AutoStats Pro

A Web-Based Platform for Automated Biostatistical Analysis in Life Sciences Research

Author:	AutoStats Pro Developing Team
Version:	v1.1
Date:	May 2026
DOI (software):	https://doi.org/10.5281/zenodo.20556901
Application URL:	https://app.autostats-pro.com
License:	Proprietary — all rights reserved
Document type:	Technical Report

Abstract. AutoStats Pro is a web-based platform for automated biostatistical analysis targeted at life sciences researchers. Built with Python and Streamlit, it provides a browser-accessible interface that guides users from raw data upload to exportable PDF report without requiring programming expertise or advanced statistical knowledge. The platform covers six analytical modules: hypothesis testing with automatic parametric/non-parametric selection, correlation and regression, multivariate analysis (PCA, K-means, hierarchical clustering), survival analysis (Kaplan-Meier, Cox regression), omics differential expression analysis (volcano plots, FDR correction, heatmaps), and structured PDF report generation. User data is never stored persistently.

1. Summary

AutoStats Pro is a web-based platform for automated biostatistical analysis targeted at life sciences researchers. Built with Python and Streamlit, it provides a browser-accessible interface that guides users from raw data upload to exportable PDF report without requiring programming expertise or advanced statistical knowledge. The platform automatically detects dataset structure and orientation, selects appropriate statistical tests based on normality and variance assessments, and generates interpretive narratives alongside publication-ready figures.

AutoStats Pro covers six analytical modules: (1) hypothesis testing with automatic parametric/non-parametric selection and effect size estimation; (2) correlation and regression analysis; (3) multivariate dimensionality reduction and clustering; (4) survival analysis; (5) omics differential expression analysis; and (6) exportable PDF reporting. All computation is performed server-side using established scientific Python libraries, with user data never stored persistently.

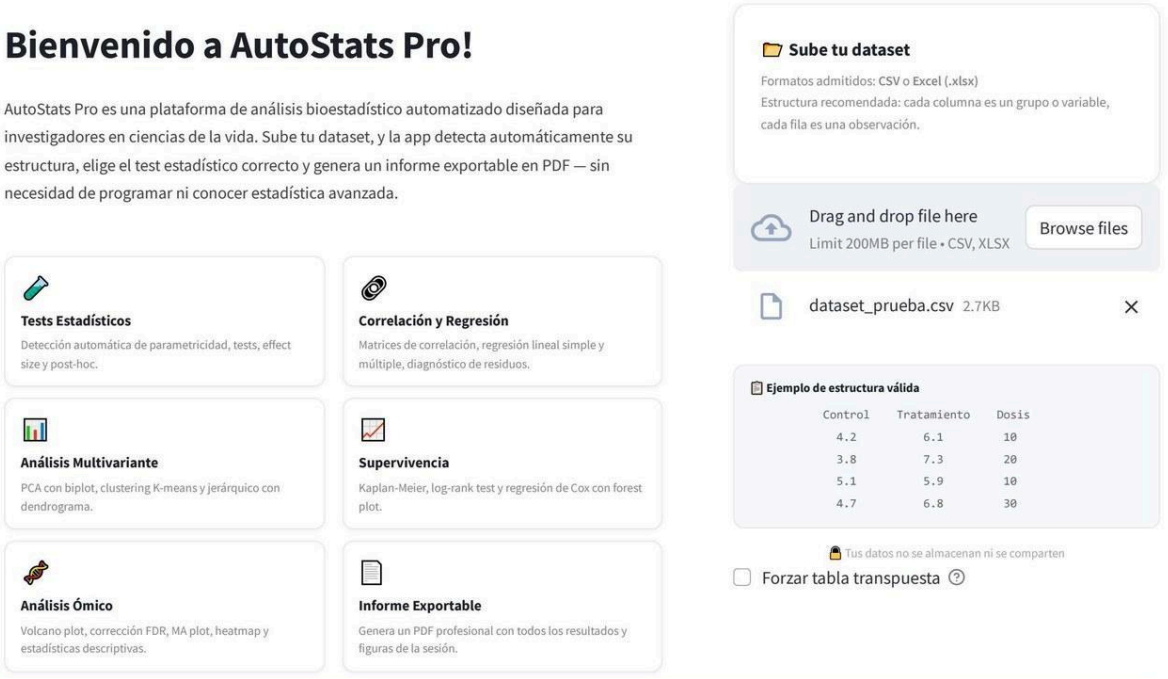

Figure 1. AutoStats Pro homepage showing the six analytical modules and dataset upload panel.

2. Statement of Need

Quantitative analysis is central to modern life sciences research, yet a persistent gap exists between the statistical methods available in software libraries and the analytical capacity of many biomedical researchers. Studies have documented widespread statistical errors in published biomedical literature, including inappropriate test selection, failure to assess distributional assumptions, neglect of multiple testing correction, and misinterpretation of effect sizes (Makin & Orban de Xivry, 2019).

Existing solutions occupy two distinct positions: fully featured statistical packages such as R, GraphPad Prism, and SPSS require either scripting knowledge or institutional licenses that may be inaccessible to independent researchers, graduate students, and small research groups. Simplified online calculators handle isolated tests but cannot support the multimodal, multi-step analyses typical of real experimental workflows, nor do they provide integrated reporting.

AutoStats Pro addresses this gap by integrating a complete biostatistical workflow — data ingestion, quality assessment, test selection, effect size estimation, post-hoc analysis, and multi-module reporting — into a single, license-free web application. The platform is particularly suited to researchers in molecular biology, clinical research, neuroscience, genomics, and related fields who require reproducible statistical outputs without the overhead of a programming environment.

3. State of the Field

The landscape of accessible biostatistical tools includes several notable packages. *pingouin* (Vallat, 2018) is a Python library offering a broad set of statistical tests with a clean API, but it requires scripting proficiency and provides no graphical interface or integrated reporting. *scipy.stats* (Virtanen et al., 2020) and *statsmodels* (Seabold & Perktold, 2010) similarly provide comprehensive statistical functions within Python but are library-level tools. *lifelines* (Davidson-Pilon, 2019) specialises in survival analysis and underpins AutoStats Pro's Kaplan-Meier and Cox regression modules.

At the graphical interface level, JASP and JAMOVI provide point-and-click statistical analysis but focus primarily on social science and psychology workflows rather than life sciences-specific analyses (survival curves, volcano plots, omics correction), and do not offer browser-based deployment without local installation. Galaxy (Afgan et al., 2018) provides a bioinformatics workflow platform but targets sequence analysis pipelines rather than exploratory biostatistical reporting.

AutoStats Pro fills a specific niche: multi-module biostatistical analysis from a single uploaded tabular dataset to a citeable PDF report, in a unified session context, accessible from any browser without installation.

4. Software Design

AutoStats Pro is implemented as a single-page Streamlit application with six analytical modules rendered as interactive tabs. Session state in Streamlit's in-memory store manages cross-module data flow, allowing figures and inference outputs generated in any module to be accumulated and rendered into the final report without persistent server-side storage. The core design trade-off was between a fully modular importable library and a self-contained deployable application; the latter was prioritised to maximise accessibility for non-programming users.

Dataset ingestion and orientation. Upon upload, the platform reads CSV files via pandas.read_csv and Excel files via pandas.read_excel. Before any statistical operation, an automatic orientation heuristic evaluates whether the data are structured column-wise (each column is a group or variable, each row an observation — the conventional layout) or row-wise (transposed). The heuristic counts numeric series of consistent length in both orientations; if transposing yields more valid groups, the dataset is silently inverted. The user retains a manual override checkbox. All values are coerced to numeric via pd.to_numeric(errors='coerce') prior to analysis, and missing values are excluded listwise at the point of each computation.

Normality and variance assessment (Module 1). The Shapiro-Wilk test (Shapiro & Wilk, 1965), implemented via scipy.stats.shapiro, is applied independently to each group after removing NaN values. The test is only executed when the group contains at least three valid observations; groups below this threshold are flagged as non-assessable and the non-parametric path is forced.

The test statistic W and its associated p-value are reported for each group, with normality declared at $\alpha = 0.05$. When all groups pass Shapiro-Wilk, Levene's test for homogeneity of variance is applied across all groups via scipy.stats.levene, using the default median-based Brown-Forsythe variant, which is more robust to non-normality than the mean-based formulation. The result of Levene's test ($p > \alpha \rightarrow$ homogeneous variances) then governs the specific parametric test selected.

Parametric test selection (Module 1). For two-group independent comparisons in which both groups are normally distributed: if Levene's test confirms variance homogeneity, scipy.stats.ttest_ind(equal_var=True) is applied (Student's t-test); if variances are heterogeneous, scipy.stats.ttest_ind(equal_var=False) is applied (Welch's t-test, which does not assume a common variance and uses Satterthwaite's degrees-of-freedom approximation). For paired designs, scipy.stats.ttest_rel is substituted when both groups are normal. For three or more groups all passing normality, scipy.stats.f_oneway performs a one-way ANOVA.

Non-parametric test selection (Module 1). When any group fails Shapiro-Wilk, the non-parametric path is taken regardless of sample size. For two independent groups, scipy.stats.mannwhitneyu(alternative='two-sided') computes the Mann-Whitney U statistic; for two paired groups, scipy.stats.wilcoxon computes the Wilcoxon signed-rank statistic on the pairwise differences. For three or more groups, scipy.stats.kruskal applies the Kruskal-Wallis H test, which is the non-parametric analogue of one-way ANOVA based on rank sums.

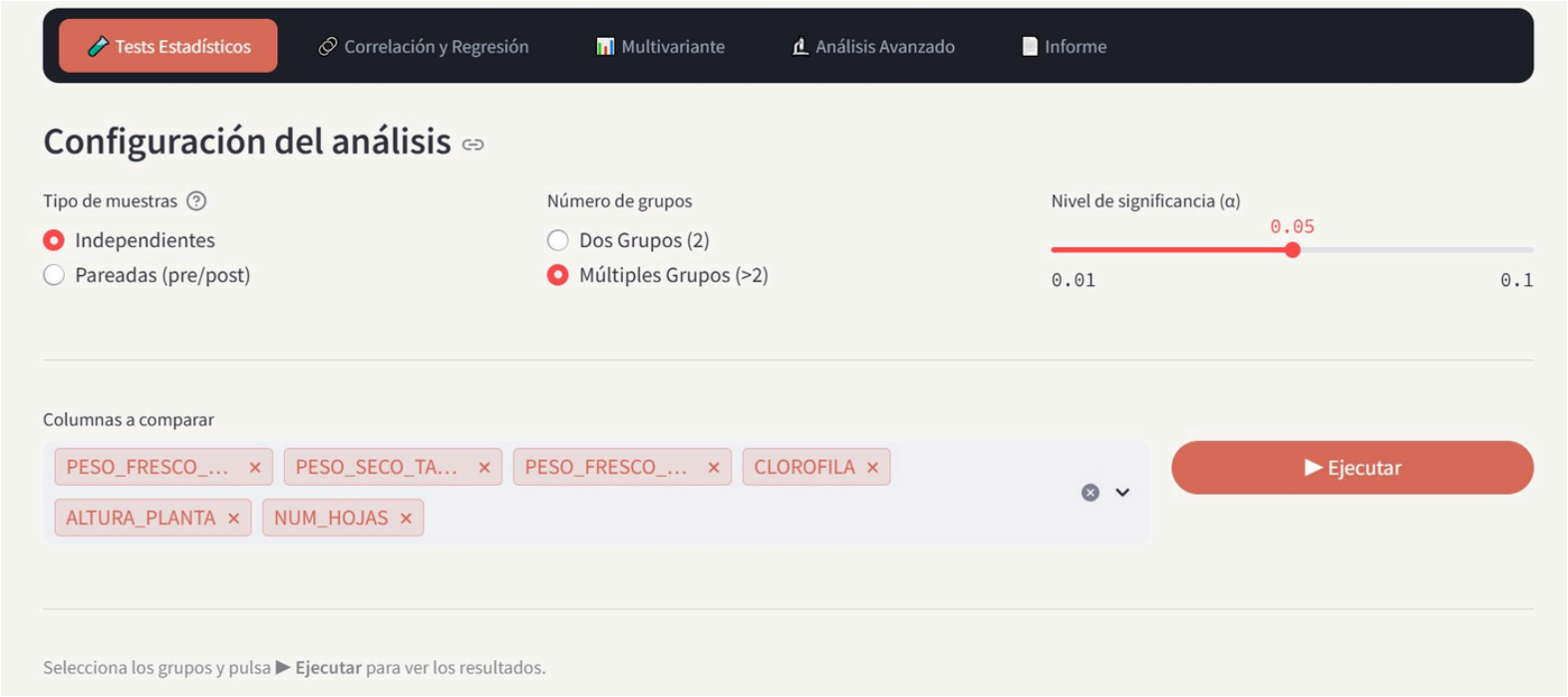

Figure 2. Module 1 configuration panel: sample type, group number, significance level (α), and variable selection.

Effect size estimation (Module 1). Effect sizes are computed for every analysis regardless of significance. For parametric two-group comparisons, Cohen's d is calculated using the pooled standard deviation formula with ddof=1 (unbiased sample standard deviations):

$$d = \frac{\bar{x}_A - \bar{x}_B}{s_{pooled}}$$

where,

$$s_{pooled} = \sqrt{\frac{(n_A - 1)s_A^2 + (n_B - 1)s_B^2}{n_A + n_B - 2}}$$

For non-parametric two-group comparisons, the rank-biserial correlation r is derived from the Mann-Whitney U statistic:

$$r = 1 - \frac{2U}{n_A \cdot n_B}$$

which provides a bounded [-1, 1] effect size measure. For multi-group comparisons (both ANOVA and Kruskal-Wallis), eta-squared is computed as:

$$\eta^2 = \frac{SS_{between}}{SS_{total}}$$

where between-group sum of squares is calculated from group means against the grand mean.

Effect size thresholds follow Cohen's (1988) conventions: for d, negligible < 0.2, small < 0.5, medium < 0.8, large ≥ 0.8; for r, negligible < 0.1, small < 0.3, medium < 0.5, large ≥ 0.5; for η^2, negligible < 0.01, small < 0.06, medium < 0.14, large ≥ 0.14.

Post-hoc multiple comparisons (Module 1). Post-hoc tests are executed automatically when the omnibus test is significant ($p < \alpha$). For one-way ANOVA, Tukey's Honestly Significant Difference (HSD) test is implemented directly: within-group mean square (MS_within) is computed from the pooled within-group sum of squares divided by degrees of freedom ($N - k$), and pairwise differences are evaluated as studentised range statistics:

$$q = \frac{|\bar{y}_i - \bar{y}_j|}{\sqrt{MS_{within} \cdot \frac{1}{2}\left(\frac{1}{n_i} + \frac{1}{n_j}\right)}}$$

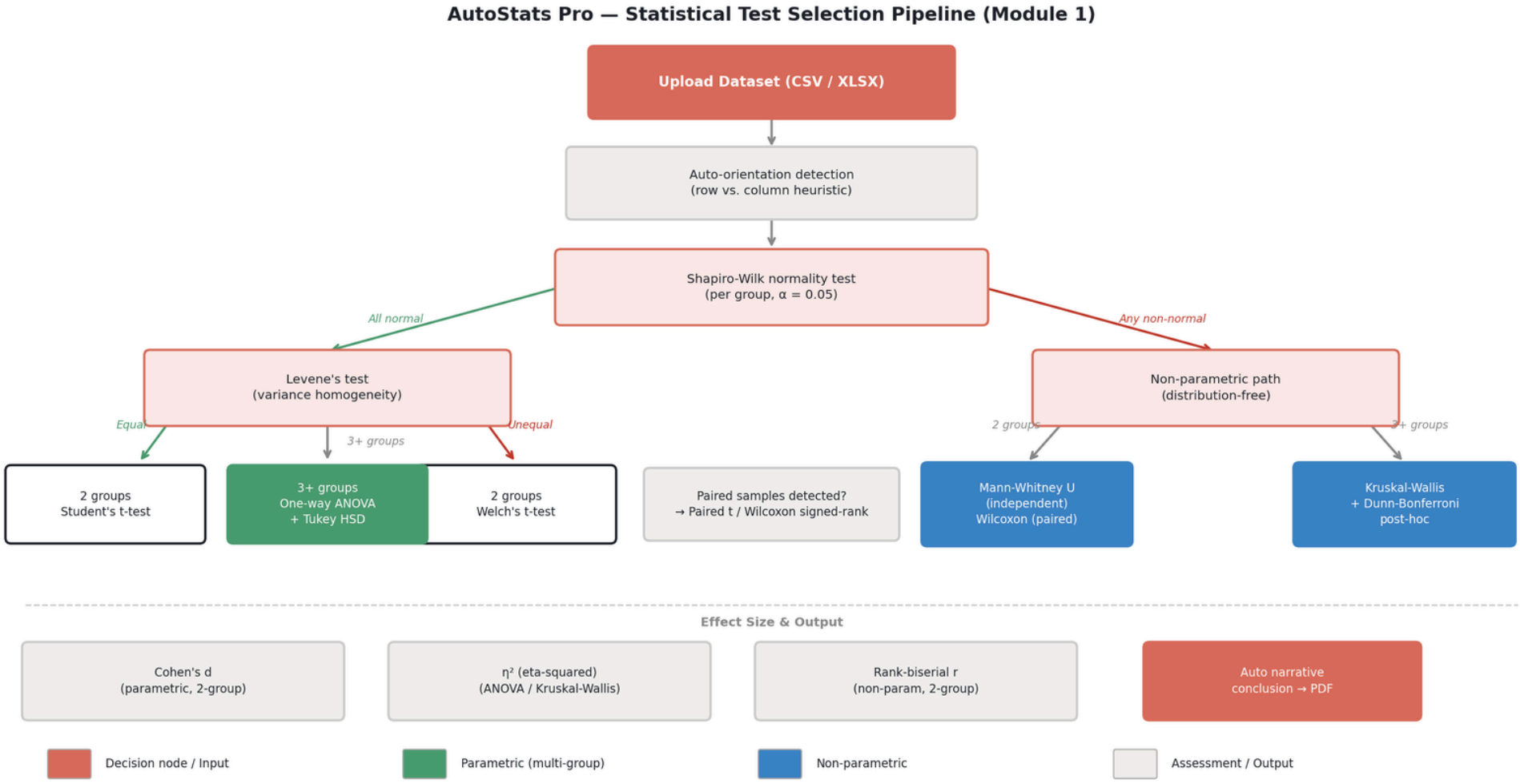

Figure 3. Automated test selection pipeline implemented in Module 1. Normality and variance assessments drive the parametric/non-parametric fork. Effect sizes are computed for all outcomes.

p-values are approximated using the standard normal survival function applied to q/√2. For the Kruskal-Wallis test, Dunn's test with Bonferroni correction is implemented manually: all observations are jointly ranked via scipy.stats.rankdata, mean ranks are computed per group, and pairwise z-statistics are derived as:

$$z = \frac{|\bar{R}_i - \bar{R}_j|}{\sqrt{\frac{N(N+1)}{12}\left(\frac{1}{n_i} + \frac{1}{n_j}\right)}}$$

Raw p-values are multiplied by the number of comparisons and clipped to 1.0.

Correlation and regression (Module 2). The correlation module selects between Pearson and Spearman methods based on a Shapiro-Wilk assessment applied independently to each variable in the selected set (minimum 3 observations per variable required); if all variables pass normality, pandas.DataFrame.corr(method='pearson') is used, otherwise corr(method='spearman'). The user may also override this selection manually. Pairwise p-values are computed explicitly for each variable pair using scipy.stats.pearsonr or scipy.stats.spearmanr on the listwise-complete subset for that pair. Linear regression is fitted via sklearn.linear_model.LinearRegression, which minimises the ordinary least squares (OLS) criterion. Both R^2 and adjusted R^2 are reported:

$$R^2_{adj} = 1 - \frac{(1 - R^2)(n - 1)}{n - k - 1}$$

where k is the number of predictors. The overall model F-statistic and its p-value are computed from the regression and residual sums of squares using scipy.stats.f.sf. Residuals are subjected to a secondary Shapiro-Wilk test and a Q-Q probability plot to assess OLS assumptions.

Multivariate analysis — PCA (Module 3). All input variables are standardised to zero mean and unit variance using sklearn.preprocessing.StandardScaler prior to PCA, as PCA is sensitive to scale. sklearn.decomposition.PCA is then applied, retaining all components up to min(n_variables, n_observations). The proportion of variance explained per component is extracted from explained_variance_ratio_, and eigenvalues from explained_variance_. A biplot superimposes observation scores (PC1 vs PC2) with loading vectors scaled to the range of the score cloud; each loading vector represents the contribution of an original variable to the two principal components. A loading table displaying all component loadings is provided for inspection.

Multivariate analysis — clustering (Module 3). All clustering inputs are standardised with StandardScaler. For K-means, sklearn.cluster.KMeans is fitted with n_init=10 (ten random initialisations to mitigate local optima sensitivity) and random_state=42 for reproducibility; the within-cluster sum of squares (inertia) is computed across k = 2 to min(10, n−1) to assist elbow-method selection. For

hierarchical clustering, scipy.cluster.hierarchy.linkage computes the linkage matrix with user-selectable linkage criteria (Ward, complete, average, single); Ward's method, which minimises within-cluster variance at each merge, is the default recommendation for biological data. Cluster assignments are derived via sklearn.cluster.AgglomerativeClustering. In both cases, cluster memberships are projected onto two dimensions using a secondary PCA for visualisation.

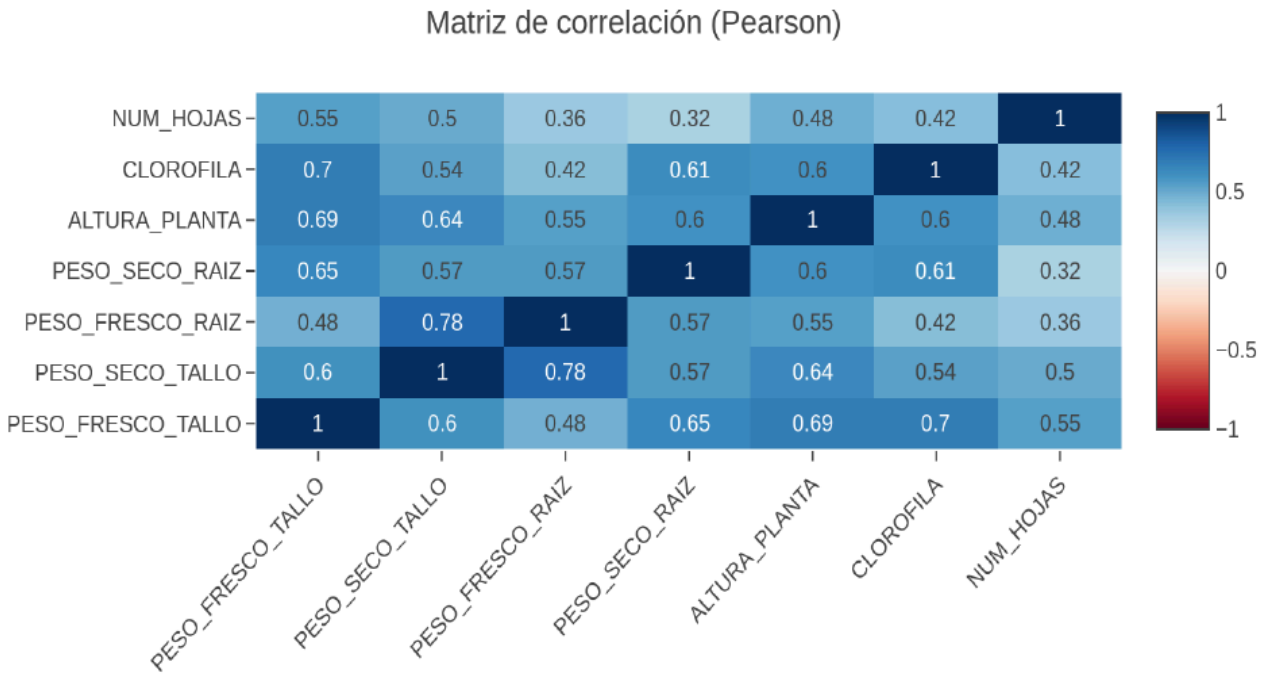

Figure 4. Pearson correlation matrix heatmap generated by Module 2, with automatic method selection driven by Shapiro-Wilk assessment.

Survival analysis (Module 4). Kaplan-Meier survival functions are estimated using lifelines.KaplanMeierFitter.fit, which implements the product-limit estimator:

$$\hat{S}(t) = \prod_{t_i \leq t} \left(1 - \frac{d_i}{n_i}\right)$$

Where d_i is the number of events and n_i the number at risk at time t_i. Greenwood's formula is used internally by lifelines to compute 95% confidence intervals. Censored observations are marked on the curve as tick marks at their last known time. For two-group comparisons, lifelines.statistics.logrank_test computes the log-rank statistic as a weighted sum of observed-minus-expected events across all event times; for three or more groups, multivariate_logrank_test extends this to a chi-squared statistic. Cox proportional hazards regression is fitted via lifelines.CoxPHFitter, which maximises the partial likelihood. Hazard ratios (HR) are obtained as the exponentiated coefficients, and 95% confidence intervals are derived from the variance-covariance matrix of the partial likelihood estimator. Model fit is assessed by Harrell's concordance index (C-index). Proportional hazards assumption is tested via lifelines.statistics.proportional_hazard_test using rank-transformed time (Schoenfeld residuals approach); p > 0.05 indicates the assumption holds for a given covariate.

Omics analysis (Module 5). The volcano plot classifies each feature by applying simultaneous thresholds on $\log_2$ fold change ($|\log_2 FC| \geq$ user-defined threshold, default 1.0) and statistical significance ($-\log_{10}(p) \geq$ threshold, default 1.301, corresponding to $p < 0.05$). Features satisfying both thresholds and positive $\log_2 FC$ are labelled "Up-regulated"; those satisfying both with negative $\log_2 FC$ are "Down-regulated"; those meeting only the significance threshold are "Significant (no FC)". p-values are clipped to a minimum of 10^{-300} before log-transformation to avoid numerical overflow. Multiple testing correction is applied via statsmodels.stats.multitest.multipletests, supporting Benjamini-Hochberg FDR (method='fdr_bh'), Bonferroni (method='bonferroni'), and Holm-Bonferroni (method='holm'). BH-FDR controls the expected proportion of false discoveries among rejected hypotheses and is the recommended default for omics data. Heatmap visualisation optionally applies per-row z-score standardisation:

$$z_{ij} = \frac{x_{ij} - \bar{x}_i}{s_i}$$

Where the mean and standard deviation are computed across samples for each gene; this centres each gene's expression and makes cross-gene comparison of relative patterns interpretable. Genes are ranked by variance across samples and the top-n most variable features are displayed, prioritising biologically informative entries.

5. Research Impact Statement

AutoStats Pro was released in May 2026 and is accessible at https://app.autostats-pro.com. The software is registered on Zenodo with DOI 10.5281/zenodo.20556901, enabling formal citation in publications that use it for statistical analysis. The platform operates as a production application with active subscribers across individual researcher and laboratory team plans.

The platform is particularly relevant for research groups generating tabular data in clinical trials, transcriptomics, proteomics, metabolomics, and cohort studies who require a reproducible, documented analytical pipeline without dedicated bioinformatician support. The integrated PDF reporting with software citation metadata is designed to support reproducibility standards increasingly required by journals and funding bodies.

The six-module architecture means a single upload can yield a complete statistical characterisation — distributional assessment, group comparisons with effect sizes, correlation structure, dimensionality reduction, and where applicable, time-to-event analysis — within a single session, eliminating the tool-switching overhead that introduces errors and reduces reproducibility in multi-step analyses.

6. Acknowledgements

The author acknowledges the developers of the open-source libraries forming the computational foundation of AutoStats Pro: *scipy*, *scikit-learn*, *lifelines*, *statsmodels*, *plotly*, *pandas*, *numpy*, and *ReportLab*. No external funding was received.

References

Afgan, E., et al. (2018). The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Research*, 46(W1), W537–W544. https://doi.org/10.1093/nar/gky379

Davidson-Pilon, C. (2019). lifelines: survival analysis in Python. *Journal of Open Source Software*, 4(40), 1317. https://doi.org/10.21105/joss.01317

Gentleman, R. C., et al. (2004). Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5(10), R80. https://doi.org/10.1186/gb-2004-5-10-r80

Illanas, L. (2026). *AutoStats Pro* (v1.1) [Software]. Zenodo. https://doi.org/10.5281/zenodo.20556901

JASP Team. (2023). *JASP* (Version 0.17) [Computer software]. https://jasp-stats.org/

Levene, H. (1960). Robust tests for equality of variances. In I. Olkin et al. (Eds.), *Contributions to Probability and Statistics* (pp. 278–292). Stanford University Press.

Love, J., et al. (2019). *JAMOVI* [Computer software]. https://www.jamovi.org

Makin, T. R., & Orban de Xivry, J.-J. (2019). Ten common statistical mistakes to watch out for when writing or reviewing a manuscript. *eLife*, 8, e48175. https://doi.org/10.7554/eLife.48175

Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

R Core Team. (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing.

Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with Python. *Proceedings of the 9th Python in Science Conference*, 92–96. https://doi.org/10.25080/Majora-92bf1922-011

Shapiro, S. S., & Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, 52(3–4), 591–611. https://doi.org/10.1093/biomet/52.3-4.591

Therneau, T. M., & Grambsch, P. M. (2000). *Modeling Survival Data: Extending the Cox Model*. Springer. https://doi.org/10.1007/978-1-4757-3294-8

Vallat, R. (2018). Pingouin: statistics in Python. *Journal of Open Source Software*, 3(31), 1026. https://doi.org/10.21105/joss.01026

Virtanen, P., et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17, 261–272. https://doi.org/10.1038/s41592-019-0686-2