Computational Humanities Research

www.cambridge.org/chr

Undate: humanistic dates for computation

Because reality is frequently inaccurate

Rebecca Sutton Koeser[1], Julia Damerow[2], Robert Casties[3] and Cole Crawford[4]

[1]Center for Digital Humanities, Princeton University, Princeton, NJ, USA; [2]School of Complex Adaptive Systems, Arizona State University, Tempe, AZ, USA; [3]DH-Team, Max Planck Institute for the History of Science, Berlin, Germany and [4]Arts and Humanities Research Computing, Harvard University, Cambridge, MA, USA

Software Paper

Cite this article: Koeser Rebecca Sutton, Julia Damerow, Robert Casties and Cole Crawford. 2025. "Undate: humanistic dates for computation: Because reality is frequently inaccurate" *Computational Humanities Research*, **1**:e5, https://doi.org/10.1017/chr.2025.10006

Received: 15 April 2025
Revised: 27 June 2025
Accepted: 17 July 2025

Keywords:
calendars; dates; software development; temporal reasoning; uncertainty

Corresponding author:
Rebecca Sutton Koeser;
Email: rebecca.s.koeser@princeton.edu

Abstract

`undate` is an ambitious, in-progress effort to develop a pragmatic Python package for computation and analysis of temporal information in humanistic and cultural data, with a particular emphasis on uncertain, incomplete, and imprecise dates and with support for multiple calendars. The development of `undate` is grounded in domain-specific work on digital and computational humanities projects from multiple institutions, including *Shakespeare and Company Project*, *Princeton Geniza Project*, and *Islamic Scientific Manuscript Initiative*. With increasing support for different formats and calendars, `Undate` aims to bridge technical gaps across different communities and methodologies. In this article, we describe the `undate` software package and the functionality of the core `Undate` and `UndateInterval` classes to work with dates and date intervals. We discuss why this software exists, how it expands on and generalizes prior work, how it compares to other approaches and tools, and its current limitations. We describe the development methodology used to create the software, our plans for active and continuing development, and the potential `undate` has to impact computational humanities research.

Plain language summary

Dates are one of these things that seem very straightforward at first glance but prove to be complex when working with humanities and cultural data, especially historical content. Representing a "simple" time period such as the "16th century" is not that simple. Should the 16th century come before or after 1558 when sorting records? How can a computer determine whether the 16th century includes the year 1558? In other cases, historic dates may be recorded in a different calendar, such as the Hebrew *Anno Mundi* or Islamic *Hijiri* calendars, with years and months that don't match up exactly with the standard Gregorian calendar, and we may want to work with and compare dates across multiple different calendar systems. Existing software solutions for working with dates are usually not built to deal with these kinds of "fuzzy" dates. `undate` is a Python software package developed to make it easier to calculate with fuzzy and incomplete dates. In this article, we describe its functionality, conceptualization, how it is being developed, and potential impact.

Introduction

`undate` is an ambitious in-progress effort to develop a pragmatic Python package for the computation and analysis of temporal information in humanistic and cultural data, with a particular emphasis on uncertain, incomplete or imprecise dates and with support for multiple calendaring systems and date formats (Koeser et al. 2025). Humanities and cultural data frequently include temporal information, and this context is crucial for interpreting and making claims based on that data. However, historical dates are often imprecise or partially unknown, they may not use the Gregorian calendar, and even within the same project or dataset dates may be inconsistent in both format and precision. There are well-known, interoperable solutions for documenting dates, most notably the ISO 8601 standard and Extended Date-Time Format (EDTF), but these standards are textual rather than numerical, and serve as more of an "interchange format" that requires transformation for interpretation and computation (Schmidt 2018, 322, 324). With `undate`, we offer a solution for working with messy historical and cultural date information in a structured way, not simply as text for display or for information retrieval, but for reasoning, calculation, and comparison.

The `undate` library provides `Undate` and `UndateInterval` classes, which we hope will eventually be as ubiquitous and easy to use as Python's built-in `datetime.date`, but with support for varying degrees of precision, unknown information, different calendars, and parsers to convert between a variety of formats and calendars. `Undate` objects are built to hold

This article was awarded Open Materials badge for transparent practices. See the Data availability statement for details.

Figure 1. `undate` project logo.

potentially imprecise dates, such as cases, where only the year or the month is known, or only the month and day. `UndateInterval` objects build on top of `Undate` objects to represent date ranges with imprecise bounds.

The design of `undate` is based on experiences gained from working on several exemplary projects, including the *Shakespeare and Company Project*, *Princeton Geniza Project*(PGP), and the *Islamic Scientific Manuscripts Initiative* (ISMI). We take additional inspiration from Lauren Klein's challenge to be intentional and explicit about humanities theories and methodologies driving technical modeling and implementations (Klein 2024), rather than limiting ourselves to existing software solutions, which prioritize contemporary and commercial use cases.

Context

The development of `undate` is a collaborative effort that started during a week-long community hackathon organized by DHTech[1] in 2022. The goal of the hackathon was to develop a reusable software for working with uncertain and imprecise dates, which was a shared problem across projects. We came up with the name "undate" because of the goal of working with temporal information that are not precisely dates. The "un" prefix is a reminder of the *un*certain and *un*known aspects of this information; the obsolete meaning associated with wavy or undulating lines provides additional resonance (*Undated, adj.*[1]), since temporal information may not be as linear as we typically assume. The project logo (Figure 1) gestures at the contrasting needs for nuanced human information that is computationally tractable. A core group of participants continued development after the hackathon, with Koeser taking the lead as the main contributor.

`Undate` objects differ from Python `datetime` objects because they accommodate uncertain and partial temporal data. An `Undate` can be initialized by providing any of year, month, or day as either numerical values *or* by string values, which can be used to indicate partially or fully unknown values. For instance, an `Undate` object may be initialized with `month=3` and `day=1`, or with a `year="19XX"`, where `X` indicates unknown digits; in this case, a year sometime in the 1900s. Optionally, a label may be provided to name a date, perhaps to label December 25 as "Christmas Day" or January 29, 2025 as "Lunar New Year." The initial values passed into an `Undate` are stored and used to determine the precision of the date (currently supported values are year, month, and day). The library then determines the earliest and latest possible dates based on the specified initial values, which are used for comparisons with other dates and calculations such as duration. To provide compatibility with and interchangeability between multiple formats, `Undate` objects can be initialized by using a parser for a specific format such as ISO8601, EDTF, or a supported calendar. An `Undate` is not only calendar-aware but calendar-explicit: the calendar is always defined, with a default of Gregorian.[2] This means that two `Undate` objects with the same numeric day, month, and year values represent different dates if they use different calendars – but they can be compared and used together. When an `Undate` object is initialized, the original values and precision of the date in the original calendar are preserved, and calendar conversion is used to determine the earliest and latest possible dates in the Gregorian calendar for comparison across calendars.[3] This means that we can preserve the precision of the date in the original calendar, such as a month or a year, even though that date may not neatly map to a date or year in the Gregorian calendar, and may have a different number of days than a year or month in the Gregorian calendar (see Table 1).

An `Undate` object conceptually represents a single date, although the support for different types of precision and uncertain or partially known temporal information makes it technically an interval; as Allen notes, "we can always decompose times into subparts" (Allen 1983, 834). An `UndateInterval` is an explicit date range between two `Undate` objects to support longer and less precise ranges of time. Like an `Undate`, an `UndateInterval` supports an optional label, since sometimes it is important to attach a specific name to events such as "20th century" or "Chinese year of the dragon." `UndateInterval` also supports calculating duration and determining whether one interval overlaps with or contains another.

Related software

`undate` builds on work from existing digital humanities research projects, with the goal of generalizing custom, one-off solutions into a reusable library. `undate` exists in an ecosystem of Python libraries for working with dates and intervals, and is part of a longer history of various approaches developed by computational humanities researchers. Our work on `undate` is inspired by the ability of humanistic researchers to reason with partial information, to infer relationships and develop insights. Software tools and computational methods often throw out or ignore partial and uncertain information; we want to support inferential reasoning in tandem with computation.

Related projects

`undate` draws on a partial date implementation from the *Shakespeare and Company Project* and calendar conversion and mixed precision dates in the *PGP*. Calendar logic and representation is additionally informed by work on the *ISMI*. The first two projects are both database-backed web applications built in Python with the Django web framework, so the uncertain date logic prioritized data storage and information retrieval. In the *Shakespeare and Company Project*, "date precision is stored in flags alongside the dates, and are used to calculate and display the date accurately based on the known portion" (Kotin and Koeser 2022, 17). Storing partial

[1] DHTech is a community of people who work on technical aspects of Digital Humanities projects, and provides a network for exchange and collaboration. In 2021, DHTech became a Special Interest Group of the Alliance of Digital Humanities Organizations (ADHO). The group organizes virtual events such as meetups as well as in-person events at conferences.

[2] For convenience and consistency with available software libraries, we use the Gregorian calendar for all date comparisons and calculations. Technically, this is "proleptic" Gregorian for dates before 1852, when the Gregorian calendar was established.

[3] Currently `undate` supports the Islamic Hijri calendar and the Hebrew Anno Mundi, with work in progress to add support for Julian and Seleucid calendars.

https://doi.org/10.1017/chr.2025.10006 Published online by Cambridge University Press

Table 1. Dates in different calendars with varying precision

Date	Calendar	`Undate` initialization	Precision	Earliest	Latest	Duration
				Gregorian		days
26 Tammuz 4816	Hebrew	`Undate(4816, 4, 26, calendar="Hebrew")`	day	1056-07-17	1056-07-17	1
Tammuz 4816	Hebrew	`Undate(4816, 4, calendar="Hebrew")`	month	1056-06-22	1056-07-20	29
4816	Hebrew	`Undate(4816, calendar="Hebrew")`	year	1055-10-01	1056-09-17	353
7 Jumādā I 1243	Islamic	`Undate(1243, 5, 7, calendar="Islamic")`	day	1827-11-26	1827-11-26	1
Jumādā I 1243	Islamic	`Undate(1243, 5, calendar="Islamic")`	month	1827-11-20	1827-12-19	30
1243	Islamic	`Undate(1243, calendar="Islamic")`	year	1827-07-25	1828-07-13	355
2 June 1863	Gregorian	`Undate(1863, 6, 2, calendar="Gregorian")`	day	1863-06-02	1863-06-02	1
June 1863	Gregorian	`Undate(1863, 6, calendar="Gregorian")`	month	1863-06-01	1863-06-30	30
1863	Gregorian	`Undate(1863, calendar="Gregorian")`	year	1863-01-01	1863-12-31	365

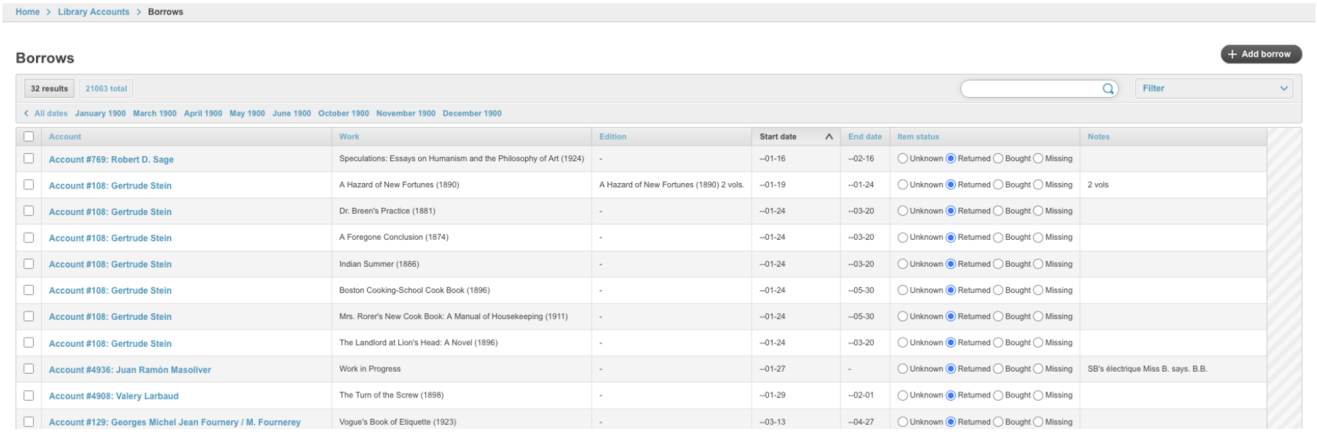

Figure 2. Screenshot from Shakespeare and Company Project admin interface for borrow events showing date filtering for events for unknown years. The built-in Django `date_hierarchy` filter displays unknown years as 1900.

dates as `datetime` objects in the database enables powerful date-specific functionality, such as hierarchical date filtering (Figure 2). However, tracking which portions of the date are known or certain in parallel leaves the potential for error, since the date can be accessed without consulting those flags, which would result in using unknown values as if they were fully known. In the *PGP*, dates are stored across multiple fields: a text field for the date in the original calendar, the calendar, and a converted Common Era (CE) date or date range for indexing and filtering in the Solr search engine and for display on the public site. We determined that neither of these approaches could be used directly for the more ambitious scope of `undate`, but adapted the strengths of both in the design of the `Undate` class: calendar conversion and imprecise dates with earliest and latest possible values, backed by `datetime` objects rather than numbers or strings. We prioritized functionality for parsing, comparison, and temporal calculations, and put off decisions on database serialization, since that may be more project-specific.

The *Shakespeare and Company Project* is based on the materials of Sylvia Beach and the English-language lending library she owned and operated in Paris in the 1920s and 1930s (Shakespeare and Company Project 2020). Despite the fact that this is a modern archive, data is drawn from handwritten cards jotted down by the clerks and Beach as they ran the shop, so details are often missing: "cards and the address books occasionally include incomplete dates – a year and a month, or just a year or a month, or no date at all" (Kotin and Koeser 2022, 10). Koeser led the work to develop a solution to "handle one-day events, like buying a book or closing out an account, as well as longer-duration activities, like a membership or borrowing a book," which the team called "partially known dates" (Koeser 2019). This solution supports filtering and sorting records based on available dating information, as well as duration calculations for book borrowing, even when the year is unknown. This means that partial dates can be included in analysis and interpretation of member borrowing activity, such as books that were borrowed and returned quickly or checked out for longer times: "did members fail to finish them or devour them quickly?" (Kotin and Koeser 2020). They can also be included in broader analysis of borrowing behavior across all library members or specific individuals (Koeser and LeBlanc 2024, 17–18).

The *PGP* is a database of documentary geniza fragments from the Cairo Geniza (Princeton Geniza Project 2022). PGP includes "material dating from the ninth to the nineteenth centuries," although "unevenly distributed, the majority of the documents date to the eleventh, twelfth and thirteenth centuries, with significant later clusters from the sixteenth and nineteenth centuries" (Koeser and Rustow, n.d.). Marina Rustow has enumerated the challenges of the different kinds of temporal uncertainty raised by these

https://doi.org/10.1017/chr.2025.10006 Published online by Cambridge University Press

materials, which includes: multiple calendars (Jewish, Islamic, Coptic, Seleucid, Julian, Gregorian); explicit uncertainty due to gaps or deterioration of the materials; implicit uncertainty of dates inferred from evidence such as handwriting or named persons, places, or even coins; and the potential unreliability of both the original authors and dates inferred by scholars (Rustow 2020). The PGP codebase supports parsing and converting dates from Jewish, Islamic, and Seleucid calendars to CE for searching, filtering, and sorting, and the web interface and datasets include dates in both original and CE calendars.[4] PGP also supports dates with varying precision, similar to the implementation in the Shakespeare and Company Project.

The *ISMI* aims to "make accessible information on all Islamic manuscripts in the exact sciences" and contains dates from the 9th to the 19th century CE in Julian, Islamic, and Gregorian calendars (Ragep and Ragep 2008). The first version of ISMI used a custom Java implementation of complex dates that allowed the user to select a calendar (Julian, Hijri/Islamic, Gregorian), select a date in that calendar, and then specify the accuracy (day, year, or a custom range). Similar to the PGP implementation, dates are converted to a standard Gregorian calendar for processing but displayed in the original calendar. Preserving the original date is important for historical accuracy, since conversion from historical calendars is not completely precise.[5] The current implementation for ISMI uses the RDF-based ResearchSpace framework with a data model based on the CIDOC-CRM Time-Span class, which preserves the original calendar presentation and accuracy while using ISO8601 Gregorian dates for calculations. Building on the work of two distinct projects with Hijri dates resulted in more robust calendar support, and testing `undate` against data from both projects gives greater confidence in our implementation.

Other approaches

There have been many attempts to handle temporal uncertainty in computational humanities research. One approach is based on specifying uncertainty for single dates, turning them effectively into intervals, formalizing existing practices like using "193X" for "sometime in the 1930s," for example, in the EDTF specification (Extended Date Time Format (EDTF) Specification 2019). Another approach is based on temporal intervals with relations, as exemplified by the temporal algebra by James Allen, which makes it possible to model and reason about relative date references even without specific, known dates (Allen 1983). Relative dating is particularly important for fields such as archaeology, which is often dependent on relative and location-specific periods; the software tool ChronoLog[6] provides an implementation for modeling and calculating relative dates based on networks of relative dates and periods (Levy et al. 2021). Relative dating is also useful for material with implicit sequences, such as historical letter editions where a temporal ordering can be established without exact dates, or photos in a film roll with "implicit chronological ordering" of other "media like film and television" (Schmidt 2018, 340). In the *Humanities Data in R*chapter on temporal data, Arnold and Tilton recommend "separate columns for each numeric component of the date or time" to make it "easier to avoid errors and to record partial information" (Arnold and Tilton 2024, 17). This is a practical solution for tracking mixed precision information, but makes temporal analysis and computation more challenging.

One well-known implementation that leverages temporal intervals for uncertain dates is Topotime, developed by Elijah Meeks and Karl Grossner in 2014 (Grossner 2014). They developed a powerful data model that made it possible to define periods like "Bronze Age Britain" or "The Life of George Dance The Younger" using complex time-span objects with differing certainty and temporal and spatial relations. As Ben Schmidt explains, "Topotime's basic unit is not a line . . . but a *polygon*; time flows from left to right, but also varies up and down in probability" (Schmidt 2018, 344). However, this resulted in a complex data model that was never widely implemented. Grossner, later reused elements in the spatio-temporal Linked Places Format (Grossner, Janowicz, and Keßler 2016). This is still in use by the World Historical Gazetteer (Grossner and Mostern 2021), which focuses on referenceable places with historical context. This is a contrast to the PeriodO "gazetteer of periods," which foregrounds geographical context (Shaw, Rabinowitz, and Golden 2018). The web-based humanities research platform Nodegoat provides a similarly complex "chronology statement" mechanism, which allows to create temporal relations between items in the database using a complicated user interface (Storing Chronology Statements).

The CIDOC-CRM ontology accommodates both approaches separately: it has a `Period` class that enable relative temporal reasoning, as well as a `Time-Span` class, for documenting a single date with uncertainty. The property `at-some-time-within` can be used to indicate "that the phenomenon must have occurred within the limits of a particular time-span without further specifying as to when precisely" (CIDOC CRM 2024, 41). In cases, where the uncertainty does not exactly map to a Gregorian year or day, properties such as `begin-of-the-begin` can be used to describe "the latest point in time the user is sure that the respective temporal phenomenon is indeed not yet happening," with a corresponding `end-of-the-end`.[7] These are similar to the `notBefore` and `notAfter` attributes in the TEI XML specification, which can be used to specify the earliest and latest possible dates for an event in a standard form (TEI class att.datable.w3c 2025).

Kaše et al. take a different approach, testing probabilistic methods for analysis of uncertain temporal intervals as an alternative to avoid problematic solutions such as using the midpoint of a time period, which results in "overestimating the number of objects dated to the middle of centuries" or omitting records with large time periods, which results in "a substantial information loss, as broadly dated records get completely omitted" (Kaše, Sobotková, and Heřmánková 2023). While potentially useful for hypothesis testing, in spite of providing a Python package, their approach is not yet easily reusable and the code does not seem to be maintained (Kaše 2023).

It is difficult to balance support for nuance, complexity, and ambiguity while still providing a solution that is practical and easy for researchers and software developers to use. A pragmatic solution must focus on a set of features to successfully satisfy a chosen set of use cases. For `undate`, we chose to represent single

[4]The actual date conversion is done with the `convertdate` library (https://github.com/fitnr/convertdate), which requires numeric inputs for months and days; the parsing provides the mapping between named and numeric months.

[5]Conversion from the Islamic Hijri calendar to Gregorian has an ambiguity of one to two days. Refer to Gautschy for a short explanation of the problem of observable new moon and very thorough astronomical calculations (2018).

[6]https://chrono.ulb.be/

[7]See appendix "Guidelines for using P81a, P81b, P82a, P82b" in Doerr, Light, and Hibel (2020).

https://doi.org/10.1017/chr.2025.10006 Published online by Cambridge University Press

Table 2. Minimum and maximum years supported by different tools

Implementation	Minimum year	Maximum year
`datetime`	1	9999
`pandas.Timestamp`	1677	2262
`numpy.datetime64[D]`	2.5e16 BC	2.5e16 AD

dates with uncertainty along with support for multiple calendar systems.

Comparison to other Python libraries

`Undate` differs from the built-in Python `datetime.date` object, which requires year, month, and day all be specified, and only supports years between 1 and 9999 (*datetime*). In contrast, `Undate` supports any of year, year-month, year-month-day, month-day, supports unknown digits, and tracks the precision of the date (year, month, or day). Internally, `undate` uses `numpy.datetime64`, which has an extensive range of years depending on the precision used. We implemented shims to wrap `numpy.datetime64` and `numpy.timedelta64` to make them easier to work with and more consistent with the `datetime.date` interface, which were used in earlier versions of `undate`. Even though the popular data analysis library Pandas uses `numpy.datetime64` internally, and in spite of support for converting dates relative to a "Julian" origin, Pandas methods for parsing dates and converting to `Timestamp` objects don't support dates before 1677AD (Neumayer 2021). Refer to Table 2 for a comparison of minimum and maximum supported years for these three implementations.

Software tools like these are primarily designed for contemporary data and commercial applications, which is understandable given their origins in scientific contexts. This is evident in the fact that `numpy.datetime64` object has extensive business day logic but does not provide a weekday method.[8] These are unnecessary and inappropriate limitations for humanistic research, which spans beyond the typically supported years in both directions. It is more familiar to think of historical dates, but humanities data may also extend into the future. For example, "The Time Horizons of Futuristic Fiction" dataset "collects 2,564 English-language narrative works set in the future each marked with the year it was released and the year it takes place." The dataset was collected in order to "systematically measure the depiction of the future in fiction," with fictive futures that "range from 1840 CE to 100 trillion CE" (Wythoff and Leane 2025).

There are other Python packages with similar or overlapping functionality with `undate`. `DateTimeRange`[9] provides similar functionality to `UndateInterval`, but for full-precision `datetime` objects. The `dateparser`[10] library bills itself as a "Python parser for human readable dates," with support for absolute and relative dates, and even two non-Gregorian calendars (Persian Jalali and Hijri/Islamic); its stated use cases are web scraping, Internet-of-Things, log files, and format conversion.

Similarly, there are other libraries with support for parsing and serializing dates in standardized formats. The EDTF parsing in `undate` was implemented with reference to the specification as well as other implementations, primarily python-edtf[11] and EDTF.js.[12] The existing python library was too heavy to use as direct dependency, and the two projects have different scope and goals; `python-edtf` also includes natural language date parsing as well as optional Django database fields, but we do not intend to support the full EDTF specification and use a different internal data model.[13] Likewise, existing grammars for parsing could not be converted directly into the needed format.[14]

Our comparison with related software here has been primarily focused on the Python ecosystem, since that is the context for the `undate` package and what we are most conversant with. However, Python is not the only programming language with significant limitations around dates and temporal logic. As just one example, JavaScript has long-standing problems with the `Date` object that were inherited in part from the Java implementation (Pint 2017). Support for a new `Temporal` object was announced earlier this year (Smith 2025), which includes relevant functionality for partially known dates such as `Temporal.PlainYearMonth` and `Temporal.PlainMonthDay`, but which will likely remain an experimental technology for some time (Temporal—JavaScript 2025). As Ben Schmidt says, it is challenging to model time and to determine the "best possible model" to represent "time in formats that can be easily understood and easily transformed" (Schmidt 2018, 326).

Development methodology

Development on `undate` began at a hackathon organized by DHTech in November 2022. This was the first hackathon organized by DHTech, and there were no clear expectations for either the experience or specific outcomes. It was planned as a week-long, virtual event. We chose the problem space of fuzzy dates because it became clear during the planning that this was something multiple people were interested in and it had the potential to impact many different projects.

The hackathon began with a kickoff meeting. People interested in participating met to talk through ideas for the week, identify tasks, and establish processes. The initial meeting was a group of about eight participants; most attendees were experienced software developers, but there were also a few with less experience. We outlined the functionality we wanted to develop, drafted and assigned tasks, and identified dependencies among tasks. Participants met in smaller groups and pairs over the course of the week for programming sessions, and by the end of the week we had an alpha version of the software. Not everyone who attended the initial meeting ended up contributing; this is typical, but the work could also have been more inclusive with additional planning beforehand (Koeser 2023). The hackathon concluded with a wrap-up meeting to discuss the results and follow-up plans. Afterwards, a small group of participants decided to continue working on the project.

[8] https://numpy.org/doc/stable/reference/routines.datetime.htmlxxxhashxxxbusiness-day-functions

[9] https://github.com/thombashi/DateTimeRange

[10] https://github.com/scrapinghub/dateparser

[11] https://github.com/ixc/python-edtf

[12] https://github.com/inukshuk/edtf.js

[13] In addition to the differences in goals, there were concerns about support for `python-edtf`. When development began on `undate` in late 2022, more than four years had elapsed since the last release, and it was incompatible with current versions of Python and the Django web framework. The package was updated in 2024 to support the full EDTF specification finalized by the Library of Congress in 2019 (Crawford 2024).

[14] `undate` parsers are implemented with Lark https://github.com/lark-parser/lark.

https://doi.org/10.1017/chr.2025.10006 Published online by Cambridge University Press

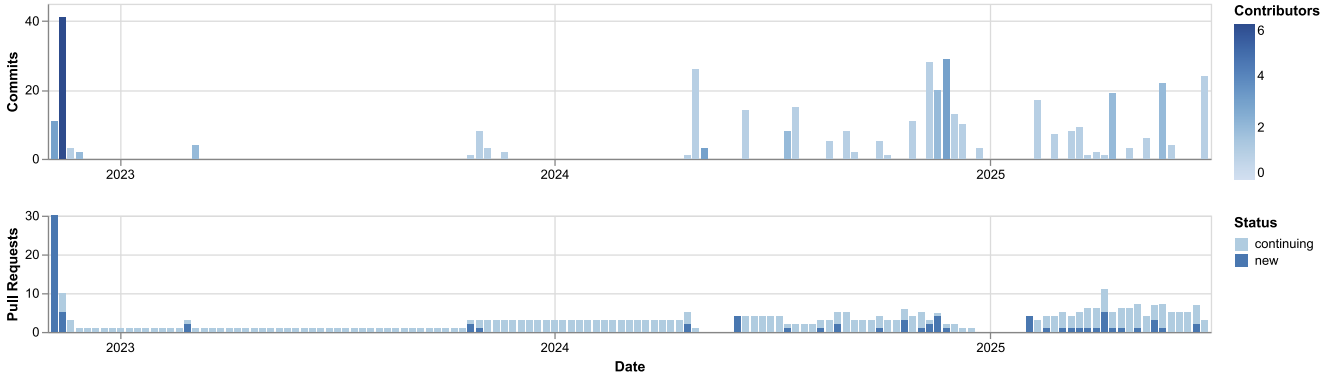

Figure 3. Git commits and open pull requests on the undate-python GitHub repository from late 2022 to mid 2025.
Note: Data and Jupyter notebook for this figure are available on CoCalc https://www.cambridge.org/S2977815825100067/CHR-Notebooks/files/Figure-3.

Prior to the hackathon, we determined the code would be hosted in a repository within the DHTech GitHub organization. At the kick-off meeting, participants made decisions about development processes. These included using the git-flow feature branch workflow: every new feature or bug fix is developed in an independent branch, and then merged into the develop branch once the feature is completed. New releases of the software are created by merging a set of changes in the develop branch into the main branch, and then tagging the main branch. We use GitHub pull requests for automated checks and code review of feature branches before merging new functionality, since we define completing a feature to include not only writing the code, but also writing unit tests and documentation. We initially required at least one review before merging a pull request. However, during the continued development, a problem quickly emerged: with only one active developer (Koeser) and two to three occasional contributors, pull requests tend to go unreviewed (see Figure 3). This is not uncommon for small open-source projects that are maintained by a handful of developers, especially for a niche project like `undate`. We, therefore, agreed a pull request could be merged if it went unreviewed over two weeks or a month, depending on development rhythms, so that progress would not stall.

Towards the end of the hackathon, we began using All Contributors[15] to recognize and celebrate the work completed, since we firmly believe that all contributions to a software package are valuable. Writing code is often privileged, but many contributions are crucial: thinking, planning, documenting, testing, reviewing, and providing sample data and examples, just to name a few. All Contributors provides GitHub integration and a command line tool for adding users and contributions. The project recognizes that an open source project does not only rely on coding but many kinds of work and makes it easy to recognize different contributions. For `undate`, providing sample data and notebooks that demonstrate specific examples and use cases in context are incredibly valuable. Contributors are listed in a `CONTRIBUTORS.md` file in the `undate` repository, which is linked from the main README. This public recognition has not yet resulted in new contributors, but we think the visibility is important and we hope the list and types of contributions will grow as more people learn about and use `undate`.

Development since the hackathon has been managed by conversations in an `#undate-dev` project channel within the DHTech Slack. In May of 2024, we started holding contributor meetings with roughly monthly frequency. Contributor meetings are used to check in on completed work, prioritize upcoming work, discuss possible approaches, and make decisions. They may also serve as a prompt for signing off on open pull requests. On occasion, contributor meetings are also used as work meetings, which can be an effective way of moving the project forward.

Tooling

To ensure code quality and maintainability, we use unit tests and continuous integration. Unit tests and code coverage checks run for every code change and pull request using GitHub Actions. We test against a matrix build of supported Python versions (currently 3.10 through 3.13). We use Treon[16] with a GitHub Actions workflow to check that example notebooks run successfully, which helps us keep the examples operable with changes to the core library. We use Sphinx[17] for code documentation, which we publish with Read the Docs[18]; we have a GitHub Actions workflow to check documentation compilation and coverage. We use pre-commit hooks to enforce formatting and other checks before committing changes to git, notably Ruff[19] for code style and mypy[20] for type checking. As a build tool, we use Hatchling[21] and for dependency management we use `pyproject.toml`, as recommended by the Scientific Python Library Development Guide (*Packaging*). We use GitHub issues for task tracking, GitHub milestones for planning releases, and GitHub discussions for development meeting notes.

Code review

In addition to automated checks, we rely on code reviews to ensure quality and as a means to discuss and agree on implementation decisions. Whenever possible and as contributor time permits, we aim for at least one review before changes are merged into the development branch. In the fall of 2024, we started to experiment with automatic AI code reviews using CodeRabbit[22] and Kypso[23] in order to gain familiarity with the tooling in this space. This has been a mediocre success. Initially, we found both tools to be quite

[15] https://allcontributors.org/

[16] https://github.com/ReviewNB/treon
[17] https://www.sphinx-doc.org/
[18] https://readsthedocs.com/
[19] https://docs.astral.sh/ruff/
[20] https://www.mypy-lang.org/
[21] https://pypi.org/project/hatchling/
[22] https://www.coderabbit.ai/
[23] https://kypso.io/

https://doi.org/10.1017/chr.2025.10006 Published online by Cambridge University Press

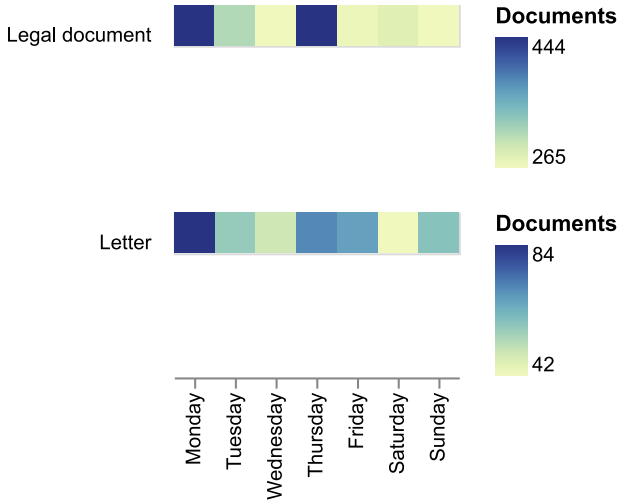

Figure 4. Heatmap of weekday frequency for PGP 2,285 legal documents and 427 letters dated with day-level precision. Saturday is the Hebrew Shabbat; Monday and Thursday are the other traditional convening days for court sessions.
Note: Data and Jupyter notebook for this figure are available on CoCalc https://www.cambridge.org/S2977815825100067/CHR-Notebooks/files/Figure-4.

brittle and limited, applying static checks from tools like Ruff in a less efficient way. Since then, we have found CodeRabbit useful for identifying typos and inconsistencies, and even flagging potential logic errors; but it also frequently flags details that are irrelevant or out of scope for a particular feature. We are undecided whether these AI code review tools are better than nothing, although we continue to use them.

Audience

`Undate` is a Python library, and as such it is primarily intended for developers, data scientists, and researchers writing code or analyzing data in Python who want to work with dates in various formats and calendars. We are particularly interested to see `undate` used by digital and computational humanities researchers and developers, but we fully expect it to be useful in other domains. We want to empower people to do more than the basics of storing dates for information retrieval and display, which is often a challenge with humanities datasets, so that intellectual capacity can be spent on more meaningful temporal analysis.

`Undate` is already useful for data analysis on humanities and cultural data, and can be used with libraries like Pandas (see Figure 4) for an example of weekday analysis of documents from *PGP* datasets (Rustow et al. 2025) and Figure 5 for an example of borrowing durations with unknown years from *Shakespeare and Company Project* datasets (Koeser and Kotin 2025). It may be useful for parsing and validating cultural heritage data, such as publication dates in library catalog data. `undate` can be used for sorting, filtering, and comparing disparate dates across calendars and precisions, and also supports checking if two date intervals overlap, or if one date or date interval falls within another.

We have been in conversation with other researchers who have dealt with similar problems. These include Erin McCarthy (Systems of Transmitting Early Modern Manuscript Verse, 1475-1700[24]); Taylor Arnold (Digital Documerica[25], Photogrammar[26]); Péter Király (research on cultural heritage metadata quality). These projects and many others have developed custom solutions for parsing and storing imprecise, intractable dates; but since the temporal data work is not their primary research goal, it is hard for others to learn from or reuse their implementations.

Analysis of "The Time-Horizons of Science Fiction" would also benefit; this dataset includes fictional works that span "a wide range of years (e.g., time travel or multi-generational narratives)." For the published dataset, the authors "entered the median of the years depicted in the work," which they note they had to format "as an integer rather than a date because the Python datetime module doesn't support years beyond 9999" (Wythoff and Leane 2025). More sophisticated modeling of the temporal setting for these works would require date intervals as well as sets of dates, and analysis based on the full range of dates could offer a richer and more complex perspective on the depictions of futurity.

`Undate` does have a number of limitations; some of those are due to design choices, while others are because it is still in development. One deliberate limitation is that we do not support time, but only dates. We made this choice for pragmatic reasons; historical dates are already uncertain enough, and specific times are rarely known. For similar reasons, we are selective in our support of the EDTF specification.[27] `undate` does not yet support database integration; future versions may add support for the Django web framework based on the partial date implementation in the *Shakespeare and Company Project* codebase, which includes custom queryset filters and template tags. Available format and calendar converters are not exhaustive; the calendars `undate` currently supports based on existing projects with known use cases and expert researchers who could inform our work. We are interested in integrating the French Republican Calendar; the decimalization of time and 10-day cycles would provide a helpful challenge to assumptions about seven-day weeks. Parsing also currently requires specifying the format or calendar, but we have begun experimenting with an omnibus parser which combines unambiguous formats from all supported parsers. Although calendar and format support is incomplete, the structure for converters is designed and documented to be extensible. We plan to continue expanding support, while keeping the development grounded in domain-specific research and expertise.

We have begun exploring implementations for ambiguous durations, such as the number of days in the month of February in an unknown year. The EDTF specification includes support for indicating that dates are uncertain, approximate, or both; we have not yet grappled with these distinctions, since we believe those decisions should be informed by use cases and examples. As Ben Schmidt notes, "the distinction between ambiguity and uncertainty" is a significant challenge for temporal modeling, since temporal uncertainty may be subjective and context-specific enough to resist standardization and resist translation across different datasets or contexts (Schmidt 2018, 344, 346). Another interesting challenge is invalid dates due to human error, such as February 29th in a non-leap year, which could be supported and treated as an approximate date.

[24] https://stemma.universityofgalway.ie/
[25] https://digitaldocumerica.org/
[26] https://photogrammar.org

[27] `undate` only handles EDTF dates without times. We have not implemented seasons, since they require geographical context to be meaningful and winter of a particular year is ambiguous, since winter in the northern hemisphere spans years. Other advanced Level 2 EDTF components such as exponential years, significant digits, set representation, and group qualification are also not currently supported, though it seems like few projects realistically need these features.

https://doi.org/10.1017/chr.2025.10006 Published online by Cambridge University Press

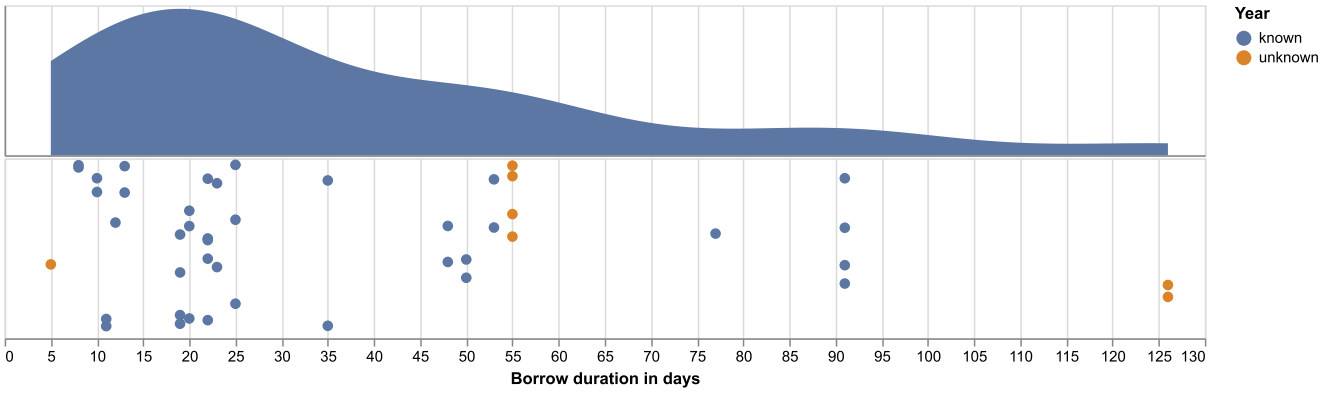

Figure 5. Raincloud plot showing how long Gertrude Stein kept the books she borrowed from the Shakespeare and Company lending library, based on *Shakespeare and Company Project* data (Koeser and Kotin 2025). Stein has 46 borrow events with calculable duration; of those 7 (15%) have no known year. Borrow events with unknown years are highlighted in orange in the lower portion of the plot.
Note: Data and Jupyter notebook for this figure are available on CoCalc https://www.cambridge.org/S2977815825100067/CHR-Notebooks/files/Figure-5.

`undate` does not yet support parsing "human-readable dates." Typical solutions to this problem are rule-based and brittle; despite long-standing interest in this problem, there are no standards, and the solutions we are aware of are unfortunately not multilingual. English-only solutions include the `cultural_dates` ruby gem[28] and `arttracks-js` JavaScript library[29], which came out of the Art Tracks project for transforming museum provenance records into searchable data[30], and `python-edtf`. We have begun experimenting with language models for parsing human-readable dates into structured format; early tests indicate it has potential, and could support the design of a solution that takes context into account.[31] Enhancements like these will be released as optional dependencies or separate packages, so that the core `undate` library remains as lightweight as possible.

Another interesting challenge is the "vexing . . . problem of the year zero" since "depending on convention, the year 0CE either does not exist at all or is entirely coincident with the year 1CE," which is a discontinuity that may cause problems and confusion (Schmidt 2018, 336). We feel strongly that this is best tackled in the context of a project or dataset that spans these years, so that the technical implementation can be worked out and evaluated with domain experts and real use cases.

Impact

`undate` is still a new library, but we are already finding it incredibly useful for data analysis and continually discovering more use cases and datasets, where `undate` is relevant. Because of the extensible format and calendar conversion capabilities, `undate` has the potential to serve as a bridge across different domains and modes, from computational and digital humanities and beyond, from databases to data science, from linked data to spreadsheets, and will eventually make it possible to do temporal analysis across different datasets, calendars, and domains. For now, `undate` is Python-only, but if successful we hope to expand to other languages, such as R or JavaScript in collaboration with partners who have appropriate domain and technical expertise.

We hope `undate` will handle the basics of working with partial, uncertain temporal information so that researchers can spend less effort wrangling temporal data and more effort on their research questions. As the library matures, we hope to support computational humanities work, such as the probabilistic approaches to temporal uncertainty mentioned earlier. We are inspired by Lauren Klein's call to foreground humanities theories with the specific temporal example of "operationalizing . . . medium-specific theories" such as "Twitter-time" by means of dataset partitioning "around local maxima of tweet frequency, allowing irregular intervals" (Roytburg et al. 2024), and hope `undate` will enable more interventions like these.

Use and reuse of code has a direct impact on quality; the more people and projects that use a piece of software, the more robust and trustworthy that software will be. We hope that projects will begin to use `undate` and contribute to its development. New contributors are not just welcome but needed. Writing code is only one aspect; providing use cases and example data, helping with documentation, thinking through questions and contributing to decisions about approaches are all invaluable contributions. Up to this moment, `undate` contributors represent four different organizations. We firmly believe that the long-term success of digital and computational tools like this depend on collaboration across institutions, countries, and domains.

Supplementary material. Computational Notebook files are available as supplementary material at https://doi.org/10.1017/chr.2025.10006 and online at https://www.cambridge.org/S2977815825100067/CHR-Notebooks.

Acknowledgments. Thanks to Jeri Weiringa for helpful comments and suggestions on a previous draft of this essay; to Erin McCarthy, Taylor Arnold, and Péter Király for sharing their experiences with similar problems; to Marina Rustow and Rachel Richman, for feedback on data visualizations of PGP data; and to our reviewers for their invaluable advice on restructuring the essay for improved clarity and for their suggestions on important additional functionality for `undate`.

Data availability statement. The software repository for the `undate` Python library, which includes example notebooks to demonstrate functionality, is available in a public GitHub repository: https://github.com/dh-tech/undate-python/

Author contributions. Conceptualization: R.S.K., J.D., R.C., C.C.; Data visualization: R.S.K.; Reviewing and editing: R.S.K., J.D., R.C., C.C.; Software: R.S.K., J.D., R.C., C.C.; Writing original draft: R.S.K.

[28] https://github.com/arttracks/cultural_dates
[29] https://github.com/workergnome/arttracks-js
[30] https://www.museumprovenance.org
[31] For instance, a photo by Russell Lee in the United States Farm Security Administration and Office of War Information (FSA-OWI) collection provides only the text "1941 July." in the date field, but the title of the photograph includes the text the "fourth of July." (Lee 1941)

Funding statement. This work was not supported by grant funding. Koeser's research is supported by the Center for Digital Humanities, Princeton University.

Competing interests. The authors declare none.

Ethical standards. The research meets all ethical guidelines and adheres to legal requirements. No AI was used to write this article. AI code review systems were used for software development as an experiment, but all code changes were reviewed and tested.

References

Allen, James F. 1983. "Maintaining Knowledge about Temporal Intervals." *Communications of the ACM* 26, no. 11, 832–843. https://doi.org/10.1145/182.358434.

Arnold, Taylor, and Lauren Tilton. 2024. Humanities Data in R: Exploring Networks, Geospatial Data, Images, and Text. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-031-62566-4.

Crawford, Cole. 2024. "Updating the Open Source Python-Edtf Library to Support the EDTF 2019 Standard." https://interaction.net.au/articles/updating-the-open-source-python-edtf-library-to-support-the-edtf-2019-standard/.

Definition of the CIDOC Conceptual Reference Model, edited by Chryssoula Bekiari, George Bruseker, Erin Canning, Martin Doerr, Philippe Michon, Christian-Emil Ore, Stephen Stead, and Athanasios Velios. 2024. https://cidoc-crm.org/Version/version-7.3.

Doerr, Martin, Richard Light, and Gerald Hibel. 2020. "Implementing the CIDOC Conceptual Reference Model in RDF—CIDOC CRM." https://cidoc-crm.org/Resources/implementing-the-cidoc-conceptual-reference-model-in-rdf.

Gautschy, Rita. 2018. "Islamic Calendar." https://www.gautschy.ch/~rita/archast/mond/arabcal.html.

Grossner, Karl. 2014. "Topotime: A Data Model and D3 Layout for Historical Time." https://hestia.open.ac.uk/topotime-a-data-model-and-d3-layout-for-historical-time/.

Grossner, Karl, Krzysztof Janowicz, and Carsten Keβler. 2016. "Place, Period, and Setting for Linked Data Gazetteers." In *Placing Names: Enriching and Integrating Gazetteers*, edited by Merrick Lex Berman, Ruth Mostern, and Humphrey Southall, 80–96. Bloomington, IN: Indiana University Press. https://doi.org/10.2307/j.ctt2005zq7.11.

Grossner, Karl, and Ruth Mostern. 2021. "Linked Places in World Historical Gazetteer." In GeoHumanities '21: *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Geospatial Humanities*, edited by Ludovic Moncla, Carmen Brando, Katherine McDonough, 40–43. New York, NY: Association for Computing Machinery. https://doi.org/10.1145/3486187.3490203.

Kaše, Vojtěch. 2023. "Tempun." https://doi.org/10.5281/zenodo.8179346.

Kaše, Vojtěch, Adéla Sobotková, and Petra Heřmánková. 2023. "Modeling Temporal Uncertainty in Historical Datasets." In *Proceedings of the Computational Humanities Research Conference 2023*, vol. 3558, edited by **Artjoms Šeļa, Fotis Jannidis, and Iza Romanowska**, 413–25. CEUR Workshop Proceedings. Paris, France: CEUR. https://ceur-ws.org/Vol-3558/#paper5123.

Klein, Lauren. 2024. *When Theory Leads: Towards a Humanities-Forward Model of Computational Research*. Aarhus: Keynote, Computational Humanities Research 2024.

Koeser, Rebecca Sutton. 2019. "Coding with Unknowns." https://cdh.princeton.edu/blog/2019/12/05/coding-unknowns/.

Koeser, Rebecca Sutton. 2023. "Join Me for a DHTech Hackathon? Its an Un-Date!" https://dh-tech.github.io/blog/2023/02/09/hackathon-undate/.

Koeser, Rebecca Sutton, Cole Crawford, Julia Damerow, Malte Vogl, and Robert Casties. 2025. "Undate Python Library." Version 0.5.1, July. https://doi.org/10.5281/zenodo.11068867.

Koeser, Rebecca Sutton, and Joshua Kotin. 2025. *Shakespeare and Company Project Datasets*. Princeton: Princeton University. https://doi.org/10.34770/kf6c-b079.

Koeser, Rebecca Sutton, and Zoe LeBlanc. 2024. "Missing Data, Speculative Reading." *Journal of Cultural Analytics* 9, no. 2, 1–28. https://doi.org/10.22148/001c.116926.

Koeser, Rebecca Sutton, and Marina Rustow. n.d. "Princeton Geniza Project Datasets." Article under review.

Kotin, Joshua, and Rebecca Sutton Koeser. 2020. *Shakespeare and Company: Top Ten Lists*. Princeton: Center for Digital Humanities, Princeton University. https://shakespeareandco.princeton.edu/analysis/2020/11/shakespeare-and-company-top-ten-lists/.

Kotin, Joshua, and Rebecca Sutton Koeser. 2022. "Shakespeare and Company Project Data Sets." *Journal of Cultural Analytics* 7, no. 1, 1–35. https://doi.org/10.22148/001c.32551.

LAB1100. 2020. "Storing Chronology Statements." Last modified August 11, 2021. https://nodegoat.net/guide.s/43/storing-chronology-statements.

Lee, Russell. 1941. "Main Street of Vale, Oregon, on the Fourth of July. Vale is One of the Shopping Centers for the Farmers Who Live and Work on the Vale-Owyhee Irrigation Project." still image. Archive Location: Oregon–Malheur County–Vale, July. https://www.loc.gov/pictures/item/2017789888/.

Levy, Eythan, Gilles Geeraerts, Frédéric Pluquet, Eli Piasetzky, and Alexander Fantalkin. 2021. "Chronological Networks in Archaeology: A Formalised Scheme." *Journal of Archaeological Science* 127, 1–27. https://doi.org/10.1016/j.jas.2020.105225.

Mozilla Foundation. 2025. "Temporal." Last modified July 10, 2025. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Temporal.

Neumayer, Wolfgang. 2021. "BC Dates in Python—Part 1—Numpy/Pandas." https://wolololf.github.io/fast-blog/datascience/history/python/numpy/pandas/time/2021/12/23/bc_dates_in_python_part_1.html.

Oxford English Dictionary. 1921. "undated, adj.[1]" Last modified March 2024. https://www.oed.com/dictionary/undated_adj1.

Princeton Geniza Project. 2022. Princeton: Center for Digital Humanities, Princeton University. https://geniza.princeton.edu/.

Pint, Maggie. 2017. "Fixing JavaScript Date Getting Started." https://maggiepint.com/2017/04/09/fixing-javascript-date-getting-started/.

Python Software Foundation. n.d. "datetime — Basic date and time types." Accessed January 17, 2025. https://docs.python.org/3/library/datetime.html.

Ragep, Jamil F., and Sally P. Ragep. 2008. "The Islamic Scientific Manuscript Initiative (ISMI) Towards a Sociology of the Exact Sciences in Islam." In *A Shared Legacy: Islamic Science East and West (Homage to Professor J. M. Mill à s Vallicrosa)*, edited by **Emilia Calvo, Mercè Comes, Roser Puig, and Monica Rius**, 15–21. Barcelona: University of Barcelona. https://islamsci.mcgill.ca/ISMI_SharedLegacy.pdf.

Roytburg, Dani, Deborah Olorunisola, Sandeep Soni, and Lauren Klein. 2024. "Words and Action: Modeling Linguistic Leadership in #BlackLivesMatter Communities." https://doi.org/10.48550/arXiv.2412.02637.

Rustow, Marina. 2020. "Dating Problems? Ask the Princeton Geniza Project Team." https://cdh.princeton.edu/blog/2020/11/18/dating-problems-ask-princeton-geniza-project-team/.

Rustow, Marina, Rebecca Sutton Koeser, Rachel Richman, Ksenia Ryzhova, Amel Bensalim, and Abdellatif Mohamed. 2025. "Princeton Geniza Project Dataset." https://doi.org/10.5281/zenodo.15839055.

Schmidt, Benjamin. 2018. "Modeling Time." In *The Shape of Data in Digital Humanities*, edited by Julia Flanders and Fotis Jannidis, 150–166. London and New York: Routledge.

Scientific Python. 2024. "**Packaging**." Scientific Python Library Development Guide. Last modified December 12, 2024. https://learn.scientific-python.org/development/tutorials/packaging/.

Shaw, Ryan, Adam Rabinowitz, and Patrick Golden. 2018. "A Deep Gazetteer of Time Periods." Mexico City. https://dh2018.adho.org/en/a-deep-gazetteer-of-time-periods/.

Shakespeare and Company Project. 2020. Princeton: *Center for Digital Humanities*, Princeton University. https://shakespeareandco.princeton.edu/.

Smith, Brian. 2025. "JavaScript Temporal is Coming." https://developer.mozilla.org/en-US/blog/javascript-temporal-is-coming/.

TEI Consortium. 2025. "TEI class att.datable.w3c." Last modified January 24, 2025. https://www.tei-c.org/release/doc/tei-p5-doc/en/html/ref-att.datable.w3c.html.

The Library of Congress. 2019. "Extended Date Time Format (EDTF) Specification." Last modified February 2, 2022. https://www.loc.gov/standards/datetime/.

Wythoff, Grant, and Theodore Leane. 2025. "Time Horizons of Futuristic Fiction." Post45 Data Collective. https://doi.org/10.18737/552626.

https://doi.org/10.1017/chr.2025.10006 Published online by Cambridge University Press