Jan 14, 2026 Version 2

Automated Mean ± SEM tables with statistical grouping letters in Python (pairwise tests, tidy data) V.2

DOI
https://dx.doi.org/10.17504/protocols.io.261ge13zjv47/v2

Carla Sandri[1]

[1]University of Tuscia, DAFNE, Viterbo

CarlaSandri

C Carla Sandri

DOI: https://dx.doi.org/10.17504/protocols.io.261ge13zjv47/v2

Protocol Citation: Carla Sandri 2026. Automated Mean ± SEM tables with statistical grouping letters in Python (pairwise tests, tidy data). **protocols.io** https://dx.doi.org/10.17504/protocols.io.261ge13zjv47/v2Version created by **Carla Sandri**

Protocol status: Working
We use this protocol and it's working

Created: January 13, 2026

Last Modified: January 14, 2026

Protocol Integer ID: 238545

Keywords: Mean ± SEM, compact letter display, statistical clustering letters, post-hoc, paired t-test, plant phenotyping, metabolomics, microgreens, Python, reproducible tables, statistical grouping letters in python, pairwise significance pattern into clustering letter, statistical grouping letter, letters for each experimental group, clustering letter, pairwise significance pattern, tidy data, pairwise test, ready summary table, sem, superscript letter, experimental group, reproducible python workflow, python, pairwise differences between treatment, other factorial structure, format table, pairwise difference, test, row per observation, summary table, pairwise

Disclaimer

This protocol provides a reproducible workflow for generating "Mean ± SEM + Letter Statistics" tables from tidy/long data. Statistical interpretation depends on the assumptions of the test used (e.g., independence, variances) and the choice of corrections for multiple comparisons. Users should verify the appropriateness of the test for their experimental design.

Abstract

This protocol describes a reproducible Python workflow for generating publication-ready summary tables reporting mean ± SEM and statistical clustering letters for each experimental group. Starting from a tidy-long format CSV file (one row per observation), the workflow (i) calculates the group mean and standard error (SEM), (ii) evaluates pairwise differences between treatments (e.g., independent two-sample t-tests), (iii) converts the pairwise significance pattern into clustering letters (compact letter display concept), and (iv) exports a final large-format table in which each cell contains the mean ± SEM followed by superscript letters. The protocol is designed for multi-treatment datasets (e.g., A–I) and can be adapted to other factorial structures.

Guidelines

Letters are a compact representation of pairwise significance patterns; always specify:
- the statistical test used
- the correction method
- the alpha threshold
- whether the tests were per metabolite or global.

Materials

Hardware
- Any standard laptop/desktop.

Software
- Python 3.9+ (3.10+ recommended)
- Jupyter Notebook (or JupyterLab) or any environment capable of running .ipynb

Python Packages
- numpy
- pandas
- scipy

Note: If you later implement ANOVA/Tukey or mixed models, you may also need statsmodels.

Input File and Data Format: Required Input (sorted CSV)
A **CSV** file with at least the following columns:
- *Metabolite* (or *Trait_): Trait name (string)*
- *Treatment_: Group identifier (string; for example, A, B, C... or NT, 20N, 40N)*
- *Replicate_: Replicate identifier (string or integer)*
- *Value_: Numeric measurement*

Example file name: *metabolomics_example_A-I_tidy.csv_*
A synthetic example dataset corresponding to this format is provided at the end of the Materials section.

Optional intermediate input/output
- *ttest_all_pairs.csv_: File containing the results of the pairwise tests used to construct the letter table (generated during the workflow)*

Output
_mean_sem_with_letters.csv_:

Final table in broad format
- Rows: Metabolite/Trait
- Columns: Treatments
- Cell contents: Mean ± SEM + superscript letters (e.g., *1.234e-02 ± 3.210e-03^^ab^^_)*

Python script (Jupyter notebook – text version)

```
import numpy as np
import pandas as pd
from itertools import combinations
from scipy.stats import ttest_ind
```

```
## IMPORT DATA (CSV tidy)
# Expected columns: Metabolite, Treatment, Replicate, Value
df = pd.read_csv("metabolomics_example_A-I_tidy.csv")

required_cols = {"Metabolite", "Treatment", "Replicate", "Value"}
if not required_cols.issubset(df.columns):
    raise ValueError(f"CSV must contain columns: {required_cols}")

## CONFIGURATION
labels = ["A", "B", "C", "D", "E", "F", "G", "H", "I"]
def p_to_stars(p):
    """
    Assegna gli asterischi in base al p-value:
    *    p < 0.05
    **   p < 0.01
    ***  p < 0.001
    **** p < 0.0001
    """
    if p < 0.0001:
        return "****"
    elif p < 0.001:
        return "***"
    elif p < 0.01:
        return "**"
    elif p < 0.05:
        return "*"
    else:
        return ""

## T-TEST
rows = []
for metabolite, sub in df.groupby("Metabolite"):

    groups = {
        t: sub[sub["Treatment"] == t]["Value"].dropna().values
        for t in labels
    }

    for g1, g2 in combinations(labels, 2):   # all combination A vs B, A vs C, ...
        x1 = groups[g1]
        x2 = groups[g2]

        if len(x1) < 2 or len(x2) < 2:
```

```
            p_val = np.nan
            stars = ""
        else:
            _, p_val = ttest_ind(x1, x2, equal_var=True)
            stars = p_to_stars(p_val)

        rows.append({
            "Metabolite": metabolite,
            "Group1": g1,
            "Group2": g2,
            "p_value": p_val,
            "significance": stars
        })

## EXPORT
ttest_df = pd.DataFrame(rows)
ttest_df.to_csv("ttest_all_pairs.csv", index=False)
print("File creato: ttest_all_pairs.csv")

## LETTERS TABLE
import string
df = pd.read_csv("ttest_all_pairs.csv")

df["p_value"] = pd.to_numeric(df["p_value"], errors="coerce")
df["Group1"] = df["Group1"].astype(str)
df["Group2"] = df["Group2"].astype(str)

treatments = ["A", "B", "C", "D", "E", "F", "G", "H", "I"]

alpha = 0.05
all_letters = list(string.ascii_lowercase)

def get_p_matrix(sub):
    P = {}
    for _, r in sub.iterrows():
        g1, g2 = r["Group1"], r["Group2"]
        P[(g1, g2)] = r["p_value"]
        P[(g2, g1)] = r["p_value"]
    return P

# LETTER ASSIGNMENT (SEQUENTIAL)
def assign_letters_for_metabolite(pmat, treatments):
```

```
letters = {t: "" for t in treatments}

# First treatment gets "a"
letters[treatments[0]] = "a"
used_letters = ["a"]

# Iterate over remaining treatments
for i in range(1, len(treatments)):
    T = treatments[i]
    prev = treatments[:i]

    assigned = False

    # Try existing letters first
    for L in used_letters:
        ok = True

        for P in prev:
            pval = pmat.get((T, P), 1.0)  # default = not significant

            # If pval is NaN, treat as "unknown" → safest is to not force separation
            if pd.isna(pval):
                pval = 1.0

            if pval < alpha:
                # significant → must NOT share this letter with P if P already has it
                if L in letters[P]:
                    ok = False
                    break
            else:
                # not significant → sharing is allowed
                pass

        if ok:
            letters[T] = L

            # Retroactively add letter to previous groups that are not significantly different
            for P in prev:
                pval = pmat.get((T, P), 1.0)
                if pd.isna(pval):
                    pval = 1.0
                if pval >= alpha and L not in letters[P]:
                    letters[P] += L
```

```
            assigned = True
            break

        # If no existing letter fits, create a new one
        if not assigned:
            new_L = all_letters[len(used_letters)]
            used_letters.append(new_L)
            letters[T] = new_L

            # Retroactively add new letter to previous groups not significantly different
            for P in prev:
                pval = pmat.get((T, P), 1.0)
                if pd.isna(pval):
                    pval = 1.0
                if pval >= alpha:
                    letters[P] += new_L

    # Sort and deduplicate letters per treatment
    for t in letters:
        letters[t] = "".join(sorted(set(letters[t])))

    return letters

# OPTIONAL CHECK: missing pairs
def count_missing_pairs(pmat, treatments):
    missing = 0
    for i in range(len(treatments)):
        for j in range(i+1, len(treatments)):
            a, b = treatments[i], treatments[j]
            if (a, b) not in pmat or pd.isna(pmat.get((a, b))):
                missing += 1
    return missing

output = []

for metabolite in df["Metabolite"].unique():
    sub = df[df["Metabolite"] == metabolite]

    pmat = get_p_matrix(sub)

    # check (optional but useful)
    miss = count_missing_pairs(pmat, treatments)
```

```
    if miss > 0:
        print(f"[WARN] {metabolite}: missing/NaN p-values for {miss} pair(s)")

    L = assign_letters_for_metabolite(pmat, treatments)

    row = {"Metabolite": metabolite}
    row.update(L)
    output.append(row)

letters_df = pd.DataFrame(output)

letters_df.to_csv("statistical_letters_sequential.csv", index=False)
print("Saved: statistical_letters_sequential.csv")
letters_df.head()

import numpy as np
import pandas as pd

df = pd.read_csv("metabolomics_example_A-I_tidy.csv")

required_cols = {"Metabolite", "Treatment", "Replicate", "Value"}
if not required_cols.issubset(df.columns):
    raise ValueError(f"CSV must contain columns: {required_cols}")

df["Treatment"] = df["Treatment"].astype(str)
df["Value"] = pd.to_numeric(df["Value"], errors="coerce")
df = df.dropna(subset=["Value"])

# CONFIG
treatments = ["A", "B", "C", "D", "E", "F", "G", "H", "I"]

SUPERSCRIPT = {
    "a":"ᵃ","b":"ᵇ","c":"ᶜ","d":"ᵈ","e":"ᵉ","f":"ᶠ","g":"ᵍ","h":"ʰ","i":"ᶦ","j":"ʲ",
    "k":"ᵏ","l":"ˡ","m":"ᵐ","n":"ⁿ","o":"ᵒ","p":"ᵖ","q":"ᵠ","r":"ʳ","s":"ˢ","t":"ᵗ",
    "u":"ᵘ","v":"ᵛ","w":"ʷ","x":"ˣ","y":"ʸ","z":"ᶻ"
}

def to_superscript(letters: str) -> str:
    letters = "" if pd.isna(letters) else str(letters)
    return "".join(SUPERSCRIPT.get(ch, "") for ch in letters)

# LOAD LETTERS TABLE
letters_df = pd.read_csv("statistical_letters_sequential.csv").set_index("Metabolite")
```

```
mets_in_data = set(df["Metabolite"].unique())
mets_in_letters = set(letters_df.index)
missing_letters = sorted(list(mets_in_data - mets_in_letters))
if missing_letters:
    print(f"[WARN] Missing letters for {len(missing_letters)} metabolite(s): {missing_letters[:5]} ...")

# MEAN & SEM
stats_df = (
    df.groupby(["Metabolite", "Treatment"])["Value"]
      .agg(mean="mean", sem=lambda x: x.std(ddof=1) / np.sqrt(len(x)))
      .reset_index()
)

# Enforce treatment order
stats_df["Treatment"] = pd.Categorical(stats_df["Treatment"], categories=treatments, ordered=True)
stats_df = stats_df.sort_values(["Metabolite", "Treatment"])

# Pivot to wide for mean and sem
mean_w = stats_df.pivot(index="Metabolite", columns="Treatment", values="mean")
sem_w  = stats_df.pivot(index="Metabolite", columns="Treatment", values="sem")

# FINAL TABLE (Mean ± SEM + letters)
final = pd.DataFrame(index=mean_w.index)

for t in treatments:
    mean_col = mean_w.get(t)
    sem_col = sem_w.get(t)

    if mean_col is None or sem_col is None:
        continue

    let = letters_df[t].reindex(mean_w.index) if t in letters_df.columns else pd.Series("", index=mean_w.index)
    sup = let.apply(to_superscript)

    final[t] = mean_col.map(lambda x: f"{x:.4e}") + " ± " + sem_col.map(lambda x: f"{x:.4e}") + sup

final_df = final.reset_index().rename(columns={"index": "Metabolite"})

final_df.to_csv("mean_sem_with_letters.csv", index=False)
print("Saved: mean_sem_with_letters.csv")
final_df.head()
```

Example input dataset (synthetic CSV)

This dataset is synthetic and provided solely for demonstration and reproducibility purposes.

```
Metabolite,Treatment,Replicate,Value
Glucose,A,1,1.23
Glucose,A,2,1.18
Glucose,A,3,1.27
Glucose,B,1,0.98
Glucose,B,2,1.01
Glucose,B,3,0.96
Citrate,A,1,0.56
Citrate,A,2,0.58
Citrate,A,3,0.54
Citrate,B,1,0.41
```

Safety warnings

Limitations (read before use)

1. Statistical inference depends on the chosen test. The provided workflow uses pairwise t-tests as an example. For factorial designs, repeated measures, non-normal data, or heteroskedasticity, alternative models/tests may be more appropriate (see Troubleshooting).
2. If testing many endpoints (e.g., dozens of metabolites), consider correcting multiple tests to control false positives.

Procedure

1 Create a dedicated Python environment (optional but recommended).

2 Install the packages: numpy, pandas, scipy.

3 Place the input CSV file (metabolomics_example_A-I_tidy.csv) in the same folder as the notebook, or set the correct path in the notebook.

4 Checkpoint: You can open the notebook and perform a cell import without errors.

5 Load the CSV file into a pandas DataFrame.

6 Verify that the required columns are present: Metabolite, Treatment, Replicate, Value.

7 Convert Value to numeric and delete rows with missing/invalid (NA/NaN) values.

8 Ensure that Treatment is stored as a string.

9 Define the treatment order (recommended) to control the order of columns in the final table (e.g., ["A", "B", "C", "D", "E", "F", "G", "H", "T"]).

10 Checkpoint: After cleaning, the data set contains only valid numeric values for analysis.

11 Group the data by Metabolite × Treatment.

12 Calculate:
Mean for each group
SEM for each group (SEM = SD / √n; where n is the number of replicates per group)

13 Checkpoint: You obtain two summary tables (mean and Group standard error) that can be reshaped into a larger format.

14 For each Metabolite, calculate all pairwise comparisons between treatments (e.g., A vs. B, A vs. C, etc.).

15 Use an independent two-sample t-test (e.g., scipy.stats.ttest_ind).

16 Store the results in a long-format table with at least:
Metabolite
Group 1
Group 2
p-value

17 Export the table to: ttest_all_pairs.csv.

18 Critical parameter: Set the alpha significance threshold alpha (default 0.05).

19 Recommended option: Apply a multiple testing correction (Bonferroni/Holm/FDR) between comparisons within each metabolite or across the entire results table, depending on the study design and reporting strategy.

20 Checkpoint: The ttest_all_pairs.csv file is created containing the p-values for all group pairs.

21 Load ttest_all_pairs.csv.

22 Ensure that p_value is numeric and that Group1/Group2 are strings.

23 Define:
the ordered list of treatments
the alpha threshold

24 Build a significance/non-significance matrix between groups for each metabolite.

25 Assign grouping letters according to the compact letter principle:

Groups that are not significantly different share at least one letter.
Groups that are significantly different do not share letters.

26 Create a "letter table" with one row per metabolite and one column per treatment.

27 Checkpoint: There is a letter table (wide format), and each metabolite is assigned letters for all treatments.

28 Reshape the mean and SEM summaries in broad format (rows = metabolite; columns = treatments).

29 For each metabolite × treatment cell:
format mean (scientific notation or decimal; choose one consistent style)
format SEM
append superscript letters (if present)

30 Export the final table to: mean_sem_with_letters.csv.

31 Checkpoint: mean_sem_with_letters.csv contains a complete summary table ready for pasting into a manuscript.

32