[Statistical Analysis with SciPy]

Importing and Setup

- Import SciPy stats module: `from scipy import stats`
- Import NumPy for array operations: `import numpy as np`
- Set random seed for reproducibility: `np.random.seed(42)`

Descriptive Statistics

- Mean: `np.mean(data)`
- Median: `np.median(data)`
- Mode: `stats.mode(data)`
- Variance: `np.var(data)`
- Standard deviation: `np.std(data)`
- Range: `np.ptp(data)`
- Interquartile range: `stats.iqr(data)`
- Skewness: `stats.skew(data)`
- Kurtosis: `stats.kurtosis(data)`
- Coefficient of variation: `stats.variation(data)`
- Geometric mean: `stats.gmean(data)`
- Harmonic mean: `stats.hmean(data)`
- Trimmed mean: `stats.trim_mean(data, 0.1)`
- Percentile: `np.percentile(data, 75)`
- Quantile: `np.quantile(data, [0.25, 0.5, 0.75])`

Probability Distributions

- Normal distribution PDF: `stats.norm.pdf(x, loc=0, scale=1)`
- Normal distribution CDF: `stats.norm.cdf(x, loc=0, scale=1)`
- Normal distribution inverse CDF: `stats.norm.ppf(q, loc=0, scale=1)`
- Generate normal random numbers: `stats.norm.rvs(loc=0, scale=1, size=1000)`
- Uniform distribution PDF: `stats.uniform.pdf(x, loc=0, scale=1)`
- Uniform distribution CDF: `stats.uniform.cdf(x, loc=0, scale=1)`
- Generate uniform random numbers: `stats.uniform.rvs(loc=0, scale=1, size=1000)`
- Exponential distribution PDF: `stats.expon.pdf(x, scale=1)`
- Exponential distribution CDF: `stats.expon.cdf(x, scale=1)`

By: Waleed Mousa

- Generate exponential random numbers: `stats.expon.rvs(scale=1, size=1000)`
- Poisson distribution PMF: `stats.poisson.pmf(k, mu=1)`
- Poisson distribution CDF: `stats.poisson.cdf(k, mu=1)`
- Generate Poisson random numbers: `stats.poisson.rvs(mu=1, size=1000)`
- Binomial distribution PMF: `stats.binom.pmf(k, n, p)`
- Binomial distribution CDF: `stats.binom.cdf(k, n, p)`
- Generate binomial random numbers: `stats.binom.rvs(n, p, size=1000)`
- Chi-square distribution PDF: `stats.chi2.pdf(x, df)`
- Chi-square distribution CDF: `stats.chi2.cdf(x, df)`
- Generate chi-square random numbers: `stats.chi2.rvs(df, size=1000)`
- Student's t-distribution PDF: `stats.t.pdf(x, df)`
- Student's t-distribution CDF: `stats.t.cdf(x, df)`
- Generate Student's t random numbers: `stats.t.rvs(df, size=1000)`
- F-distribution PDF: `stats.f.pdf(x, dfn, dfd)`
- F-distribution CDF: `stats.f.cdf(x, dfn, dfd)`
- Generate F random numbers: `stats.f.rvs(dfn, dfd, size=1000)`

Hypothesis Testing

- One-sample t-test: `stats.ttest_1samp(data, popmean)`
- Independent two-sample t-test: `stats.ttest_ind(data1, data2)`
- Paired t-test: `stats.ttest_rel(data1, data2)`
- One-way ANOVA: `stats.f_oneway(data1, data2, data3)`
- Two-way ANOVA: `stats.f_oneway(*(group for name, group in data.groupby(['factor1', 'factor2'])))`
- Chi-square goodness of fit test: `stats.chisquare(observed, expected)`
- Chi-square test of independence: `stats.chi2_contingency(contingency_table)`
- Shapiro-Wilk test for normality: `stats.shapiro(data)`
- Anderson-Darling test for normality: `stats.anderson(data)`
- Kolmogorov-Smirnov test: `stats.kstest(data, 'norm')`
- Mann-Whitney U test: `stats.mannwhitneyu(data1, data2)`
- Wilcoxon signed-rank test: `stats.wilcoxon(data1, data2)`
- Kruskal-Wallis H-test: `stats.kruskal(data1, data2, data3)`
- Friedman test: `stats.friedmanchisquare(data1, data2, data3)`
- Levene's test for equality of variances: `stats.levene(data1, data2)`
- Bartlett's test for equality of variances: `stats.bartlett(data1, data2)`
- Fligner-Killeen test for equality of variances: `stats.fligner(data1, data2)`

By: Waleed Mousa

Correlation and Regression

- Pearson correlation coefficient: `stats.pearsonr(x, y)`
- Spearman rank correlation: `stats.spearmanr(x, y)`
- Kendall's tau: `stats.kendalltau(x, y)`
- Simple linear regression: `stats.linregress(x, y)`
- Multiple linear regression: `stats.linregress(X, y)`
- Polynomial regression: `np.polyfit(x, y, deg=2)`
- R-squared (coefficient of determination): `1 - (np.sum((y - y_pred)**2) / np.sum((y - np.mean(y))**2))`
- Adjusted R-squared: `1 - ((1 - r_squared) * (n - 1) / (n - k - 1))`
- F-statistic: `((r_squared / (k - 1)) / ((1 - r_squared) / (n - k)))`
- Durbin-Watson statistic: `stats.durbin_watson(residuals)`

Non-parametric Methods

- Kernel density estimation: `stats.gaussian_kde(data)`
- Bootstrap sample: `stats.bootstrap((data,), np.mean, n_resamples=1000)`
- Jackknife resampling: `stats.jackknife(data, np.mean)`
- Permutation test: `stats.permutation_test((data1, data2), stats.ttest_ind)`

Multivariate Analysis

- Principal Component Analysis: `from sklearn.decomposition import PCA; PCA().fit_transform(X)`
- Canonical correlation analysis: `from sklearn.cross_decomposition import CCA; CCA().fit(X, Y).transform(X, Y)`
- MANOVA: `from statsmodels.multivariate.manova import MANOVA; MANOVA.from_formula('y1 + y2 ~ group', data=data).mv_test()`
- Hotelling's T-squared test: `stats.hotelling_t2(X1, X2)`

Time Series Analysis

- Autocorrelation: `stats.autocorr(data)`
- Partial autocorrelation: `from statsmodels.tsa.stattools import pacf; pacf(data)`
- Augmented Dickey-Fuller test: `from statsmodels.tsa.stattools import adfuller; adfuller(data)`
- KPSS test: `from statsmodels.tsa.stattools import kpss; kpss(data)`

By: Waleed Mousa

- Granger causality test: `from statsmodels.tsa.stattools import grangercausalitytests; grangercausalitytests(data, maxlag=5)`

Bayesian Statistics

- Bayes factor: `stats.bayes_mvs(data)`
- Bayesian Information Criterion (BIC): `stats.bic(residuals)`
- Akaike Information Criterion (AIC): `stats.aic(residuals)`

Sampling and Experimental Design

- Simple random sample: `np.random.choice(population, size=n, replace=False)`
- Stratified sample: `from sklearn.model_selection import StratifiedShuffleSplit; StratifiedShuffleSplit(n_splits=1, test_size=0.3).split(X, y)`
- Cluster sample: `from sklearn.cluster import KMeans; KMeans(n_clusters=k).fit_predict(X)`
- Systematic sample: `population[::k]`
- Latin square design: `stats.latin_square(n)`

Power Analysis

- Power of t-test: `stats.ttest_ind_solve_power(effect_size=0.5, nobs1=100, alpha=0.05, ratio=1.0, alternative='two-sided')`
- Power of ANOVA: `stats.f_oneway_solve_power(dfnum=2, dfden=27, alpha=0.05, effect_size=0.25)`
- Sample size calculation for t-test: `stats.ttest_ind_solve_power(effect_size=0.5, power=0.8, alpha=0.05, ratio=1.0, alternative='two-sided')`

Reliability Analysis

- Cronbach's alpha: `from statsmodels.stats.inter_rater import fleiss_kappa; fleiss_kappa(data)`
- Intraclass correlation coefficient: `stats.ttest_ind(group1, group2)`

Effect Size Calculations

By: Waleed Mousa

- Cohen's d: `(np.mean(group1) - np.mean(group2)) / np.sqrt((np.std(group1, ddof=1)**2 + np.std(group2, ddof=1)**2) / 2)`
- Eta-squared: `ss_effect / (ss_effect + ss_error)`
- Odds ratio: `(a * d) / (b * c)`
- Risk ratio: `(a / (a + b)) / (c / (c + d))`

Data Transformation

- Z-score normalization: `stats.zscore(data)`
- Min-max scaling: `(data - np.min(data)) / (np.max(data) - np.min(data))`
- Box-Cox transformation: `stats.boxcox(data)`
- Yeo-Johnson transformation: `stats.yeojohnson(data)`
- Logarithmic transformation: `np.log1p(data)`

Outlier Detection

- Z-score method: `np.abs(stats.zscore(data)) > 3`
- Interquartile range (IQR) method: `(data < Q1 - 1.5 * IQR) | (data > Q3 + 1.5 * IQR)`
- Modified Z-score method: `0.6745 * (data - np.median(data)) / stats.median_abs_deviation(data) > 3.5`
- Grubbs' test: `stats.grubbs(data)`

Confidence Intervals

- Normal distribution CI: `stats.norm.interval(alpha=0.95, loc=np.mean(data), scale=stats.sem(data))`
- T-distribution CI: `stats.t.interval(alpha=0.95, df=len(data)-1, loc=np.mean(data), scale=stats.sem(data))`
- Binomial proportion CI: `stats.binom.interval(n=len(data), p=np.mean(data), alpha=0.05)`
- Poisson CI: `stats.poisson.interval(alpha=0.95, mu=np.mean(data))`

Survival Analysis

- Kaplan-Meier estimator: `from lifelines import KaplanMeierFitter; KaplanMeierFitter().fit(durations, event_observed)`
- Cox proportional hazards model: `from lifelines import CoxPHFitter; CoxPHFitter().fit(df, duration_col='T', event_col='E')`

By: Waleed Mousa

- Log-rank test: `from lifelines.statistics import logrank_test; logrank_test(durations_1, durations_2, event_observed_1, event_observed_2)`

Spatial Statistics

- Moran's I: `from pysal.explore import esda; esda.Moran(y, w).I`
- Geary's C: `from pysal.explore import esda; esda.Geary(y, w).C`
- Getis-Ord G: `from pysal.explore import esda; esda.G(y, w).G`

Multivariate Normality Tests

- Mardia's test: `from statsmodels.stats.multivariate_normal import mardia; mardia(data)`
- Henze-Zirkler test: `from statsmodels.stats.multivariate_normal import henze_zirkler; henze_zirkler(data)`

Robust Statistics

- Median absolute deviation: `stats.median_abs_deviation(data)`
- Huber's M-estimator: `from statsmodels.robust import scale; scale.huber(data)`
- Theil-Sen estimator: `from scipy.stats import theilslopes; theilslopes(y, x)`

Factor Analysis

- Exploratory Factor Analysis: `from factor_analyzer import FactorAnalyzer; FactorAnalyzer().fit(data)`
- Confirmatory Factor Analysis: `from statsmodels.stats.factor import FactorAnalysis; FactorAnalysis().fit(data)`

Cluster Analysis

- K-means clustering: `from sklearn.cluster import KMeans; KMeans(n_clusters=k).fit(X)`
- Hierarchical clustering: `from scipy.cluster.hierarchy import linkage; linkage(X, method='ward')`
- DBSCAN clustering: `from sklearn.cluster import DBSCAN; DBSCAN().fit(X)`

By: Waleed Mousa

Time Series Decomposition

- Seasonal decomposition: `from statsmodels.tsa.seasonal import seasonal_decompose; seasonal_decompose(data, model='additive')`

Statistical Process Control

- Control chart (X-bar chart): `from statsmodels.stats.stattools import control_chart; control_chart(data)`

Meta-Analysis

- Fixed effects meta-analysis: `from statsmodels.stats.meta_analysis import CombineResults; CombineResults.combine_effects(effects, variances)`
- Random effects meta-analysis: `from statsmodels.stats.meta_analysis import CombineResults; CombineResults.combine_effects(effects, variances, method='random')`

Structural Equation Modeling

- Path analysis: `from statsmodels.stats.sem import SEM; SEM.from_formula('y ~ x1 + x2', data=data).fit()`

Item Response Theory

- 1PL (Rasch) model: `from psychometrics import irt; irt.twopl(difficulty, discrimination=1, ability)`
- 2PL model: `from psychometrics import irt; irt.twopl(difficulty, discrimination, ability)`

Multilevel Modeling

- Random intercept model: `from statsmodels.regression.mixed_linear_model import MixedLM; MixedLM.from_formula('y ~ x', groups='group', data=data).fit()`

Statistical Quality Control

- Capability analysis: `from statsmodels.stats.stattools import cpk_index; cpk_index(data, lower=lsl, upper=usl)`
- Process capability index: `(usl - lsl) / (6 * np.std(data, ddof=1))`

By: Waleed Mousa

Nonlinear Regression

- Curve fitting: `from scipy.optimize import curve_fit; curve_fit(lambda x, a, b: a * np.exp(b * x), x_data, y_data)`

Statistical Tests for Circular Data

- Rayleigh test: `from scipy.stats import rayleigh; rayleigh.fit(data)`
- Watson's U2 test: `from astropy.stats import watson_u2; watson_u2(data)`

Extreme Value Analysis

- Generalized extreme value distribution fit: `from scipy.stats import genextreme; genextreme.fit(data)`
- Peak over threshold analysis: `from scipy.stats import genpareto; genpareto.fit(data[data > threshold])`

Functional Data Analysis

- Functional principal component analysis: `from skfda.decomposition import FPCA; FPCA().fit_transform(data)`

Statistical Learning Theory

- Support Vector Machine: `from sklearn.svm import SVC; SVC().fit(X, y)`
- Cross-validation: `from sklearn.model_selection import cross_val_score; cross_val_score(model, X, y, cv=5)`

Copulas

- Gaussian copula: `from scipy.stats import multivariate_normal; multivariate_normal.cdf(data)`
- Clayton copula: `from copulas.multivariate import GaussianMultivariate; GaussianMultivariate().fit(data).probability_density(data)`

Stochastic Processes

- Brownian motion simulation: `np.cumsum(np.random.normal(0, 1, size=1000))`

By: Waleed Mousa

- Ornstein-Uhlenbeck process: `from scipy.integrate import odeint; odeint(lambda y, t, theta, mu, sigma: theta * (mu - y), y0, t, args=(theta, mu, sigma))`

Causal Inference

- Propensity score matching: `from sklearn.linear_model import LogisticRegression; LogisticRegression().fit(X, treatment).predict_proba(X)[:, 1]`
- Difference-in-differences estimation: `np.mean(post_treatment - pre_treatment) - np.mean(post_control - pre_control)`

Spatial Point Pattern Analysis

- Ripley's K function: `from astropy.stats import RipleysKEstimator; RipleysKEstimator(area=area).evaluate(data)`

Statistical Network Analysis

- Erdős-Rényi random graph model: `from networkx.generators.random_graphs import erdos_renyi_graph; erdos_renyi_graph(n, p)`