June 22, 2023

Contents

1 Setup

Importiamo le librerie qui usate

```
>>> import numpy as np
>>> import pandas as pd
>>> import pingouin as pg
>>> from scipy import stats
```

2 Medie

2.1 Test t: 1 gruppo vs valore teorico

```
>>> x = [5.5, 2.4, 6.8, 9.6, 4.2]
>>> stats.ttest_1samp(x, popmean = 4)
TtestResult(statistic=1.3973913920955365, pvalue=0.23482367964421416, df=4)
>>> pg.ttest(x, 4)
               T  dof alternative  ...   cohen-d   BF10     power
T-test  1.397391    4   two-sided  ...  0.624932  0.766  0.191796

[1 rows x 8 columns]
```

2.2 Test t: 2 gruppi indipendenti

```
>>> np.random.seed(123)
>>> trt  = np.random.normal(size=18)
>>> ctrl = np.random.normal(size=22)
>>> stats.ttest_ind(trt, ctrl, equal_var = False)
Ttest_indResult(statistic=0.62859959726889, pvalue=0.5339329415063301)
>>> pg.ttest(trt, ctrl)
             T        dof alternative  ...   cohen-d   BF10     power
T-test  0.6286  33.040969   two-sided  ...  0.203618  0.363  0.095756
```

```
[1 rows x 8 columns]
```

2.3 Anova (2+ gruppi indipendenti)

```
>>> df = pg.read_dataset('anova')
>>> df.head()
   Subject   Hair color  Pain threshold
0        1  Light Blond              62
1        2  Light Blond              60
2        3  Light Blond              71
3        4  Light Blond              55
4        5  Light Blond              48
>>> df["Hair color"].value_counts()
Hair color
Light Blond       5
Dark Blond        5
Dark Brunette     5
Light Brunette    4
Name: count, dtype: int64
>>> # oneway classica
>>> pg.anova(dv='Pain threshold', between='Hair color', data=df, detailed = True)
       Source           SS  DF          MS         F     p-unc       np2
0  Hair color  1360.726316   3  453.575439  6.791407  0.004114  0.575962
1      Within  1001.800000  15   66.786667       NaN       NaN       NaN
>>> chunks = [data["Pain threshold"].values
...           for color, data in df.groupby("Hair color")]
>>> stats.f_oneway(*chunks)
F_onewayResult(statistic=6.791407046264094, pvalue=0.00411422733307741)
>>> # non assumendo numerosità comuni e/o varianza costante
>>> pg.welch_anova(dv='Pain threshold', between='Hair color', data=df)
       Source  ddof1     ddof2         F     p-unc       np2
0  Hair color      3  8.329841  5.890115  0.018813  0.575962
```

2.4 Test t: 2 gruppi appaiati

```
>>> pre = [5.5, 2.4, np.nan, 9.6, 4.2]
>>> post = [6.4, 3.4, 6.4, 11., 4.8]
>>> stats.ttest_rel(pre, post, nan_policy="omit")
TtestResult(statistic=-5.901869285972221, pvalue=0.009712771595911211, df=3)
>>> pg.ttest(pre, post, paired=True)
               T  dof alternative  ...   cohen-d   BF10     power
T-test -5.901869    3   two-sided  ...  0.306268  7.169  0.072967

[1 rows x 8 columns]
```

2.5 Anova per misure ripetute (2+ gruppi appaiati)

```
>>> # dataset in formato long
>>> df = pg.read_dataset('rm_anova')
>>> df.head()
   Subject  Gender Region  ... DesireToKill  Disgustingness Frighteningness
0        1  Female  North  ...         10.0            High            High
1        1  Female  North  ...          9.0            High             Low
2        1  Female  North  ...          6.0             Low            High
3        1  Female  North  ...          6.0             Low             Low
4        2  Female  North  ...         10.0            High            High

[5 rows x 7 columns]
>>> pg.rm_anova(dv='DesireToKill', within='Disgustingness',
...             subject='Subject', data=df, detailed=True)
           Source          SS  DF  ...     p-unc       ng2  eps
0  Disgustingness   27.485215   1  ...  0.000793  0.025784  1.0
1           Error  209.952285  92  ...       NaN       NaN  NaN

[2 rows x 8 columns]
>>> # dataset in formato wide
>>> df = pg.read_dataset('rm_anova_wide')
>>> df.head()
   Before  1 week  2 week  3 week
0     4.3     5.3     4.8     6.3
1     3.9     2.3     5.6     4.3
2     4.5     2.6     4.1     NaN
3     5.1     4.2     6.0     6.3
4     3.8     3.6     4.8     6.8
>>> pg.rm_anova(df)
   Source  ddof1  ddof2         F     p-unc       ng2       eps
0  Within      3     24  5.200652  0.006557  0.346392  0.694329
```

3 Non parametric

3.1 Wilcoxon

```
>>> pre  = np.array([20, 22, 19, 20, 22, 18, 24, 20, 19, 24, 26, 13])
>>> post = np.array([38, 37, 33, 29, 14, 12, 20, 22, 17, 25, 26, 16])
>>> stats.wilcoxon(pre, post)
WilcoxonResult(statistic=20.5, pvalue=0.2661660677806492)
>>> pg.wilcoxon(pre, post, correction = False)
          W-val alternative     p-val       RBC      CLES
Wilcoxon   20.5   two-sided  0.266166 -0.378788  0.395833
>>> pg.wilcoxon(pre, post) # con correzione di continuità
          W-val alternative     p-val       RBC      CLES
```

```
Wilcoxon   20.5   two-sided  0.285765 -0.378788  0.395833
```

3.2 Mann Whitney

```
>>> trt = np.random.uniform(low=0, high=1, size=20)
>>> ctrl = np.random.uniform(low=0.2, high=1.2, size=20)
>>> stats.mannwhitneyu(trt, ctrl, use_continuity=True)
MannwhitneyuResult(statistic=149.0, pvalue=0.17192970543827346)
>>> pg.mwu(trt, ctrl)
     U-val alternative    p-val    RBC    CLES
MWU  149.0   two-sided  0.17193  0.255  0.3725
```

TODO: scipy.stats.brunnermunzel

3.3 Kruskal Wallis

```
>>> df = pg.read_dataset('anova')
>>> # pingouin
>>> pg.kruskal(data=df, dv='Pain threshold', between='Hair color')
             Source  ddof1         H     p-unc
Kruskal  Hair color      3  10.58863  0.014172
>>> # scipy
>>> stats.kruskal(*chunks)
KruskalResult(statistic=10.588630377524138, pvalue=0.014171563303136903)
```

3.4 Friedman test

Tipo un wilcoxon con più colonne di 2

```
>>> # dati da friedman.test in R
>>> df = pd.DataFrame(np.array([5.40, 5.50, 5.55,
...                             5.85, 5.70, 5.75,
...                             5.20, 5.60, 5.50,
...                             5.55, 5.50, 5.40,
...                             5.90, 5.85, 5.70,
...                             5.45, 5.55, 5.60,
...                             5.40, 5.40, 5.35,
...                             5.45, 5.50, 5.35,
...                             5.25, 5.15, 5.00,
...                             5.85, 5.80, 5.70,
...                             5.25, 5.20, 5.10,
...                             5.65, 5.55, 5.45,
...                             5.60, 5.35, 5.45,
...                             5.05, 5.00, 4.95,
...                             5.50, 5.50, 5.40,
...                             5.45, 5.55, 5.50,
...                             5.55, 5.55, 5.35,
...                             5.45, 5.50, 5.55,
```

```
...                              5.50, 5.45, 5.25,
...                              5.65, 5.60, 5.40,
...                              5.70, 5.65, 5.55,
...                              6.30, 6.30, 6.25]).reshape(22,3),
...                      columns = ["t0", "t1", "t2"])
>>> stats.friedmanchisquare(df.t0, df.t1, df.t2)
FriedmanchisquareResult(statistic=11.142857142857132, pvalue=0.003805040775511383)
>>> pg.friedman(df)
          Source         W  ddof1          Q     p-unc
Friedman  Within  0.253247      2  11.142857  0.003805
```

4 Proporzioni

4.1 Test binomiale e CI clopper pearson

```
>>> test = stats.binomtest(3, n=15, p=0.1) #p è la probabilità sotto h0 da rifiutare
>>> test
BinomTestResult(k=3, n=15, alternative='two-sided', statistic=0.2, pvalue=0.1840610691063910
>>> test.proportion_ci()
ConfidenceInterval(low=0.04331200510583602, high=0.48089113380685317)
```

4.2 Test di Fisher

Si ha per le tabelle 2x2

```
>>> # odds ratio (stima) calcolato è diverso da quello di R (vedi doc), p-uguale
>>> tea = np.array([[3, 1], [1, 3]])
>>> stats.fisher_exact(tea)
SignificanceResult(statistic=9.0, pvalue=0.48571428571428565)
```

TODO: stats.barnard_exact

4.3 Chisquare

Per le tabelle $n \times m$

```
>>> obs = np.array([[10, 10, 20],
...                 [20, 20, 20]])
>>> stats.chi2_contingency(obs)
Chi2ContingencyResult(statistic=2.7777777777777777, pvalue=0.24935220877729622, dof=2, expec
       [18., 18., 24.]]))
>>> data = pg.read_dataset('chi2_independence')
>>> pg.chi2_independence(data, x='sex', y='target')
(target          0           1
sex
0       43.722772   52.277228
1       94.277228  112.722772, target      0     1
sex
```

```
0        24.5  71.5
1       113.5  93.5,                   test    lambda       chi2  ...          pval    cramer
0             pearson  1.000000  22.717227  ...  1.876778e-06  0.273814  0.997494
1        cressie-read  0.666667  22.931427  ...  1.678845e-06  0.275102  0.997663
2      log-likelihood  0.000000  23.557374  ...  1.212439e-06  0.278832  0.998096
3       freeman-tukey -0.500000  24.219622  ...  8.595211e-07  0.282724  0.998469
4  mod-log-likelihood -1.000000  25.071078  ...  5.525544e-07  0.287651  0.998845
5              neyman -2.000000  27.457956  ...  1.605471e-07  0.301032  0.999481

[6 rows x 7 columns])
```

4.4 McNemar

```
>>> data = pg.read_dataset('chi2_mcnemar')
>>> pg.chi2_mcnemar(data, 'treatment_X', 'treatment_Y')
(treatment_Y   0   1
treatment_X
0             20  40
1              8  12,                chi2  dof  p-approx   p-exact
mcnemar  20.020833    1  0.000008  0.000003)
```

4.5 Q di Cochrane

Mc nemar per più tempi/trattamenti su stessi soggetti

```
>>> df = pg.read_dataset('cochran')
>>> df.head()
   Subject    Time  Energetic
0        1  Monday          1
1        2  Monday          0
2        3  Monday          0
3        4  Monday          0
4        5  Monday          1
>>> df_wide = df.pivot_table(index="Subject", columns="Time", values="Energetic")
>>> pg.cochran(df_wide)
         Source  dof         Q     p-unc
cochran  Within    2  6.705882  0.034981
```

5 Tassi

5.1 Comparazione 2 tassi

Il test di poisson di python verifica che la differenza tra tassi sia nulla (quello di R che il rapporto sia unitario)

```
>>> # poisson.test(c(11, 6+8+7), c(800, 1083+1050+878))
>>> stats.poisson_means_test(11, 800, 6+8+7, 1083+1050+878)
```

```
SignificanceResult(statistic=1.5342150126346437, pvalue=0.13862291985862774)
>>> # i risultati sono diversi ma il manuale di python dice
```

I risultati di questo test sono differenti da quelli di R ma la documentazione di python dice che ha maggior potenza del test poissoniano esatto di R.

6 Correlazione

```
>>> # generare dati
>>> mean, cov = [4, 6], [(1, .5), (.5, 1)]
>>> x, y = np.random.multivariate_normal(mean, cov, 30).T
>>> data = {"x": x, "y": y}
>>> df = pd.DataFrame(data)
```

6.1 Pearson

```
>>> stats.pearsonr(df.x, df.y)
PearsonRResult(statistic=0.42350936826041, pvalue=0.019697851908720997)
>>> pg.corr(df.x, df.y)
          n         r          CI95%     p-val   BF10     power
pearson  30  0.423509  [0.07, 0.68]  0.019698  3.041  0.663505
```

6.2 Spearman

```
>>> stats.spearmanr(df.x, df.y)
SignificanceResult(statistic=0.35973303670745277, pvalue=0.05087374850723393)
>>> pg.corr(df.x, df.y, method="spearman")
           n         r          CI95%     p-val    power
spearman  30  0.359733  [-0.0, 0.64]  0.050874  0.50986
```

6.3 Tests

```
>>> pg.rcorr
<function rcorr at 0x7f3603b989a0>
```

7 Varianze

Vediamo le funzioni per la comparazione di k varianze sotto diverse ipotesi sempre meno restrittive

7.1 Test di Bartlett

Testa parametricamente la differenza di varianze ipotizzando una distribuzione normale del carattere nella popolazione. Se a 2 gruppi è il test F.

```
>>> a = [8.88, 9.12, 9.04, 8.98, 9.00, 9.08, 9.01, 8.85, 9.06, 8.99]
>>> b = [8.88, 8.95, 9.29, 9.44, 9.15, 9.58, 8.36, 9.18, 8.67, 9.05]
>>> c = [8.95, 9.12, 8.95, 8.85, 9.03, 8.84, 9.07, 8.98, 8.86, 8.98]
>>> stats.bartlett(a, b, c)
BartlettResult(statistic=22.789434813726768, pvalue=1.1254782518834628e-05)
```

7.2 Test di Levene

Testa parametricamente la differenza di varianze non ipotizzando distribuzioni normali

```
>>> stats.levene(a, b, c)
LeveneResult(statistic=7.584952754501659, pvalue=0.002431505967249681)
```

7.3 Test di Fligner

Equivalente non parametrico

```
>>> stats.fligner(a, b, c)
FlignerResult(statistic=10.803687663522238, pvalue=0.00450826080004775)
```

8 Sopravvivenza

Utilizziamo la libreria `lifelines`

8.1 Logrank test

```
>>> T1 = [1, 4, 10, 12, 12, 3, 5.4]
>>> E1 = [1, 0, 1,  0,  1,  1, 1]
>>> T2 = [4, 5, 7, 11, 14, 20, 8, 8]
>>> E2 = [1, 1, 1, 1,  1,  1,  1, 1]

>>> from lifelines.statistics import logrank_test
>>> results = logrank_test(T1, T2, event_observed_A=E1, event_observed_B=E2)
>>> results.print_summary()
<lifelines.StatisticalResult: logrank_test>
               t_0 = -1
 null_distribution = chi squared
degrees_of_freedom = 1
         test_name = logrank_test

---
 test_statistic    p  -log2(p)
           0.09 0.77      0.38
>>> # results.p_value, results.test_statistic
```

9 Agreement

9.1 Cohen's K

9.2 Lin coefficient

10 Reliability/consistency

10.1 Cronbach α

```
>>> pg.cronbach_alpha
<function cronbach_alpha at 0x7f3603b99f80>
```

10.2 ICC

```
>>> pg.intraclass_corr
<function intraclass_corr at 0x7f3603b9a160>
```

11 Multiplicity

```
scipy.stats.tukey_hsd
pg.pairwise_tukey

statsmodels.stats.multitest.multipletests
scipy.stats.false_discovery_control

pg.multicomp
pg.pairwise_gameshowell
pg.pairwise_tukey
pg.pairwise_tests
pg.pairwise_corr
pg.ptests
```