Statistical tests

Parametric tests

Tests for means

1-sample				
Name	Sample	Hypothesis	Statistics	Notes
z-test (σ is known)	$X^n = (X_1, \dots, X_n)$, $X \sim N(\mu, \sigma^2)$	$H_0:\ \hat{\mu} = \mu_0$ $H_1:\ \hat{\mu} \neq \mu_0$.	$Z(X_n) = \frac{\overline{X}_n - \mu_0}{\sigma/\sqrt{n}} \sim N(0,1)$	• Requires data normality Code: Appendix A
t-test (σ is known)	$X^n = (X_1, \dots, X_n)$, $X \sim N(\mu, \sigma^2)$	$H_0:\ \hat{\mu} = \mu_0$ $H_1:\ \hat{\mu} \neq \mu_0$.	$T(X_n) = \frac{\overline{X}_n - \mu_0}{S/\sqrt{n}} \sim St(n-1)$	• Requires data normality • `scipy.stats.ttest_1samp`
2-samples (independent)				
z-test (σ is known)	$X_1^n = (X_{11}, \dots, X_{1n})$, $X_2^n = (X_{21}, \dots, X_{2n})$, $X_1^n \sim N(\mu_1, \sigma_1^2)$ $X_2^n \sim N(\mu_2, \sigma_2^2)$	$H_0:\ \mu_1 = \mu_2$ $H_1:\ \mu_1 <=> \mu_2$.	$Z(X_1^{n_1}, X_2^{n_2}) = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}} \sim N(0,1)$	• Requires data normality • `statsmodels.stats.weightstats.CompareMeans.ztest_ind`
t-test (σ is known)	$X_1^n = (X_{11}, \dots, X_{1n})$, $X_2^n = (X_{21}, \dots, X_{2n})$, $X_1^n \sim N(\mu_1, \sigma_1^2)$ $X_2^n \sim N(\mu_2, \sigma_2^2)$	$H_0:\ \mu_1 = \mu_2$ $H_1:\ \mu_1 <=> \mu_2$.	$T(X_1^{n_1}, X_2^{n_2}) = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}} \approx\sim St(\nu)$, $\nu = \frac{\left(\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}\right)^2}{\frac{S_1^4}{n_1^2(n_1-1)} + \frac{S_2^4}{n_2^2(n_2-1)}}$	• Requires data normality • Behrens–Fisher problem: no exact solution • Well approximated by Student dist. • `scipy.stats.ttest_ind`
2-samples (related)				
t-test	$X_1^n = (X_{11}, \dots, X_{1n})$, $X_2^n = (X_{21}, \dots, X_{2n})$, $X_1^n \sim N(\mu_1, \sigma_1^2)$ $X_2^n \sim N(\mu_2, \sigma_2^2)$	$H_0:\ \mu_1 = \mu_2$ $H_1:\ \mu_1 \neq \mu_2$	$T(X_n) = \frac{\overline{X}_1 - \overline{X}_2}{S/\sqrt{n}} \sim St(n-1)$, $S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (D_i - \overline{D})^2$, $D_i = (x_{1i} - x_{2i})$	• Requires data normality • `scipy.stats.ttest_rel`

Tests for proportions

1-sample				
Name	Sample	Hypothesis	Statistics	Notes
Binomial test	$X^n = (X_1, \dots, X_n)$, $X \sim Ber(p)$	$H_0:\ p = p_0$ $H_1:\ p <=> p_0$.	$Z(X_n) = \frac{\hat{p}-p_0}{\sqrt{p_0(1-p_0)/n}} \sim N(0,1)$ $\hat{p} = \overline{X}_n$	• `scipy.stats.binom_test`
2-samples (independent)				
z-test	$X_1^n = (X_{11}, \dots, X_{1n})$, $X_2^n = (X_{21}, \dots, X_{2n})$, $X_1^n \sim Ber(p_1)$ $X_2^n \sim Ber(p_2)$	$H_0:\ p_1 = p_2$ $H_1:\ p_1 <=> p_2$.	$Z(X_1, X_2) = \frac{\hat{p}_1-\hat{p}_2}{\sqrt{P(1-P)(\frac{1}{n_1}+\frac{1}{n_2})}} \sim N(0,1)$ $P = \frac{\hat{p}_1 n_1 + \hat{p}_2 n_2}{n_1+n_2}$	• Code: Appendix B
2-samples (related)				
z-test	$X_1^n = (X_{11}, \dots, X_{1n})$, $X_2^n = (X_{21}, \dots, X_{2n})$, $X_1^n \sim Ber(p_1)$ $X_2^n \sim Ber(p_2)$	$H_0:\ p_1 = p_2$ $H_1:\ p_1 <\neq> p_2$	$\begin{array}{cccc} X_1 \setminus X_2 & 1 & 0 & \Sigma \\ \hline 1 & e & f & e+f \\ 0 & g & h & g+h \\ \Sigma & e+g & f+h & n \end{array}$ $\hat{p}_1 = \frac{e+f}{n}$ $\hat{p}_2 = \frac{e+g}{n}$ $Z(X_1, X_2) = \frac{f-g}{\sqrt{f+g-\frac{(f-g)^2}{n}}} \sim St(n-1)$	• Code: Appendix C

Tests for variance

2-samples (independent)				
F-test	$X_1^n = (X_{11}, \dots, X_{1n}),$ $X_2^n = (X_{21}, \dots, X_{2n}),$ $X_1^n \sim N(\mu_1, \sigma_1)$ $X_2^n \sim N(\mu_2, \sigma_2)$	$H_0: \ \sigma_1 = \sigma_2$ $H_1: \ \sigma_1 <=> \sigma_2.$	$F(X_1, X_2) = \frac{S_1^2}{S_2^2} \sim F(n_1 - 1, n_2 - 2)$	• Requires data normality • Extremely sensitive to not normal data • Code: Appendix D
k-samples (independent)				
Bartlett's test	$X_1^n = (X_{11}, \dots, X_{1n}),$ $\dots$ $X_2^n = (X_{21}, \dots, X_{2n}),$ $X_1^n \sim N(\mu_1, \sigma_1)$ $\dots$ $X_2^n \sim N(\mu_2, \sigma_2)$	$H_0: \ \sigma_1 = \cdots = \sigma_k$ $H_1: \ \sigma_i \neq \sigma_j$	$T = \frac{(N-k)ln(s_p^2) - \sum_{i=1}^{k}(N_i - 1)ln(s_i^2)}{1 + (1/(3(k-1)))((\sum\limits_{i=1}^{k} 1/(N_i - 1)) - 1/(N-k))}$ $T \sim \chi^2(k-1)$ $s_p^2 = \sum\limits_{i}^{k}(N_i - 1)s_i^2/(N-k)$	• Require data normality • Less sensitive than the Fisher test to not normal data • `scipy.stats.bartlett`
Levene's test	$X_1^n = (X_{11}, \dots, X_{1n}),$ $\dots$ $X_2^n = (X_{21}, \dots, X_{2n}),$ $X_1^n \sim N(\mu_1, \sigma_1)$ $\dots$ $X_2^n \sim N(\mu_2, \sigma_2)$	$H_0: \ \sigma_1 = \cdots = \sigma_k$ $H_1: \ \sigma_i \neq \sigma_j$	$W = \frac{N-k}{k-1} \frac{\sum\limits_{i=1}^{k} N_i (Z_{i.} - Z_{..})^2}{\sum\limits_{i=1}^{k} \sum\limits_{j=1}^{N-i} (Z_{ij} - Z_{i.})^2}$ $W \sim F(k-1, N-k)$ Given a variable Y with sample of size N divided into k subgroups, where N_i is the sample size of the i-th subgroup $Z_{ij} = \lvert Y_i - \overline{Y}_{ij} \rvert$	• Require data normality • Less sensitive than the Bartlett test to not normal data • `scipy.stats.levene`

Tests normality

2-samples (independent)				
Shapiro-Wilk test	$X_1^n = (X_{11}, \ldots, X_{1n})$	$H_0: \ X \sim N(\mu, \sigma^2)$ $H_1:$ not true	$W(X^n) = \frac{(\sum\limits_{i=1}^{n} a_i X_i)^2}{\sum\limits_{i=1}^{n} (X_i - \overline{X})^2}$ a_i- ordinal statistics of normal distribution	• Very sensitive • `scipy.stats.shapiro`
D'Agostino's K-squared test	$X_1^n = (X_{11}, \ldots, X_{1n})$	$H_0: \ X \sim N(\mu, \sigma^2)$ $H_1:$ not true	A lot of formulas here	• `scipy.stats.normaltest`
Anderson-Darling test	$X_1^n = (X_{11}, \ldots, X_{1n})$	$H_0: \ X \sim N(\mu, \sigma^2)$ $H_1:$ not true	A lot of formulas here	• Worse than Shapiro-Wilk test • `scipy.stats.anderson`
Pearson's χ^2 test	$X^n = (X_1, \ldots, X_n)$	$H_0: \ X \sim N(\mu, \sigma^2)$ $H_1:$ not true	$\chi^2(X^n) = \sum\limits_{i=1}^{k} \frac{(n_i - np_i)^2}{np_i}$ μ, σ known $\sim \chi^2(k-1)$ μ, σ unknown $\sim \chi^2(k-3)$	• s`cipy.stats.chisquare`

Non-parametric tests

Name	Sample	Hypothesis	Statistics	Notes
1-sample				
Sign-test	$X^n = (X_1, \dots, X_n)$ $X_i \neq m_0$	$H_0:\ med(X) = m_0$ $H_1:\ med(X) <\neq> m_0$	$T(X_n) = \sum_{i=1}^{n} [x_i > m_0]$ $T(X_n) \sim Bin(n, 0.5)$	• Works for censored samples • `scipy.stats.sign_test`
Wilcoxon test	$X^n = (X_1, \dots, X_n)$ $X_i \neq m_0$	$H_0:\ med(X) = m_0$ $H_1:\ med(X) <\neq> m_0$	$W(X^n) = \sum_{i}^{n} rank(\lvert x_i - m_0 \rvert) \cdot sign(X_i - m_0)$ $W(X^n) \sim N(0, \frac{n(n+1)(2n+1)}{6}), (n \geq 20)$	• `scipy.stats.wilcoxon`
Permutation test	$X^n = (X_1, \dots, X_n)$ $X_i \neq m_0$	$H_0:\ E(X) = m_0$ $H_1:\ E(X) <\neq> m_0$	$T(X^n) = \sum_{i}^{n} (X_i - m_0)$	• use more information than previous Code: Appendix E
2-sample (independent)				
Mann-Whitney test	$X_1^n = (X_{11}, \dots, X_{1n}),$ $X_2^n = (X_{21}, \dots, X_{2n})$ $X_{1i} \neq X_{2i}$	$H_0:\ F_{X_1}(x) = F_{X_2}(x)$ $H_1:\ F_{X_1}(x) = F_{X_2}(x + \Delta)$	$X_{(1)} < \dots < X_{(n_1+n_2)}$ $R = \sum_{i=1}^{n_1} \text{rank}(X_{1i})$	• `scipy.stats.mannwhitneyu`
Permutation test	$X_1^n = (X_{11}, \dots, X_{1n}),$ $X_2^n = (X_{21}, \dots, X_{2n})$ $X_{1i} \neq X_{2i}$	$H_0:\ E(X_1 - X_2) = 0$ $H_1:\ E(X_1 - X_2) <\neq> 0$	$T(X^n) = \sum_{i}^{n} (X_{1i} - X_{2i})$	• use more information than previous Code: Appendix F
2-sample (related)				
Sign test	$X_1^n = (X_{11}, \dots, X_{1n}),$ $X_2^n = (X_{21}, \dots, X_{2n})$ $X_{1i} \neq X_{2i}$	$H_0:\ p(X_1 > X_2) = 1/2$ $H_1:\ p(X_1 > X_2) <\neq> 1/2$	$T(X_1^n, X_2^n) = \sum_{i}^{n} (X_{1i} > X_{2i})$ $T(X_1^n, X_2^n) \sim Bin(n, 1/2)$	• `scipy.stats.wilcoxon`
Wilcoxon test	$X_1^n = (X_{11}, \dots, X_{1n}),$ $X_2^n = (X_{21}, \dots, X_{2n})$ $X_{1i} \neq X_{2i}$	$H_0:\ med(X_1 - X_2) = 0$ $H_1:\ med(X_1 - X_2) <\neq> 0$	$W(X^n) = \sum_{i}^{n} rank(\lvert X_{1i} - X_{2i} \rvert) \cdot sign(X_{1i} - X_{2i})$	• `scipy.stats.wilcoxon`
Permutation test	$X_1^n = (X_{11}, \dots, X_{1n}),$ $X_2^n = (X_{21}, \dots, X_{2n})$ $X_{1i} \neq X_{2i}$	$H_0:\ F_{X_1}(x) = F_{X_2}(x)$ $H_1:\ F_{X_1}(x) = F_{X_2}(x + \Delta)$	$T(X_1^n, X_2^n) = \frac{1}{n_1} \sum_{i=1}^{n_1} X_{1i} - \frac{1}{n_2} \sum_{i=1}^{n_2} X_{2i}$	• Code: Appendix G

Independence tests

- `scipy.stats.chi2_contingency`
- `scipy.stats.fisher_exact`

Appendix A

1-sample z-test for means

```
import numpy as np

def z_stat(sample1, sample2, std):

    mu_0 = np.mean(sample1)
    mu_exp = np.mean(sample2)
    N = len(mu_0)

    return (mu_exp - mu_0)/(std / np.sqrt(N))

def proportions_diff_z_test(z_stat, alternative = 'two-sided'):

    if alternative not in ('two-sided', 'less', 'greater'):
        raise ValueError("alternative not recognized\n"
                         "should be 'two-sided', 'less' or 'greater'")

    if alternative == 'two-sided':
        return 2 * (1 - stats.norm.cdf(np.abs(z_stat)))

    if alternative == 'less':
        return stats.norm.cdf(z_stat)

    if alternative == 'greater':
        return 1 - stats.norm.cdf(z_stat)
```

Appendix B

2-sample z-test for proportions (independent)

```
def proportions_diff_z_stat_ind(sample1, sample2):
    n1 = len(sample1)
    n2 = len(sample2)

    p1 = float(sum(sample1)) / n1
    p2 = float(sum(sample2)) / n2
    P = float(p1*n1 + p2*n2) / (n1 + n2)

    return (p1 - p2) / np.sqrt(P * (1 - P) * (1. / n1 + 1. / n2))

def proportions_diff_z_test(z_stat, alternative = 'two-sided'):
    if alternative not in ('two-sided', 'less', 'greater'):
        raise ValueError("alternative not recognized\n"
                         "should be 'two-sided', 'less' or 'greater'")

    if alternative == 'two-sided':
        return 2 * (1 - stats.norm.cdf(np.abs(z_stat)))

    if alternative == 'less':
        return stats.norm.cdf(z_stat)

    if alternative == 'greater':
        return 1 - stats.norm.cdf(z_stat)
```

Appendix C

2-sample z-test for proportions (related)

```python
def proportions_diff_z_stat_rel(sample1, sample2):
    sample = list(zip(sample1, sample2))
    n = len(sample)

    f = sum([1 if (x[0] == 1 and x[1] == 0) else 0 for x in sample])
    g = sum([1 if (x[0] == 0 and x[1] == 1) else 0 for x in sample])

    return float(f - g) / np.sqrt(f + g - float((f - g)**2) / n )
```

Appendix D

F-test for variances

```
def f_test(sample1, sample2):
    F = np.std(sample1) / np.std(sample2)

    df1 = len(sample1) - 1
    df2 = len(sample2) - 1

    return scipy.stats.f.cdf(F, df1, df2)
```

Appendix E

1-sample Permutation test

```
def permutation_t_stat_1sample(sample, mean):
    t_stat = sum(sample - mean)
    return t_stat

def permutation_zero_distr_1sample(sample, mean, max_permutations = None):
    centered_sample = sample - mean

    if max_permutations:
        signs_array = set([tuple(x) for x in 2 * np.random.randint(2, size = (max_permutations,
                                                                              len(sample))) - 1 ])
    else:
        signs_array =  itertools.product([-1, 1], repeat = len(sample))
    distr = [sum(centered_sample * np.array(signs)) for signs in signs_array]

    return distr

def permutation_test(sample, mean, max_permutations = None, alternative = 'two-sided'):
    if alternative not in ('two-sided', 'less', 'greater'):
        raise ValueError("alternative not recognized\n"
                         "should be 'two-sided', 'less' or 'greater'")

    t_stat = permutation_t_stat_1sample(sample, mean)

    zero_distr = permutation_zero_distr_1sample(sample, mean, max_permutations)

    if alternative == 'two-sided':
        return sum([1. if abs(x) >= abs(t_stat) else 0. for x in zero_distr]) / len(zero_distr)

    if alternative == 'less':
        return sum([1. if x <= t_stat else 0. for x in zero_distr]) / len(zero_distr)

    if alternative == 'greater':
        return sum([1. if x >= t_stat else 0. for x in zero_distr]) / len(zero_distr)
```

Appendix F

2-sample Permutation test (independent)

```
def permutation_t_stat_ind(sample1, sample2):
    return np.mean(sample1) - np.mean(sample2)

def get_random_combinations(n1, n2, max_combinations):
    index = list(range(n1 + n2))
    indices = set([tuple(index)])
    for i in range(max_combinations - 1):
        np.random.shuffle(index)
        indices.add(tuple(index))
    return [(index[:n1], index[n1:]) for index in indices]

def permutation_zero_dist_ind(sample1, sample2, max_combinations = None):
    joined_sample = np.hstack((sample1, sample2))
    n1 = len(sample1)
    n = len(joined_sample)

    if max_combinations:
        indices = get_random_combinations(n1, len(sample2), max_combinations)
    else:
        indices = [(list(index), filter(lambda i: i not in index, range(n))) \
                    for index in itertools.combinations(range(n), n1)]

    distr = [joined_sample[list(i[0])].mean() - joined_sample[list(i[1])].mean() \
             for i in indices]
    return distr

def permutation_test(sample, mean, max_permutations = None, alternative = 'two-sided'):
    if alternative not in ('two-sided', 'less', 'greater'):
        raise ValueError("alternative not recognized\n"
                         "should be 'two-sided', 'less' or 'greater'")

    t_stat = permutation_t_stat_ind(sample, mean)

    zero_distr = permutation_zero_dist_ind(sample, mean, max_permutations)

    if alternative == 'two-sided':
        return sum([1. if abs(x) >= abs(t_stat) else 0. for x in zero_distr]) / len(zero_distr)

    if alternative == 'less':
        return sum([1. if x <= t_stat else 0. for x in zero_distr]) / len(zero_distr)

    if alternative == 'greater':
        return sum([1. if x >= t_stat else 0. for x in zero_distr]) / len(zero_distr)
```

Appendix G

2-sample Permutation test (related)

```
def permutation_t_stat_1sample(sample, mean):
    t_stat = sum(sample - mean)
    return t_stat

def permutation_zero_distr_1sample(sample, mean, max_permutations = None):
    centered_sample = sample - mean

    if max_permutations:
        signs_array = set([tuple(x) for x in 2 * np.random.randint(2, size = (max_permutations,
                                                                                  len(sample))) - 1 ])
    else:
        signs_array =  itertools.product([-1, 1], repeat = len(sample))
    distr = [sum(centered_sample * np.array(signs)) for signs in signs_array]

    return distr

def permutation_test(sample, mean, max_permutations = None, alternative = 'two-sided'):
    if alternative not in ('two-sided', 'less', 'greater'):
        raise ValueError("alternative not recognized\n"
                         "should be 'two-sided', 'less' or 'greater'")

    t_stat = permutation_t_stat_1sample(sample, mean)

    zero_distr = permutation_zero_distr_1sample(sample, mean, max_permutations)

    if alternative == 'two-sided':
        return sum([1. if abs(x) >= abs(t_stat) else 0. for x in zero_distr]) / len(zero_distr)

    if alternative == 'less':
        return sum([1. if x <= t_stat else 0. for x in zero_distr]) / len(zero_distr)

    if alternative == 'greater':
        return sum([1. if x >= t_stat else 0. for x in zero_distr]) / len(zero_distr)
```