第2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
- 質的変数（単数回答）集計の関数化
- mapとapply
- 質的変数（複数回答）の単純集計
- 量的変数の単純集計
- クロス集計
- （補足）リスト内包表記
- 課題
- （補足）統計的仮説検定

関数によるグラフ作成の自動化

- Pythonで集計結果をもとにグラフを描く
- 設定のためのコードが多く必要で面倒では？
- 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）

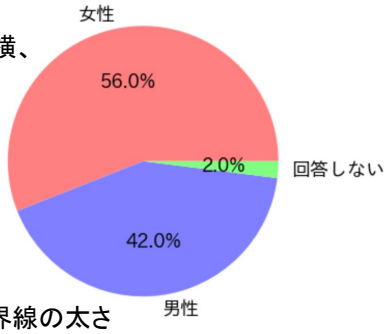
- 質的変数（単数回答）を集計し、円グラフを描く関数を定義します。
- 手順
  - 新しいコードセルを追加します。
  - 下記のpie関数を定義します。
  - 定義いたpie関数を呼び出してみます。

```
#pie関数（引数：データフレーム、列名）
def pie(data, column):
  plt.rcParams["font.size"] = 18 #フォントサイズの設定

  ret = data[column].value_counts() #集計
  #円グラフを描く
  ret.plot.pie(autopct="%.1f%%", figsize=(6, 6), \
    wedgeprops={"linewidth": 0, "edgecolor": "white"}, \
    label = "", \
    colors=("#ff8080", "#8080ff", "#80ff80"))
  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?")
```

グラフサイズ（横、縦のインチ）

境界線の太さと色

円グラフの色分け（#RRGGBB形式）



質的変数(単数回答)集計の関数化(2)

- 集計表も出力できるように関数を修正します。
  - 追記部分は赤枠

```
#pie関数(引数:データフレーム、列名)
def pie(data, column):
  plt.rcParams["font.size"] = 18 #フォントサイズの設定

  ret = data[column].value_counts() #集計
  #円グラフを描く
  ret.plot.pie(autopct="%.1f%%", figsize=(6, 6), \
    wedgeprops={"linewidth": 0, "edgecolor": "white"}, \
    label = "", \
    colors=("#ff8080", "#8080ff", "#80ff80"))
  plt.show() #グラフ描画の確定

  kensu = data[column].count() #回答件数
  ret.loc["総計"] = kensu          #総計業を追加
  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[column].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?")
```


mapとapply

- データフレームの各列(各行)のすべての要素に処理を適用したい場合にmap、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

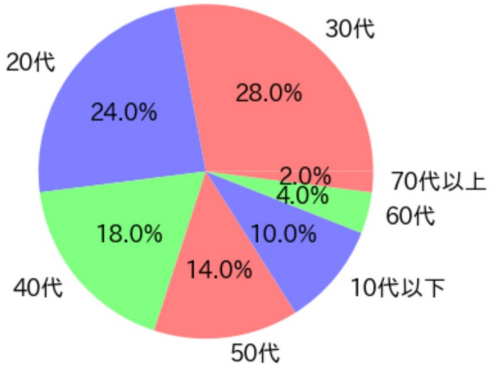
```
#各要素に関数を適用(map)
def test(x):
  return x + "です"
ret = data["あなたの性別は?"].map(test)
display(ret)

#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。

質的変数(単数回答)集計の関数化(3)

- pie関数の呼び出し方
  - pie(データフレーム, 列名)
- 例:
  - pie(data, “あたなの性別は？”)
  - pie(data, “あなたの年代は？”)

- 一見便利だが？
  - 色分けが少ない
  - 年代の場合、件数の多い順にソートされていて見づらい
  - 項目名でソートできないか？



	あなたの年代は?	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%

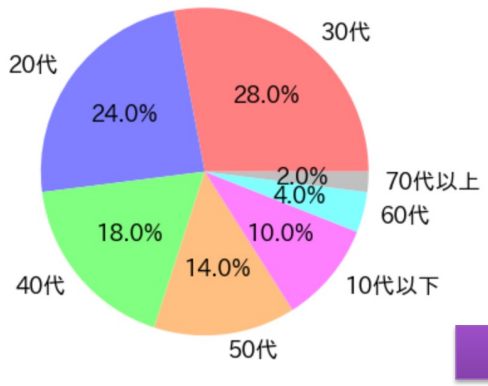
質的変数(単数回答)集計の関数化(4)

- 色分けを増やし、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(引数:データフレーム、列名)
def pie(data, column, sort_index=False):
  plt.rcParams["font.size"] = 18 #フォントサイズの設定

  ret = data[column].value_counts() #集計
  #項目名でソートするか否か
  if sort_index == True:
    ret = ret.sort_index()
  #円グラフを描く
  ret.plot.pie(autopct="%.1f%%", figsize=(6, 6), \
    wedgeprops={"linewidth": 0, "edgecolor": "white"}, \
    label = "", \
    colors=("#ff8080", "#8080ff", "#80ff80", "#ffc080", "#ff80ff", "#80ffff", \
            "#c0c0c0","#d0c060", "#c080ff", "#80ffc0"))
  plt.show() #グラフ描画の確定
```

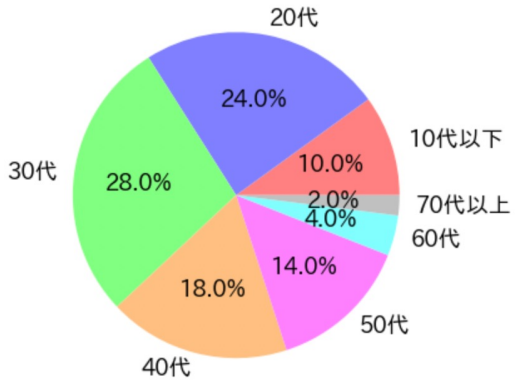
質的変数(単数回答)集計の関数化(5)

```
pie(data, "あなたの年代は?")
```



	あなたの年代は?	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%

```
pie(data, "あなたの年代は?", sort_index=True)
```



	あなたの年代は?	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%

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辛いので、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。

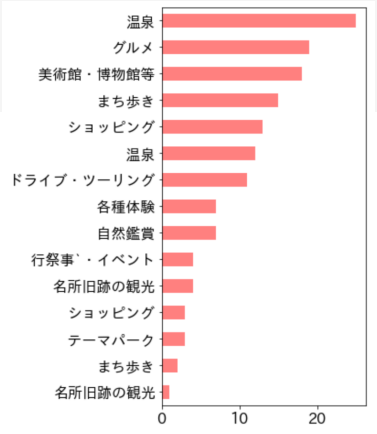
```
df = data["今回の静岡観光の目的は?(複数選択可)"].str.split(',\s*').apply(lambda x: pd.Series(1,index=x))
df = df.fillna(0).astype(int)
display(df)
```

- 解説
  - 複数回答の列を上の行から1つずつ取り出して、strアクセサによって文字列メソッドを適用する
    - strアクセサを使うと、データの各要素に対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
  - applyメソッドのラムダ式で、データの各要素に対してカンマで区切った項目名を列ラベルとし、値を1としたSeries型変数を生成する
    - 結果として、上記Series型変数を各行とするDataFrame型変数が生成される
  - 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

質的変数(複数回答)の単純集計(3)

- 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() #各列の合計
ret = ret.sort_values(0) #0列でソート
ret.plot.barh(figsize=(6, 8), color=("#ff8080")) #横棒グラフ
plt.show()

#集計表
kensu = data["今回の静岡観光の目的は？（複数選択可）"].count()
ret = pd.DataFrame(ret)
ret["割合"] = ret[0].apply(lambda x: '{:.01f}'.format(x / kensu * 100) + "%")
ret = ret.sort_values(0, ascending=False)
display(ret)
print("N =", kensu)
```



- 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。
- 集計結果の0列目に合計の件数が入っているので、sort_valuesでソート

量的変数の単純集計

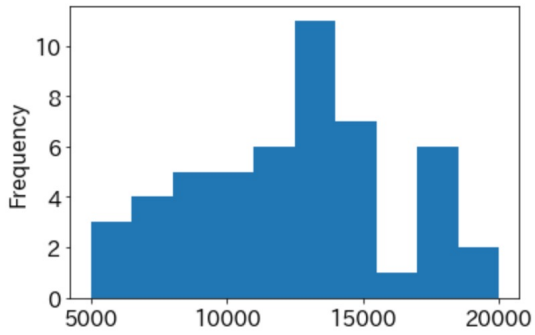
- 量的変数については、和（sum）、平均値（mean）、最小値（min）、最大値（max）などの集計が可能です。
- 基本統計量の一括計算がdescribeメソッドで可能です。
- 量的変数に対してヒストグラムの描画がplot.histメソッドで可能です。
  - 新しくコードセルを追加して、下記を記述して実行

```
#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



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）

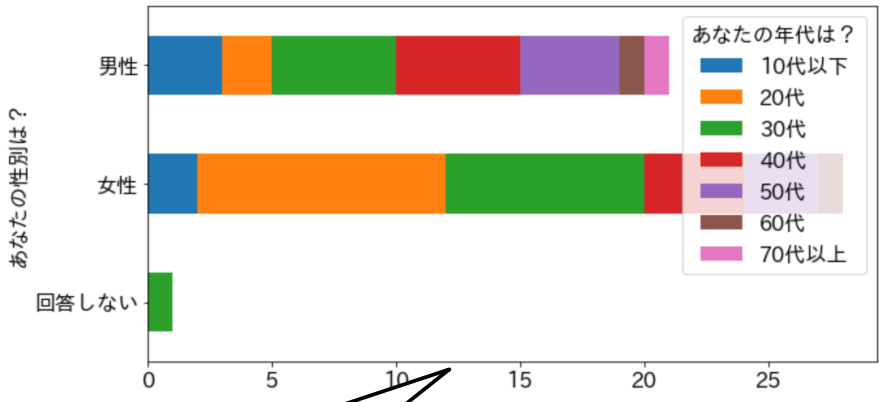
あなたの年代は? / あなたの性別は?	10代以下	20代	30代	40代	50代	60代	70代以上
回答しない	0	0	1	0	0	0	0
女性	2	10	8	4	3	1	0
男性	3	2	5	5	4	1	1

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

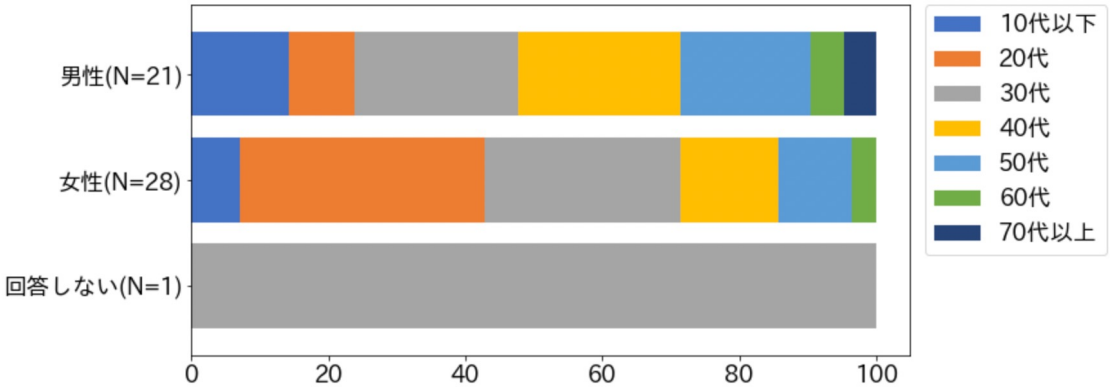
クロス集計(3)

- 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("")
```

クロス集計(4)

- 100%積み上げグラフの例



(補足)リスト内包表記(1)

- リスト内包表記は、あるシーケンスをもとにして、新しいリストを返す式として記述します。
  - 構文

リスト変数 = [一時変数による式 for 一時変数 in シーケンス]

  - 例:

```
a = [2, 4, 6, 8, 10]
b = []
for i in a:
    b.append(i*i)
print(b)
```

for文の場合

=

```
a = [2, 4, 6, 8, 10]
b = [i * i for i in a]
print(b)
```

内包表記の場合

リストa内の要素をiに取り出してきて、iを基にリストbを作る

(補足)リスト内包表記(2)

- 複数リストの同じ順番の要素から新たなリストを作る
  - zip関数を使うことで、複数のリストから同じ順番に要素を取り出して、新たなリストを作成することができます。
  - 構文

リスト変数 = [一時変数による式 for 一時変数i, 一時変数j in zip(シーケンスA, シーケンスB)]

  - 例:3名の生徒の英語と数学の合計点を求める。

```
eigo = [80, 35, 65]
sugaku = [64, 76, 35]
goukei = [i + j for i, j in zip(eigo, sugaku)]
print(goukei)
```

```
[144, 111, 100]
```


(補足)クロス集計

- 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・・・

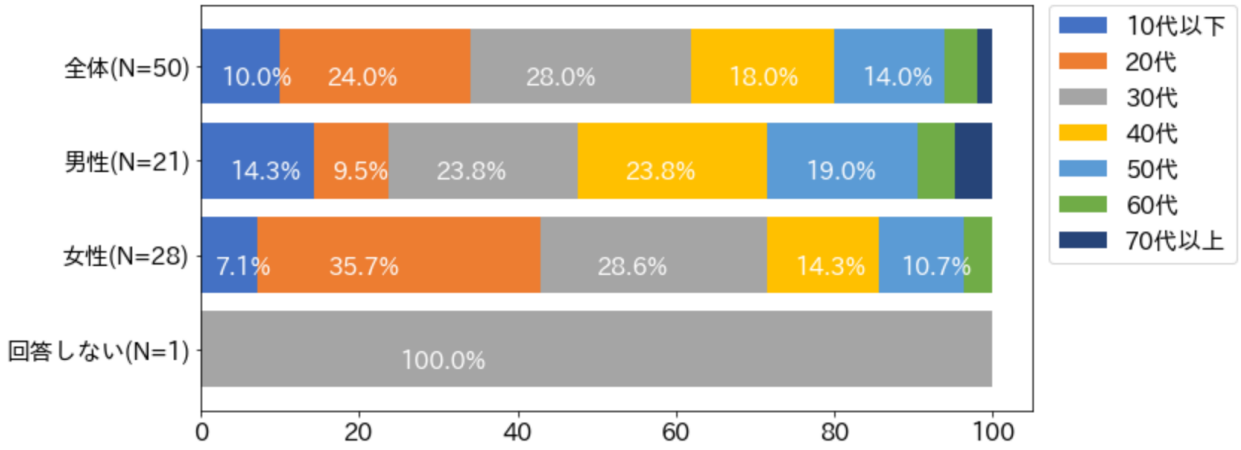
- グラフにパーセント表示を追加する

・・・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

- 全体行とパーセント表示の例



課題

- クロス集計及びそのグラフ描画を関数化してください。
- 解答例(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

課題

- 解答例（後半）

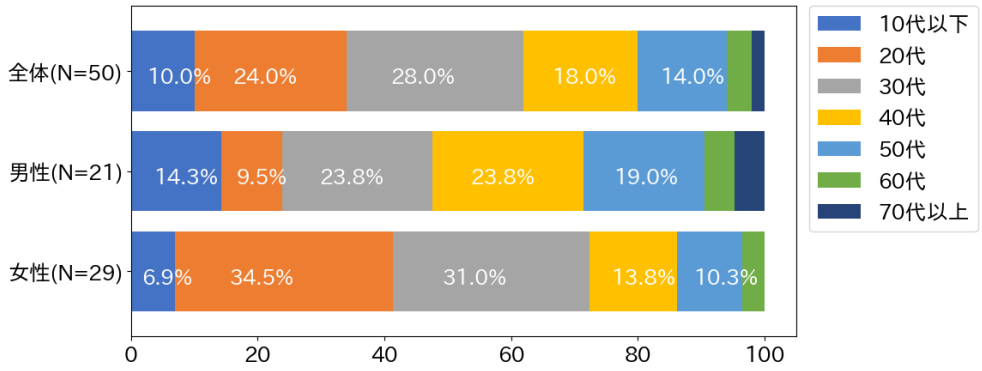
```
  ax = plt.gca()
  for p in ax.patches:
    if p.get_width() >= 5:
      ax.annotate('{:.01f}'.format(p.get_width()) + "%", \
                  (p.get_x() + p.get_width() * 0.25, \
                  p.get_y() + p.get_height() * 0.35), \
                  color="white")
  plt.savefig("図1.png")
  plt.show()

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?")
```


（補足）統計的仮説検定

- 母集団に関する帰無仮説と対立仮説を設定する
  - 帰無仮説・・・母集団に差がない、効果がないなどという仮説で、本来主張したいこととは逆の仮説H0
  - 対立仮説・・・母集団に差がある、効果があるなどという仮説H1
- 検定統計量
  - 実際のデータから計算される検定を行うための統計量
  - 例：
    - t値：量的変数の母分散が未知の分布における標準正規分布の近似値
    - χ2（カイ2乗）値：質的変数の観測度数と期待度数の差の2乗を期待度数で割って足し合わせた値
- 有意水準と棄却域
  - 有意水準・・・どの程度低い確率の結果が示されたら帰無仮説を棄却するかという基準α。一般に5%や1%が用いられる。
  - 棄却域・・・帰無仮説のもとで、非常に生じにくい検定統計量の値の範囲

(補足)カイ二乗(χ^2)検定(独立性の検定)

- 質的変数の有意差の検定
  - 例題:性別によって年代の分布に差があるか？
    - 男性の方が中高年が多い
    - 女性の方が20代、30代が多い
    - → 統計的に差があると言ってよいのか？



(補足)性別によって統計的に差があると言えるか？

- 男性と女性で、年代の傾向に優位な差があるか？ないか？
  - 年代の傾向は性別に依存している？独立している？
- 帰無仮説:男性と女性で、年代の傾向には差がない。
  - つまり、性別と年代の傾向は独立している。
- 対立仮説:男性と女性で、年代の傾向には差がある。
  - つまり、性別に年代の傾向は依存している。

クロス集計表

あなたの年代は? / あなたの性別は?	10代以下	20代	30代	40代	50代	60代	70代以上	合計
女性	2	10	9	4	3	1	0	21
男性	3	2	5	5	4	1	1	29
合計	5	12	14	9	7	2	1	50

(補足)カイ二乗値とp値を求める①

- カイ二乗値
  - クロス集計表の全ての項目における下記の和

$$\frac{(\text{観測度数} - \text{期待度数})^2}{\text{期待度数}}$$

クロス集計表

あなたの年代は? / あなたの性別は?	10代以下	20代	30代	40代	50代	60代	70代以上	合計
女性	2	10	9	4	3	1	0	21
男性	3	2	5	5	4	1	1	29
合計	5	12	14	9	7	2	1	50

  - 期待度数
    - 男女で年代に差がない(独立している)と仮定した場合の値
    - 例えば、全体では女性21人、男性29人の計50人なので、10代以下は合計5人だから、女性は2.1人、男性は2.9人が期待度数となる
  - 自由度
    - 縦横の合計が与えられた際に、自由に埋められる項目の数 → 今回は自由度6
  - p値
    - たまたま(本当の母集団の傾向と違うサンプルの偏りなどで)このような結果が得られた確率

(補足)カイ二乗値とp値を求める②

- 課題のcrossbar関数を修正する

```
from scipy import stats
def crossbar(data, c1, c2):
    col1 = data[c1]
    col2 = data[c2]
    cross = pd.crosstab(col1, col2)

    #カイ二乗検定
    chi2, p, dof, exp = stats.chi2_contingency(cross, correction=False)
    print("期待度数", "\n", exp)
    print("自由度", "\n", dof)
    print("カイ二乗値", "\n", chi2)
    print("p値", "\n", p)

    cross.loc["全体"] = cross.sum()
```

カイ二乗検定を行うためのscipyモジュールのインポート

```
期待度数
 [[2.9  6.96 8.12 5.22 4.06 1.16 0.58]
 [2.1  5.04 5.88 3.78 2.94 0.84 0.42]]
自由度
 6
カイ二乗値
 6.824875544087368
p値
 0.33734681693849855
```

p値 > 0.05(5%)なので、帰無仮説は棄却されない＝差があるとは言えない