Undate in Action

Rebecca Sutton Koeser[1]

[1] Center for Digital Humanities, Princeton University, Princeton, New Jersey, USA

Abstract

This paper provides an introduction to the `undate` python library with an emphasis on demonstrating supported functionality with some example use cases from specific projects. `undate` is designed for working with uncertain and partially known dates, and also includes support for dates in multiple calendars and with mixed precision. This paper provides an overview of the basic functionality with comparison to the Python built-in `datetime.date`, demonstrating support for missing and partially known values. Use cases and data from *Princeton Geniza Project* and *Shakespeare and Company Project* are used to further demonstrate the value and practical application of `undate` for parsing and comparing mixed precision dates in multiple calendars, and for calculating and plotting durations for events with known days but unknown years.

Keywords: dates, calendars, partial information, missing data, software, digital humanities, python

1 Introduction

`undate` is a python library for working with uncertain or partially known dates, with support for multiple calendars [5]. It 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" [6]. The software is freely available on GitHub[1] and published under an open source software license. This project was originally started at a DHTech Hackathon [4], and builds on experiences from other digital humanities projects. The ultimate goal is to build an active community of people that are using and maintaining the project, and developing it further.[2]

Figure 1: `undate` project logo.

For an in-depth understanding of the context, goals, humanities and software development methodologies that inform the project, and a comparison with related software, I encourage you to read the software paper "Undate: humanistic dates for computation" [6]. The logo (Figure 1)

Rebecca Sutton Koeser. "Undate in Action." In: *Digital Humanities Tech Symposium 2025*, ed. by Julia Damerow and Rebecca Sutton Koeser. Vol. 2. Anthology of Computers and the Humanities. 2025, 38–53. https://doi.org/10.63744/SFtXXpIE4ERh.

[1] `https://github.com/dh-tech/undate-python/`

[2] This paper was originally presented as a "tool presentation." An interactive version of the original is available online in the form of a Marimo notebook at https://rlskoeser.github.io/undate-in-action/

was designed as a reminder of the competing needs for nuanced human-readable information and computationally tractable data.

1.1 How is `Undate` like `datetime.date`?

Before describing `undate`'s functionality for handling uncertain dates, first, as a point of reference, I demonstrate how it compares with the standard Python `datetime.date` implementation.

Both an `undate.Undate` and a `datetime.date` can be initialized by specifying numeric values for year, month, and day. Both can be printed using the default serialization (ISO8601, or YYYY-MM-DD). `undate` is compatible with `datetime.date`, so we can also can compare these two objects. This means that `undate` can be easily used instead of or in conjunction with `datetime.date`.

```
import datetime

from undate import Undate

year, month, day = 2000, 11, 7

# these are equivalent and we initialize them the same way
dt_november7 = datetime.date(year, month, day)
november7 = Undate(year, month, day)

assert november7 == dt_november7
```

1.2 How is `Undate` *not* like `datetime.date`?

However, there are some key differences between the packages. An `Undate` can be initialized with partial information. `Undate` can also take an optional label, since sometimes it's useful to name a date. It can be initialized with year and month, just year, or even month and day with no year.

```
# November 2000
november = Undate(year, month, label="November 2000")
# Year 2000
year2k = Undate(year, label="Y2K")
# November 7 in an unknown year
november7_some_year = Undate(month=month, day=day,
    label="Some November 7")
november7.label = "November 7, 2000"

# sometimes names are important
easter1916 = Undate(1916, 4, 23, label="Easter 1916")
```

If you try to do that with `datetime.date`, you get a `TypeError` because all three fields are required.

```
try:
    datetime.date(year, month)
except TypeError as err:
    print(err)

"function missing required argument 'day' (pos 3)"
```

Each of these `Undate` objects can be printed out in a standard format; the `Undate` class also tracks how precisely a date is specified, and can calculate the duration. If we iterate over the example partially-known dates initialized in the code block above and print label, formatted date, precision, and duration in days, we get output like this (here and throughout, formatting has been added for readability):

```
for example_date in [
    november,
    year2k,
    november7_some_year,
    november7,
    easter1916,
]:
    print(f"""{example_date.label} {example_date}
- Date precision: {example_date.precision}
- Duration in days: {example_date.duration().days}""")
```

November 2000 `2000-11`

- Date precision: MONTH
- Duration in days: 30

Y2K `2000`

- Date precision: YEAR
- Duration in days: 366

Some November 7 `--11-07`

- Date precision: DAY
- Duration in days: 1

November 7, 2000 `2000-11-07`

- Date precision: DAY
- Duration in days: 1

Easter 1916 `1916-04-23`

- Date precision: DAY
- Duration in days: 1

We can also do some simple calculations, like checking whether one date falls within another date.

```
assert november in year2k    # 2000-11 in 2000
assert november7 in november  # 2000-11-07 in 2000-11
assert easter1916 not in year2k  # 1916-02-23 not in 2000
assert november7_some_year not in year2k  # --11-07 not in 2000
```

2 What else can `Undate` do?

2.1 Partially known values

In the last set of examples, the values used to initialize `Undate` instances were all integers and parts of the date were either known or unknown. But what if you know part of a date?

You can initialize `Undate` with strings and use `X` to indicate unknown values. (We chose `X` based on the Extended Date Time Format (EDTF) specification).

Here we initialize two dates with partially known information, where a year or month cannot be precisely pinned down but some information is known. As before, we can iterate over these dates and output label, formatted date, precision, and duration:

```
someyear_1900s = Undate("19XX", label="1900s")
late2022 = Undate(2022, "1X", label="late 2022")

for example_date in [someyear_1900s, late2022]:
    print(f"""{example_date.label} {example_date}
- Date precision: {example_date.precision}
- Duration in days: {max(example_date.duration().days)}
""")
```

1900s `19XX`

- Date precision: YEAR
- Duration in days: 366

late 2022 `2022-1X`

- Date precision: MONTH
- Duration in days: 31

If you to initialize a `datetime.date` object like this, you get another `TypeError`.

```
try:
    datetime.date("19XX", 1, 1)
except TypeError as err:
    print(err)

"'str' object cannot be interpreted as an integer"
```

2.2 Uncertain durations

If you were paying close attention, you may have noticed I used the `max()` function in the code block above where I output the duration for the partially known dates.

The most recent version of `undate` (version 0.5) includes experimental support for uncertain time deltas. This is just one aspect of support for working with dates and date ranges in terms of duration. The `undate` package also includes an `UndateInterval` class, with first-class support for date ranges between two uncertain dates, or an open interval starting or ending with an `Undate`. For reasons of space, I don't discuss the interval functionality in this paper.

```
print(f"""{someyear_1900s.label} {someyear_1900s}
- duration: {someyear_1900s.duration()}
- duration in days: {someyear_1900s.duration().days}

{late2022.label} {late2022}
- duration: {late2022.duration()}
- duration in days: {late2022.duration().days}
""")
```

1900s `19XX`

- duration: `UnDelta(days=[365,366])`
- duration in days: `UnInt(lower=365, upper=366)`

late 2022 `2022-1X`

- duration: `UnDelta(days=[30,31])`
- duration in days: `UnInt(lower=30, upper=31)`

Even without precise information, we can still do some useful comparisons. February of an unknown year is still shorter than October, November, or December.

```
some_february = Undate(month=2, label="February of some year")
# February of an unknown year is shorter than
# an uncertain month 1X (October, November, or December)
assert some_february.duration() < late2022.duration()

print(f"""{some_february.label} {some_february}
- duration: {some_february.duration()}
- duration in days: {some_february.duration().days}
""")
```

February of some year `--02`

- duration: `UnDelta(days=[28,29])`
- duration in days: `UnInt(lower=28, upper=29)`

3 Example use cases from specific projects

`undate` is informed by work from existing digital humanities projects, and has been developed with the goal of generalizing custom solutions into a reusable library to benefit the whole community. In particular, "`undate` draws on a partial date implementation from the *Shakespeare and Company Project* and calendar conversion and mixed precision dates in the *Princeton Geniza Project* (PGP). Calendar logic and representation is additionally informed by work on the *Islamic Scientific Manuscripts Initiative* (ISMI)" [6].

3.1 Princeton Geniza Project

The *Princeton Geniza Project* (PGP) is a long-running project focused on materials from a synagogue in Cairo; these contents are primarily medieval, and largely written in Hebrew script. Because they are older and fragmentary, many of them cannot be dated; those documents that do have dates use a variety of calendars — the Hebrew *Anno Mundi* and Seleucid calendars, as well as the Islamic *Hijri* calendar. The Hebrew calendar is a lunisolar calendar and the Islamic calendar is a lunar calendar, so they don't map neatly to the months and years of the Gregorian calendar.

The PGP is one of the precursors that fed into the development of `undate`, due to this need to support storing, filtering, and searching dates from multiple calendars, with mixed precision, and various kinds of temporal uncertainty [10]. Here, I use data from the published PGP datasets [11] to demonstrate `undate` calendar parsing and conversion functionality.

An `Undate` instance preserves any initial values for year, month, and day in the original calendar. As you see in Table 1, the ISO format `Undate` corresponds to the original date rather than the standardized date.

Original Date	Calendar	Standard date	`Undate`	Precision	Weekday
1570	Seleucid	1259	1570	year	
Tammuz 1288	Seleucid	0977-06-21/0977-07-19	1288-04	month	
19 Adar 1427	Seleucid	1116-03-05	1427-12-19	day	Sunday
4890	Anno Mundi	1129-09-16/1130-09-05	4890	year	
Av 5564	Anno Mundi	1804-07-09/1804-08-07	5564-05	month	
10 Nisan 4716	Anno Mundi	0956-03-24	4716-01-10	day	Monday
537	Hijrī	1142-07-27/1143-07-15	0537	year	
Shawwāl 425	Hijrī	1034-08-29/1034-09-07	0425-10	month	
5 Safar 584	Hijrī	1188-04-05	0584-02-05	day	Tuesday

Table 1: A sampling of dates from PGP documents in different calendars; dates have been parsed by `Undate`, which was then used to report precision and weekday.

Under the hood, `undate` calculates the earliest and latest possible dates in range for comparison and sorting. When an `Undate` object is initialized with a different calendar, the earliest and latest dates are based on conversion to the Gregorian calendar so that dates can be compared and used together across calendars. (The code used to parse the PGP document dates displayed in this example is available in Appendix A). When working across calendars, dates with the same precision (year, month) often have different durations in days.

An `Undate` object is aware of date precision, which means that in a dataset with mixed precision dates like PGP, once dates are parsed we can easily see the variation in the data (Table 2).[3]

Preserving available details in mixed precision dates offers new opportunities for analysis; for instance, we can analyze weekday frequency by document type (Figure 2). In the PGP data, Legal documents and letters are the types of documents most likely to have a clear date. When visualized with a heatmap, the results match our expectations: Saturday is light for both types, because it is the Hebrew Shabbat, a day of rest. Monday and Thursday are the traditional convening days for court sessions, which is reflected in the preponderance of Legal documents on those days.[4]

[3] With the caveat that in this case the results are somewhat skewed by the dates that can easily be parsed; PGP data includes modifiers not yet supported by `undate`, which were ignored during parsing.

[4] Analysis and visualization adapted from prior work. [6]

Original Date precision	Documents
year	831
month	1,021
day	1,558

Table 2: Totals for PGP documents and date precision

Only includes documents with standard dates that can be parsed by `undate`.

Original Date	Calendar	`Undate`	Earliest	Latest	Precision	Duration
1570	Seleucid	1570	1258-09-07	1259-09-26	year	383
1474	Seleucid	1474	1162-09-18	1163-09-06	year	354
Tevet 1363	Seleucid	1363-10	1051-12-14	1052-01-11	month	29
[12]91	Seleucid	1291	0979-09-30	0980-09-17	year	353
Heshvan 1453	Seleucid	1453-08	1141-10-11	1141-11-08	month	29
1347	Seleucid	1347	1035-09-12	1036-09-28	year	383
Shevat (4)791	Anno Mundi	4791-11	1031-01-03	1031-02-01	month	30
4812	Anno Mundi	4812	1051-09-15	1052-09-03	year	385
502	Hijrī	0502	1108-08-18	1109-08-06	year	354
Muḥarram 557	Hijrī	0557-01	1161-12-28	1162-01-26	month	30
Safar 1248	Hijrī	1248-02	1832-06-30	1832-07-28	month	29

Table 3: Another sampling of dates from PGP documents in different calendars parsed by `Undate`. Earliest and latest dates are calculated in Gregorian / proleptic Gregorian calendar. Semantic durations like year and month result in different durations in days when working with multiple calendars.

3.2 Shakespeare and Company Project

The *Shakespeare and Company Project* is based on the materials from Sylvia Beach's famous English-language lending library that operated in Paris in the 1920s and 1930s. This project is one of the precursors that fed into the development of `undate`, with an implementation to support incomplete dates written on handwritten cards [3, 9].

This project includes borrowing events with unknown years; with `undate` we can calculate how long a book was borrowed even when we don't know the exact year. We use Gertrude Stein as an example here, since she is a well known figure with documented borrowing activity without known years (Figure 3), although she is far from the only one.[5] Stein has 46 borrow events with calculable duration; of those 7, or 15%, have no known year. If we limit our analysis to borrow events with known years, we arrive at an average borrow length of 32 days and a maximum of 91 days. However, when we include events with unknown years we get an average of 38 and a maximum of 126 days. In the context of this project, we calculate the duration of borrow events with unknown years as the shortest possible duration between the start and end date; that is, the the dates are in the same year (or following year, when the end date is an earlier month than the start date). This is based on our assumption that the clerks working in the book shop would have noted the year if the book was out for a longer time period, as they did in other cases.

The full code used to calculate durations and plot them is available in Appendix B. Here we

[5] Stein is famous enough that these unknowns could perhaps be resolved through research; that is not the case for the lesser known members of this library.

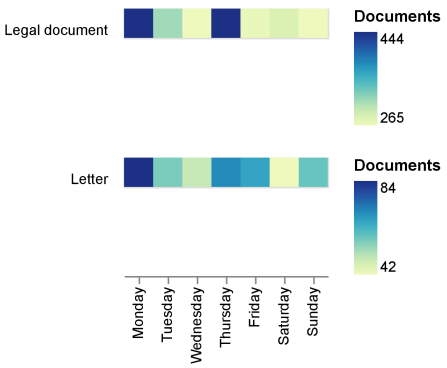

Figure 2: Legal documents and letters frequency by weekday.

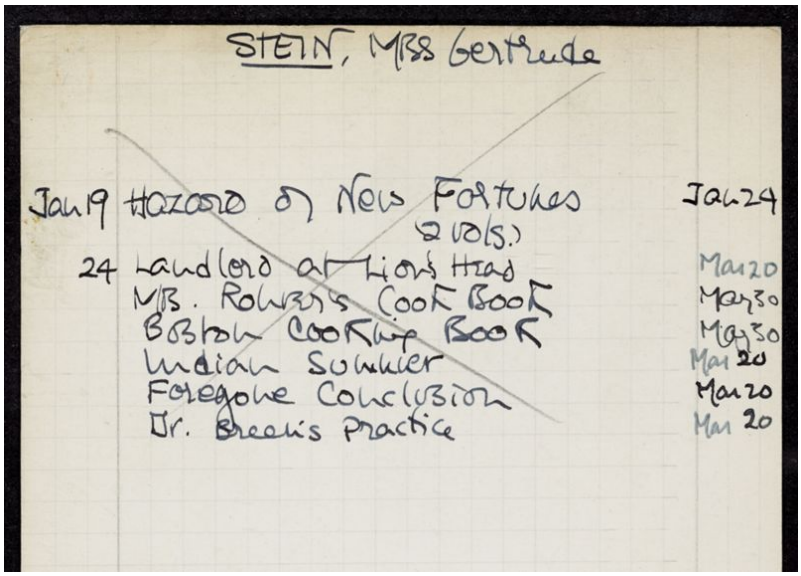

Figure 3: Detail from Gertrude Stein's lending library card showing handwritten borrow events with no year. [1, 2]

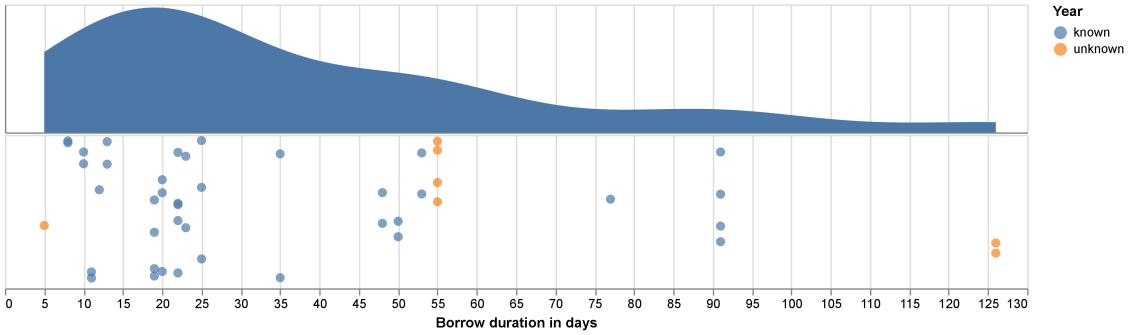

Figure 4: Raincloud plot showing how long Gertrude Stein typically kept the books she borrowed. Borrows with unknown years are highlighted in orange in the lower portion (the "rain") of the plot.

show a sample of the parsed borrow events (Table 4) and a raincloud plot of Stein's borrowing (Figure 4) to demonstrate the value of including partial information in our analysis. As you can see, leaving out partially known dates would greatly change the distribution.

Start Date	End Date	Duration	Title	Author
–01-19	–01-24	5	*A Hazard of New Fortunes*	Howells, William Dean
–01-24	–05-30	126	*Boston Cooking-School Cook Book*	Farmer, Fannie Merritt
–01-24	–03-20	55	*The Landlord at Lion's Head*	Howells, William Dean
–01-24	–05-30	126	*Mrs. Rorer's New Cook Book*	Rorer, Sarah Tyson
–01-24	–03-20	55	*Indian Summer*	Howells, William Dean
–01-24	–03-20	55	*A Foregone Conclusion*	Howells, William Dean
–01-24	–03-20	55	*Dr. Breen's Practice*	Howells, William Dean
1920-04-29	1920-06-03	35	*The Letters of George Meredith*	Meredith, George

Table 4: Sample borrowing activity for Gertrude Stein at the Shakespeare and Company lending library in Paris, including borrow events with unknown years but calculable durations.

4 Conclusion

This paper provides a demonstration of some of the current functionality of the `undate` python library, although it is certainly not exhaustive. I have barely touched on intervals and some of the calculations and logic around uncertainty that are currently supported. This package is still in active development, with tremendous potential in multiple directions. I and my collaborators plan to continue adding support for more calendars and formats (e.g., RDF and CIDOC-CRM), and improve the support for ambiguity and granularity in parsing and date precisions. Other avenues of interest are representing the temporal uncertainty in data visualizations based on `undate` parsing and analysis [12], or making more use of labels for analysis. Development on `undate` input and use cases from the community to steer and prioritize the work, to ensure that work is interpretable and usable in the context of a specific project while still being general enough to be usable and accurate across disparate projects. As `undate` matures, I hope to start seeing it used in more projects, and we would love to know if there is an interest or need in adapting it to other programming languages or extending it for specific ecosystems and frameworks.

Acknowledgements

This work is supported by the Center for Digital Humanities at Princeton University. Thanks to my reviewers, especially: David Ragnar Nelson, who caught and corrected several errors in the code examples introduced in the conversion from notebook to paper; and Julia Damerow, for suggestions to improve the flow.

References

[1] Beach, Sylvia. "Stein, Gertrude; Sylvia Beach Papers". C0108. Manuscripts Division, Department of Special Collections, Princeton University Library.

[2] "Gertrude Stein". Shakespeare and Company Project. Publisher: Center for Digital Humanities, Princeton University. URL: `https://shakespeareandco.princeton.edu/members/stein-gertrude/`.

[3] Koeser, Rebecca Sutton. "Coding with Unknowns". Dec. 2019. URL: `https://cdh.princeton.edu/blog/2019/12/05/coding-unknowns/`.

[4] Koeser, Rebecca Sutton. "Join me for a DHTech hackathon? It's an un-date!" Feb. 2023. URL: `https://dh-tech.github.io/blog/2023/02/09/hackathon-undate/`.

[5] Koeser, Rebecca Sutton, Crawford, Cole, Damerow, Julia, Vogl, Malte, and Casties, Robert. "undate python library". July 2025. DOI: `10.5281/zenodo.11068867`.

[6] Koeser, Rebecca Sutton, Damerow, Julia, Casties, Robert, and Crawford, Cole. “Undate: humanistic dates for computation”. In: *Computational Humanities Research 1* (2025), e5. DOI: 10.1017/chr.2025.10006.

[7] Koeser, Rebecca Sutton and Kotin, Joshua. “Shakespeare and Company Project Datasets”. Publisher: Princeton University. 2025. URL: https://doi.org/10.34770/kf6c-b079.

[8] Koeser, Rebecca Sutton and LeBlanc, Zoe. “Missing Data, Speculative Reading”. In: *Journal of Cultural Analytics 9*, no. 2 (May 2024). DOI: 10.22148/001c.116926.

[9] Kotin, Joshua and Koeser, Rebecca Sutton. “Shakespeare and Company Project Data Sets”. In: *Journal of Cultural Analytics 7*, no. 1 (Feb. 2022). DOI: 10.22148/001c.32551.

[10] Rustow, Marina. “Dating Problems? Ask the Princeton Geniza Project Team”. Nov. 2020. URL: https://cdh.princeton.edu/blog/2020/11/18/dating-problems-ask-princeton-geniza-project-team/.

[11] Rustow, Marina, Koeser, Rebecca Sutton, Richman, Rachel, Ryzhova, Ksenia, Bensalim, Amel, and Mohamed, Abdellatif. “Princeton Geniza Project Dataset”. July 2025. DOI: 10.5281/zenodo.15839055.

[12] Yau, Nathan. “Visualizing the Uncertainty in Data”. Jan. 2018. URL: https://flowingdata.com/2018/01/08/visualizing-the-uncertainty-in-data/.

A Parsing PGP document dates in multiple calendars

Princeton Geniza Project data includes documents with original dates in multiple calendars. `undate` does not yet provide an omnibus parser, so dates must be parsed according to their a known, supported calendar. The dates in this dataset are complicated and messy, and include qualifiers and uncertainty not yet supported by `undate` parsing. For demonstration purposes, this example code removes and ignores those complexities.

Adapted from an example notebook included in the `undate` software repository.

```
import re

from lark.exceptions import UnexpectedEOF
from lark.visitors import VisitError
import polars as pl


# set this to True to see details about parsing
VERBOSE_PARSE_OUTPUT = False


def remove_ordinals(val):
    # utility method to turn ordinals into numerals
    # i.e., 1st, 2nd, 3rd -> 1, 2, 3
    return re.sub(r"(\d+)(st|nd|rd|th)", "\\1", val)


def parse_original_date(doc_date_original, doc_date_calendar):
    # Map PGP original calendar to undate calendar converter
    undate_calendar = None
    if doc_date_calendar == "Anno Mundi":
```

```
        undate_calendar = "Hebrew"
    elif doc_date_calendar == "Hijrī":
        undate_calendar = "Islamic"
    elif doc_date_calendar == "Seleucid":
        undate_calendar = "Seleucid"

    if undate_calendar:
        value = doc_date_original

        # some dates have unknown digits, e.g. 1[.] Kislev 48[..] or 152[.]
        # Replace with zeros for now, since the calendar parsers
        # doesn't yet support this syntax
        # (even though Undate supports unknown digits)
        if "[." in value:
            if VERBOSE_PARSE_OUTPUT:
                print(f"ignoring missing digits for now {value}")
            value = (
                value.replace("[.]", "0")
                .replace("[..]", "00")
                .replace("[...]", "000")
            )

        # some dates have inferred numbers,
        # e.g. Friday, [25] Nisan [4810] or 8 Elul (4)811'
        # for now, we strip out brackets before parsing;
        # in future, we could indicate uncertainty based on these
        value = (
            value.replace("[", "")
            .replace("]", "")
            .replace("(", "")
            .replace(")", "")
        )

        # for now, remove modifiers not supported by the undate parser:
        #   Late Tevet 4903, Last decade of Kislev 5004, first third of ...
        modifiers = [
            "Late ",
            "(first|middle|last)( third|half|decade|tenth)? (of )?",
            "(Beginning|end) of ",
            "last day",
            "First 10 days",
            " of",
            "spring",
            "decade ",
            "night, ",
        ]
        for mod in modifiers:
            value = re.sub(mod, "", value, flags=re.I)
```

```
        # about 62 have ordinals; strip them out
        value = remove_ordinals(value)

        # parse the simplified, cleaned up date string
        # with the specified calendar
        try:
            return Undate.parse(value, undate_calendar)
        except (VisitError, ValueError, UnexpectedEOF) as err:
            if VERBOSE_PARSE_OUTPUT:
                print(f"Parse error : {value} ({undate_calendar}): {err}")
            return None

# Use Polars to load data file into a DataFrame
# Filter to documents with standard dates
pgp_documents_df = pl.read_csv("pgp_documents.csv") \
    .filter(
        pl.col("doc_date_standard").is_not_null()
    )

# Create a struct of original date and calendar,
# which we can pass to our parse method.
# If parsing succeeds, returns and Undate or UndateInterval
pgp_documents_df = pgp_documents_df.with_columns(
    undate_orig=pl.struct(
        "doc_date_original", "doc_date_calendar"
    ).map_elements(
        lambda row: parse_original_date(
            row["doc_date_original"], row["doc_date_calendar"]
        ),
        return_dtype=pl.datatypes.Object,
    )
)
days = [
    "Monday",
    "Tuesday",
    "Wednesday",
    "Thursday",
    "Friday",
    "Saturday",
    "Sunday",
]

# filter to records that were successfully parsed;
# determine weekday for use with day-level precision dates

pgp_documents_df = (
    pgp_documents_df.filter(pl.col("undate_orig").is_not_null())
    .with_columns(
        pl.col("type").fill_null("Unknown"),  # set null type to unknown
```

```
        orig_date_precision=pl.col("undate_orig").map_elements(
            lambda x: str(x.precision).lower(),
            return_dtype=pl.datatypes.String,
        ),
        undate_weekday=pl.col("undate_orig").map_elements(
            lambda x: x.earliest.weekday, return_dtype=pl.datatypes.Int32
        ),
    )
    .with_columns(
        undate_weekday_name=pl.col("undate_weekday").map_elements(
            lambda x: days[x], return_dtype=pl.datatypes.String
        )
    )
)

# filter to relevant fields and display the first ten rows
pgp_documents_df.select(
    "pgpid",
    "type",
    "doc_date_original",
    "doc_date_calendar",
    "doc_date_standard",
    "undate_orig",
    "orig_date_precision",
    "undate_weekday_name",
).head(10)
```

B Determine and plot durations for Shakespeare and Company Project borrowing

This project includes borrowing events with unknown years; with `undate` we can calculate how long a book was borrowed even when we don't know the year.

First we define a couple of methods that we'll use with the data:

- `undate_duration` to parse the dates and calculate duration in days
- `known_year` to provide an indicator for whether the year was known, which we'll use when plotting the data

We load the full events data from the Shakespeare and Company published datasets [7] and filter it to borrow events for Gertrude Stein based on `event_type` and `member_ids`. Then we use the custom methods defined first to calculate borrow durations and add a variable indicating whether the year is known.

Finally, we define and use a method to generate a raincloud plot, showing the distribution of borrow durations in two different ways, and color the rain drops in the lower portion of the plot to highlight the durations based on borrow events with unknown years. (The raincloud plot is adapted from prior work in [8].)

```
import altair as alt

from undate import UndateInterval
from undate.date import ONE_DAY, UnInt
```

```
from undate.converters.iso8601 import ISO8601DateFormat


def undate_duration(start_date, end_date):
    isoformat = ISO8601DateFormat()

    unstart = isoformat.parse(start_date)
    unend = isoformat.parse(end_date)
    interval = UndateInterval(earliest=unstart, latest=unend)

    # borrow durations in Shakespeare and Company Project were defined as
    # not including both ends (or half of the day on both ends);
    # to reconcile differences between duration logic with undate,
    # which includes both endpoints, we subtract one day
    return (interval.duration() - ONE_DAY).days


def known_year(date):
    return "known" if ISO8601DateFormat().parse(date).known_year else "unknown"


stein_borrow_events_df = (
    pl.read_csv("SCoData_events_v2.0_2025.csv", infer_schema_length=10000)
    .filter(
        pl.col("member_ids").eq("stein-gertrude"),
        pl.col("event_type").eq("Borrow"),
        # for simplicity, we're going to limit to events that Project
        # provides a borrow duration for; this indicates
        # the duration is calculable (i.e., start and end dates known)
        pl.col("borrow_duration_days").is_not_null(),
    )
    .select(
        "event_type",
        "start_date",
        "end_date",
        "borrow_duration_days",
        "item_title",
        "item_authors",
        "item_year",
    )
)

# calculate durations; returns a dataframe with one column
duration_df = stein_borrow_events_df.select("start_date", "end_date").map_rows(
    lambda x: undate_duration(x[0], x[1]), return_dtype=pl.datatypes.Int32
)

# add fields to the main dataframe for duration and whether year is known
stein_borrow_events_df = stein_borrow_events_df.with_columns(
```

```
    undate_duration=duration_df["map"],
    known_year=stein_borrow_events_df["start_date"].map_elements(
        known_year, return_dtype=pl.datatypes.String
    ),
)


def raincloud_plot(dataset, fieldname, field_label, color_opts=None):
    """Create a raincloud plot for the density of the specified field
    in the given dataset. Takes an optional tooltip for the strip plot.
    Returns an altair chart."""

    # create a density area plot of specified fieldname

    duration_density = (
        alt.Chart(dataset)
        .transform_density(
            fieldname,
            as_=[fieldname, "density"],
        )
        .mark_area(orient="vertical")
        .encode(
            x=alt.X(
                fieldname, title=None, axis=alt.X(labels=False, ticks=False)
            ),
            y=alt.Y(
                "density:Q",
                # suppress labels and ticks because we're going to combine this
                title=None,
                axis=alt.Axis(
                    labels=False, values=[0], grid=False, ticks=False
                ),
            ),
        )
        .properties(height=100, width=800)
    )

    # Now create jitter plot of the same field
    # jittering / stripplot adapted from https://stackoverflow.com/a/71902446/9706217

    chart_color_opts = {}
    if color_opts is not None:
        chart_color_opts = {"color": color_opts}

    stripplot = (
        alt.Chart(dataset)
        .mark_circle(size=50)
        .encode(
            x=alt.X(
```

```
                fieldname,
                title=field_label,
                axis=alt.Axis(labels=True),
            ),
            y=alt.Y("jitter:Q", title=None, axis=None),
            **chart_color_opts,
            # color=alt.Color(color_by),  # .scale(**color_scale),
        )
        .transform_calculate(jitter="(random() / 200) - 0.0052")
        .properties(
            height=120,
            width=800,
        )
    )

    # use vertical concat to combine the two plots together
    raincloud_plot = alt.vconcat(duration_density, stripplot).configure_concat(
        spacing=0
    )
    return raincloud_plot


stein_borrows_plot = raincloud_plot(
    stein_borrow_events_df,
    "undate_duration",
    "Borrow duration in days",
    alt.Color("known_year", title="Year"),
)
```