の静岡観光の目的は？（複数選択可）
温泉, まち歩き
グルメ, ショッピング, 温泉
グルメ, ショッピング, まち歩き, 美術館・博物館等
温泉, ドライブ・ツーリング
グルメ, ショッピング, 名所旧跡の観光, テーマパーク

変換後

	今回の静岡観光の目的は？（複数選択可）
0	[温泉, まち歩き]
1	[グルメ, ショッピング, 温泉]
2	[グルメ, ショッピング, まち歩き, 美術館・博物館等]
3	[温泉, ドライブ・ツーリング]
4	[グルメ, ショッピング, 名所旧跡の観光, テーマパーク]

コードの解説(2)

```
df = data["今回の静岡観光の目的は?(複数選択可)"].str.split(',\s*').apply(lambda x: pd.Series(1,index=x))
```

- applyメソッドのラムダ式を用いて、データの各リストに対して項目名を要素名とし、値を1.0としたSeries型変数を生成する
- 上記Series型変数を各行とするDataFrame型変数が生成される

	今回の静岡観光の目的は？（複数選択可）
0	[温泉, まち歩き]
1	[グルメ, ショッピング, 温泉]
2	[グルメ, ショッピング, まち歩き, 美術館・博物館等]
3	[温泉, ドライブ・ツーリング]
4	[グルメ, ショッピング, 名所旧跡の観光, テーマパーク]

	0
温泉	1.0
まち歩き	1.0
グルメ	NaN
ショッピング	NaN
美術館・博物館等	NaN
ドライブ・ツーリング	NaN
名所旧跡の観光	NaN
テーマパーク	NaN
自然鑑賞	NaN
行祭事・イベント	NaN
各種体験	NaN

	温泉	まち歩き	グルメ	ショッピング	美術館・博物館等	ドライブ・ツーリング	名所旧跡の観光	テーマパーク	自然鑑賞	行祭事・イベント	各種体験
0	1.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1.0	NaN	1.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	1.0	1.0	1.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN
3	1.0	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN

コードの解説(3)

```
df = df.fillna(0).astype(int)
```

- 1.0でない列にはNaN (No Answer)が入るので、これを0に修正し整数(int)に補正する
  - fillna(x)メソッドでは、NaNを指定の値xで埋めることができる
  - astype(t)メソッドでは、要素の型tを指定できる

	温泉	まち歩き	グルメ	ショッピング	美術館・博物館等	ドライブ・ツーリング	名所旧跡の観光	テーマパーク	自然鑑賞	行祭事・イベント	各種体験
0	1.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1.0	NaN	1.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	1.0	1.0	1.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN
3	1.0	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN

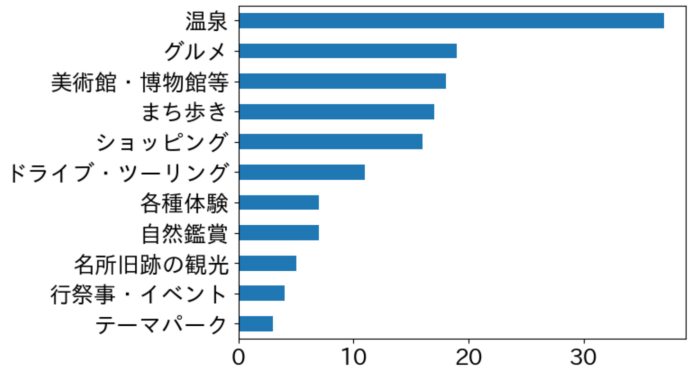
	温泉	まち歩き	グルメ	ショッピング	美術館・博物館等	ドライブ・ツーリング	名所旧跡の観光	テーマパーク	自然鑑賞	行祭事・イベント	各種体験
0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0
2	0	1	1	1	1	0	0	0	0	0	0
3	1	0	0	0	0	1	0	0	0	0	0

質的変数(複数回答)のグラフ描画

- 各列を縦方向に集計し、横棒グラフを描画しましょう。
  - 新しいコードセルを追加して、下記を記述し実行

```
ret = df.sum()
ret = ret.sort_values()
ret.plot.barh()
plt.show()
```

  - データフレームのsumメソッドで各列の合計が計算できる。
  - 集計結果をsort_valuesメソッドでソート



量的変数の単純集計

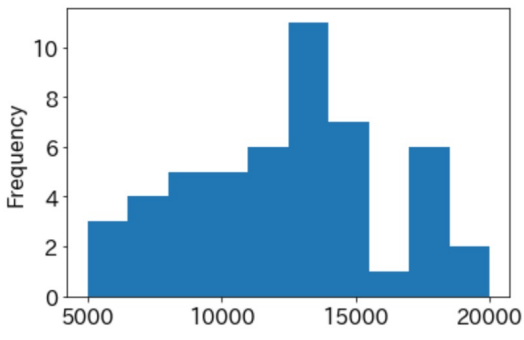
- 量的変数については、和(sum)、平均値(mean)、最小値(min)、最大値(max)などの集計が可能です。
- 基本統計量の一括計算がdescribeメソッドで可能です。
- 量的変数に対してヒストグラムの描画がplot.histメソッドで可能です。
  - 新しくコードセルを追加して、下記を記述して実行

```
#col7 = data["今回の静岡観光の１人あたりの宿泊費は？（単位は円、数字のみでお答えください。例：15000）"]
col7 = data.iloc[:,7]
print(col7.sum())
print(col7.mean())
print(col7.min())
print(col7.max())
print(col7.describe())
col7.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



(補足)クロス集計(1)

- 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。
  - 新しくコードセルを追加して、下記を記述して実行

```
col1 = data["あなたの性別は？"]
col2 = data["あなたの年代は？"]

cross = pd.crosstab(col1, col2)
display(cross)
```

集計したい2つの列を指定

  - クロス集計表(cross)

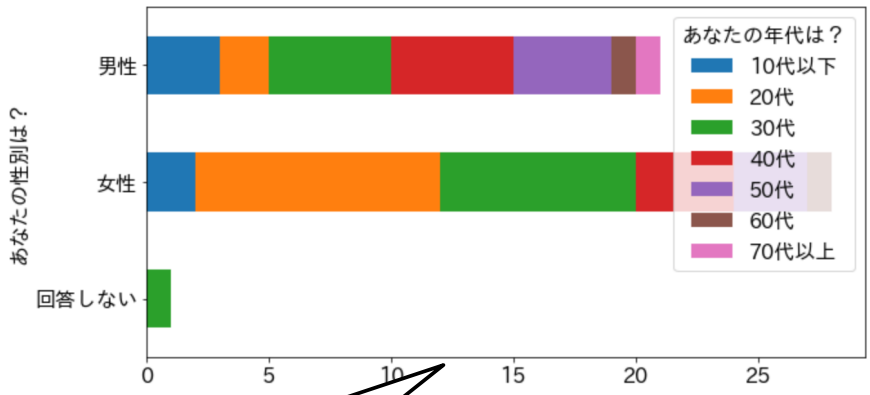
あなたの性別は? \ あなたの年代は?	10代以下	20代	30代	40代	50代	60代	70代以上
回答しない	0	0	1	0	0	0	0
女性	2	10	8	4	3	1	0
男性	3	2	5	5	4	1	1

(補足)クロス集計(2)

- 比較のため、積み上げグラフを描いてみましょう。

積み上げグラフ

```
cross.plot.barh(figsize=(10, 5), stacked=True)
plt.show()
```



凡例が被ってしまったり、全体が100%になっていないので見辛い

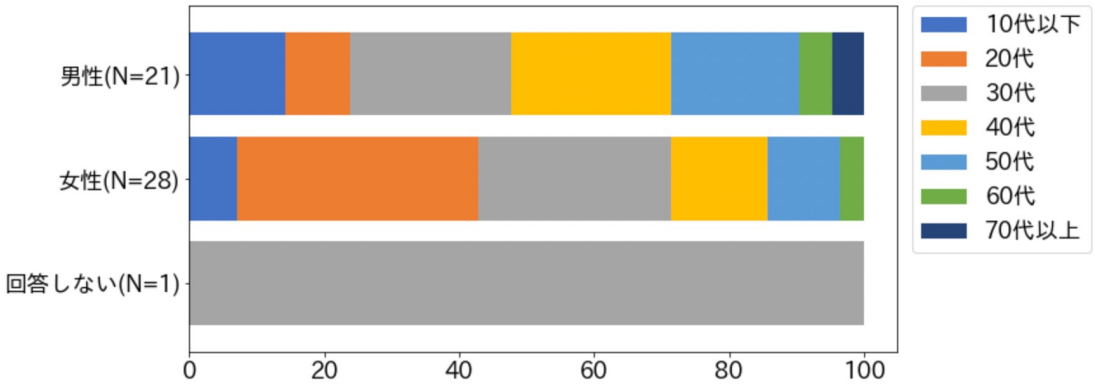
(補足)クロス集計(3)

- 積み上げグラフの場合、実数でなく割合(パーセント)に変換して描くと合計が100%で揃って見やすくなります。
  - 新しくコードセルを追加して、下記を記述して実行

```
cross = pd.crosstab(col1, col2)
#実数の表crossをパーセントの表cross2に変換(axis=1で列方向の合計で各要素を割る)
cross2 = cross.apply(lambda x: x / x.sum() * 100, axis=1)

#列方向(axis=1)にsumで合計を計算し、新しい「計」の列に記録
cross["計"] = cross.apply(lambda x: x.sum(), axis=1)
#内包表記でラベルを作る
labels = [i + "(N=" + str(j) + ")" for i, j in zip(cross.index, cross.iloc[:, -1])]
#ラベルをcross2の行名に設定
cross2.index = labels

#横棒グラフを描く
cross2.plot.barh(figsize=(10, 5), stacked=True, width=0.8, \
        color=("#4472c4", "#ed7d31", "#a5a5a5", "#ffc000", "#5b9bd5", \
        "#70ad47", "#264478", "#9e480e", "#636363", "#008000"))
#凡例の左上が、グラフの右上を(1, 1)としたとき、それよりも0.02だけ右に配置
plt.legend(bbox_to_anchor=(1.02, 1), loc="upper left", borderaxespad=0)
plt.ylabel("")
```


(補足)クロス集計(4)

- 100%積み上げグラフの例



(補足)クロス集計(5)

- 全体との比較を行いたい場合は、全体の行を追加する

```
cross = pd.crosstab(col1, col2)
#全体の行を追加
cross.loc["全体"] = cross.sum()
#実数の表crossをパーセントの表cross2に変換(axis=1で列方向の合計で各要素を割る)
cross2 = cross.apply(lambda x: x / x.sum() * 100, axis=1)
```

・・・省略・・・

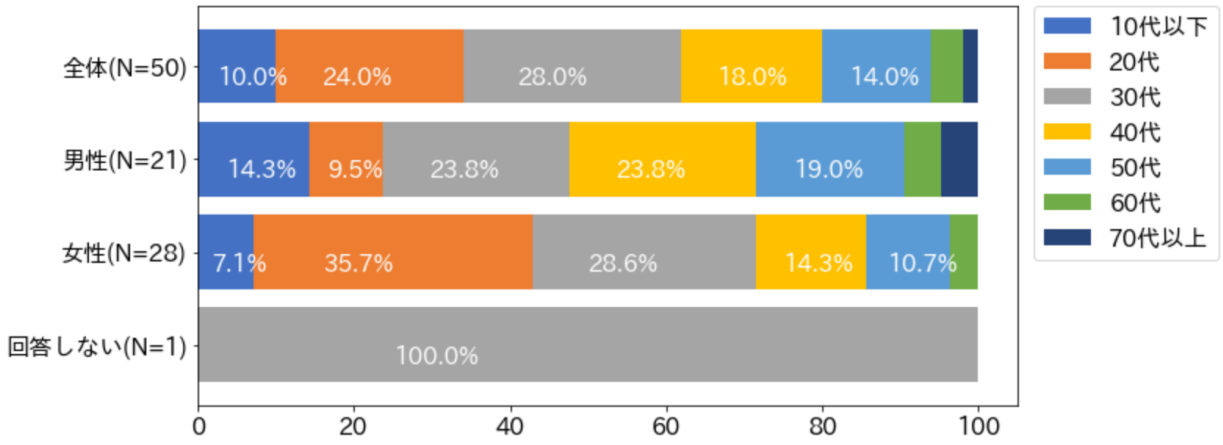
- グラフにパーセント表示を追加する

・・・省略・・・

```
plt.ylabel("")

ax = plt.gca() #グラフの現在の軸を取得
for p in ax.patches: #1つ1つのグラフの棒を取得し繰り返し
  if p.get_width() >= 5: #棒の幅が5以上ならパーセントを表示
    #annotateメソッドでパーセントを描く
    ax.annotate('{:.01f}'.format(p.get_width()) + "%", \
                (p.get_x() + p.get_width() * 0.25, \
                p.get_y() + p.get_height() * 0.35), \
                color="white")
plt.show()
```


(補足)クロス集計(6)

- 全体行とパーセント表示の例



(補足)クロス集計の関数化(1)

- クロス集計及びそのグラフ描画を関数化してみましょう。
- 関数化の例(前半)

```
def crossbar(data, c1, c2):
  col1 = data[c1]
  col2 = data[c2]
  cross = pd.crosstab(col1, col2)
  cross.loc["全体"] = cross.sum()
  cross2 = cross.apply(lambda x: x / x.sum() * 100, axis=1)
  #display(cross2)

  cross["計"] = cross.apply(lambda x: x.sum(), axis=1)
  #display(cross)
  labels = [i + "(N=" + str(j) + ")" for i, j in zip(cross.index, cross.iloc[:, -1])]
  #display(labels)
  cross2.index = labels

  cross2.plot.barh(figsize=(10, 5), stacked=True, width=0.8, \
                   color=("#4472c4", "#ed7d31", "#a5a5a5", "#ffc000", "#5b9bd5", \
                          "#70ad47", "#264478", "#9e480e", "#636363", "#008000"))
  plt.legend(bbox_to_anchor=(1.02, 1), loc="upper left", borderaxespad=0)
  plt.ylabel("")
```

次ページに続く

(補足)クロス集計の関数化(2)

- 関数化の例(後半)

```
  ax = plt.gca()
  for p in ax.patches:
    if p.get_width() >= 5:
      ax.annotate('{:.01f}'.format(p.get_width()) + "%", \
                  (p.get_x() + p.get_width() * 0.25, \
                  p.get_y() + p.get_height() * 0.35), \
                  color="white")
  plt.savefig("図1.png")
  plt.show()

crossbar(data, "あなたの性別は?", "あなたの職業は?")
crossbar(data, "あなたの性別は?", "今回の静岡観光の満足度は?")
```