




Tidy-TS: A Type-Safe Framework for Statistical Data Analysis in the TypeScript Ecosystem

John Thomas Menchaca MD ^{1,2}

¹ University of Utah, Department of Biomedical Informatics ² University of Utah, Department of Internal Medicine

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

Tidy-TS is a type-safe library for statistical computing, data transformation, and visualization in the TypeScript ecosystem. It introduces a functional grammar for manipulating tabular data using arrays of objects, which are the idiomatic format in JavaScript runtimes. Inspired by the tidyverse philosophy ([Wickham et al., 2019](#)), it brings static typing, schema validation, and compile time guarantees to data workflows in TypeScript.

The library supports pipelines that load data from files, APIs, or databases, apply transformations that preserve type information, run statistical analyses, and produce interactive visualizations using Vega-Lite ([Satyanarayan et al., 2016, 2017](#)). These workflows remain entirely within TypeScript and avoid the need to switch languages for analysis or presentation.

Tidy-TS is designed for use in browsers, servers, and notebooks. It targets teams working fully in TypeScript who face fragmented toolchains and inconsistent data handling. Features such as asynchronous row-wise operations, concurrency controls, columnar storage, and WebAssembly acceleration support mid scale interactive analytics in both research and production environments. The addition of a type system extends this paradigm to include substantive compile-time guarantees through static types, which existing literature suggests can prevent 15–38% of production bugs ([Bunge, 2019](#); [Gao et al., 2017](#); [Khan et al., 2021](#)).

Statement of need

JavaScript and TypeScript are widely used for building interactive data applications, but they lack a type-aware and expressive toolkit for statistical computing. Existing libraries like Arquero and Danfo.js offer partial solutions for dataframe operations, but they do not provide a unified type-safe environment that combines data wrangling, statistical analysis, schema validation, asynchronous control, and visualization.

Without a shared grammar or consistent type model, developers are often left writing brittle glue code across loosely coupled tools. This leads to duplicated logic, reduced testability, and unnecessary language switching. Many teams using TypeScript for data ingestion and application logic still rely on Python or R for analysis and modeling. This workflow involves exporting and reimporting data with assumptions about column names, types, and missing values. Subtle errors such as incorrect joins, inconsistent handling of null and undefined values, or mismatched column references often go undetected until late stages. These issues create avoidable risk in data pipelines. Tidy-TS is designed to prevent this class of error through structured typing and declarative composition.

This library implements and builds upon tidyverse grammar using TypeScript's structural type system. Each operation in a pipeline maintains and refines the type signature of the data. For example, mutate not only adds a column but also updates the inferred structure so that

41 downstream transformations and visualizations remain type-safe. Operations like select, drop,
 42 join, pivot, and summarize all preserve static typing. Descriptive statistics follow conventions
 43 from R but enforce type correctness at compile time rather than through runtime checks.
 44 For example, functions that summarize over potentially null values require the developer to
 45 explicitly handle missing data before returning numeric output. This type-safe approach is
 46 illustrated in Example 1, which demonstrates how a complete data analysis workflow maintains
 47 type information throughout transformations, grouping, and aggregation.

48 **Example 1: Type-Safe Data Transformation Pipeline**

```
import { createDataFrame, stats as s } from "@tidy-ts/dataframe";

const sales = createDataFrame([
  { region: "North", product: "Widget", quantity: 10, price: 100 },
  { region: "North", product: "Gizmo", quantity: 20, price: 50 },
  { region: "South", product: "Widget", quantity: 20, price: 100 },
  { region: "East", product: "Widget", quantity: 8, price: 100 },
]);

const analysis = sales
  .mutate({
    revenue: (row) => row.quantity * row.price,
    moreQuantityThanAvg: (row, _index, df) => row.quantity > s.mean(df.quantity)
  })
  .groupBy("region")
  .summarize({
    total_revenue: (group) => s.sum(group.revenue),
    avg_quantity: (group) => s.mean(group.quantity)
  })
  .arrange("total_revenue", "desc");
```

49 Tidy-TS also includes support for statistical hypothesis testing (Example 2). These functions
 50 are validated against R using randomized test suites to ensure parity in results. Output
 51 from statistical tests is returned in typed objects that contain test names, effect sizes, p
 52 values, confidence intervals, and relevant statistics in a structured format. This improves
 53 both interpretability and composability. These features are particularly important in clinical
 54 research and in applications that process real-time clinical data. Example 2 demonstrates how
 55 Tidy-TS both exposes statistical tests directly as well as provides an omnibus compare API
 56 that guides users to the correct test based on their intention, invoking the tidyverse philosophy
 57 of human-centered design. Both methods return the same structured, typed results.

58 **Example 2: Statistical Hypothesis Testing with Type Safety**

```
import { stats as s } from "@tidy-ts/dataframe";

// Create test "height" data for 6 individuals
const heights = [170, 165, 180, 175, 172, 168];

// Direct test API
// Access specific statistical tests
const directTest = s.test.t.oneSample({
  data: heights,
  mu: 170,
  alternative: "two-sided",
  alpha: 0.05
});
```

```
// Compare API
// Intent-driven design options to assist with picking
// the correct test.
const compareAPI = s.compare.oneGroup.centralTendency.toValue({
  data: heights,
  comparator: "not equal to" // or "less than" | "greater than"
  hypothesizedValue: 170,
  parametric: "parametric", // or "nonparametric" | "auto"
  alpha: 0.05
});

// Both return the same typed result:
// {
//   test_name: "One-sample t-test",
//   p_value: 0.47...,
//   effect_size: { value: 0.31..., name: "Cohen's D" },
//   test_statistic: { value: 0.76..., name: "T-Statistic" },
//   confidence_interval: { lower: 166.08..., upper: 177.24...,
//     confidence_level: 0.95 },
//   degrees_of_freedom: 5,
//   alpha: 0.05,
//   alternative: "Two-Sided"
// }
```

59 Many modern analytics workflows also rely on remote API calls or external services for validation
60 and analysis. Tidy-TS supports asynchronous transformations natively, permitting row-wise
61 async operations within mutate, filter, and summarize operations. It also includes both
62 dataframe-level and operation-level controls for concurrency and retries. This eases the task
63 of building analytics pipelines dependent on external services or subject to API rate limits.
64 Applications that call remote artificial intelligence (AI) models or services are a key use case.

65 Modern workflows also ingest data from various sources. Tidy-TS provides tools to import
66 CSV, Parquet, Arrow, and JSON in a type-safe manner with the help of Zod schema. Likewise,
67 databases queries made with raw SQL can be made type-safe with schema validation. For
68 data queried via popular type-safe Object-Relational Mappers, Tidy-TS can seamlessly create
69 dataframes using their provided types.

70 Tidy-TS builds on lessons from tools like pandas, dplyr, and Polars but is adapted for modern
71 development in TypeScript, providing a model for integrating asynchronous transformations,
72 structural typing, statistical functions, visualization, and multi runtime deployment within the
73 constraints and idioms of the TypeScript ecosystem.

74 Research applications

75 Tidy-TS supports common data analysis workflows in research environments that rely on
76 TypeScript. At the University of Utah's Department of Biomedical Informatics, the library
77 is used for healthcare data analysis where static typing and schema validation help improve
78 analytical accuracy and reduce runtime errors, supporting both ad hoc quality improvement
79 analysis and real-time evaluation of clinical data streams intended for integration into the
80 electronic health record.

81 The ability to perform asynchronous transformations also allows researchers to incorporate
82 external data sources without switching between programming languages. For example,
83 researchers can fetch data from a repository, process and clean it, invoke async AI tools for
84 analysis, perform statistical testing, and then visualize the results (see Figure 1) all within a
85 single type-safe TypeScript workflow without switching to Python or R for analysis.

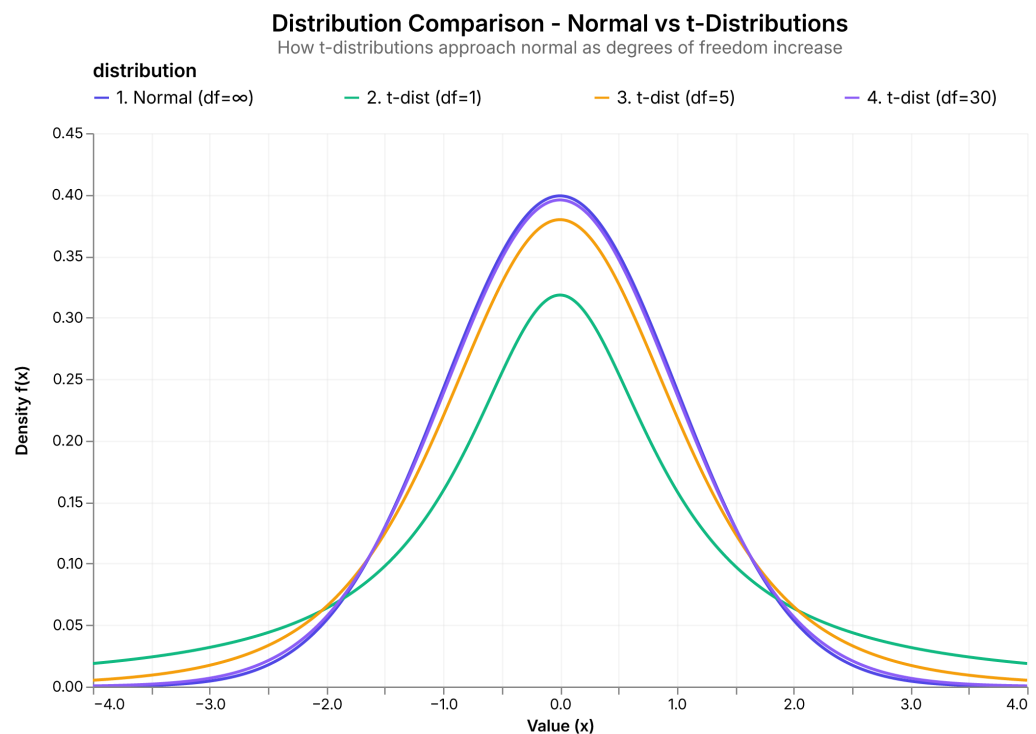


Figure 1: Publication-quality statistical visualization generated with Tidy-TS, showing normal and t-distributions with varying degrees of freedom.

Tidy-TS runs in multiple environments including Node.js, Deno, Bun, and modern browsers. It can also be used interactively in Jupyter notebooks through the Deno kernel, enabling data analyses and exploration alongside graph outputs with minimal configuration. This extends the reach of the type-safe analytics and reproducibility to an interactive workflow commonly used in data science and research.

Acknowledgements

No external funding was received in support of this work. The author is grateful to Dr. Kensaku Kawamoto for his mentorship and support, without which this work would not be possible.

Tidy-TS is also only possible due to numerous open source projects. Beyond the pioneering work by the tidyverse teams, the author would like to specifically acknowledge and give thanks for the excellent work by the University of Washington Interactive Data Lab for their inspirational work with Vega-Lite, Vega, and the arquero javascript library.

Bunge, B. (2019). *Adopting TypeScript at scale*. JSConf Hawaii 2019. <https://www.youtube.com/watch?v=P-J9Eg7hJwE&feature=youtu.be&t=702>

Gao, Z., Bird, C., & Barr, E. T. (2017). To type or not to type: Quantifying detectable bugs in JavaScript. *Proceedings of the 39th International Conference on Software Engineering*, 758–769.

Khan, S., Uddin, G., & Niazi, M. (2021). A systematic literature review on the empirical evidence for the effectiveness of type systems. *IEEE Transactions on Software Engineering*, 47(12), 2670–2691. <https://doi.org/10.1109/TSE.2019.2961751>

Satyanarayan, A., Moritz, D., Wongsuphasawat, K., & Heer, J. (2017). Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1),

- 108 341–350. <https://doi.org/10.1109/TVCG.2016.2599030>
- 109 Satyanarayan, A., Russell, R., Hoffswell, J., & Heer, J. (2016). Reactive vega: A streaming
110 dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visual-*
111 *ization and Computer Graphics*, 22(1), 659–668. [https://doi.org/10.1109/TVCG.2015.](https://doi.org/10.1109/TVCG.2015.2467091)
112 [2467091](https://doi.org/10.1109/TVCG.2015.2467091)
- 113 Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Golemund,
114 G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Lin Pedersen, T., Miller, E., Milton
115 Bache, S., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H.
116 (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686.
117 <https://doi.org/10.21105/joss.01686>

DRAFT