Multimedia Tools and Applications (2024) 83:32141–32157
https://doi.org/10.1007/s11042-023-16553-w

VedicDateTime: An R package to implement Vedic calendar system

Neeraj Dhanraj Bokde[1] · Prajwal Kailasnath Patil[2] · Saradindu Sengupta[3] · Manisha Sawant[4] · Andrés E. Feijóo-Lorenzo[5]

Received: 27 December 2022 / Revised: 14 July 2023 / Accepted: 18 August 2023 /
Published online: 19 September 2023

Abstract
Calendar systems adopted across the world are either solar, lunar or lunisolar, based on the movements of the sun, the moon, or both. The Gregorian (solar) calendars are considered as time references for modern computations. However, Vedic calendars, being lunisolar, can be more effective for the analysis of activities that depend on both celestial bodies. In this paper, we present `VedicDateTime`, an open-source framework that implements the Vedic calendar and provides conversions for Gregorian dates. Along with package details, we also provide two case studies that make use of the proposed package for time-series analysis. The objective of this paper is to motivate researchers to explore the potential of the Vedic calendar from the perspective of time series analysis.

Keywords Vedic calendar · Gregorian calendar · Panchanga · Time series analysis · R package

Mathematics Subject Classification (2010) 00-01 · 99-00

Vedic entities referred to in this article and their corresponding English meanings are represented in the following table

Panchanga	Five arms of vedic calendar
Tithi	A lunar day

✉ Neeraj Dhanraj Bokde
neerajdhanraj@qgg.au.dk

1 Center for Quantitative Genetics and Genomics, Aarhus University, 8000 Aarhus, Denmark

2 KLE Technological University, B. V. Bhoomaraddi Campus, Vidyanagar, Hubballi, Karnataka 580031, India

3 Indian Institute of Information Technology and Management, Kerala, Kazhakkoottam, Kerala 695581, India

4 Department of Electronics and Communication Engineering, National Institute of Technology, Calicut 673601, India

5 Departamento de Enxeñería Eléctrica, Universidade de Vigo, EEI, Campus de Lagoas-Marcosende, 36310 Vigo, Spain

Vara	The day of the week
Karana	Half *tithi*
Nakshatra	Constellation
Shukla paksha	Waxing phase of the moon
Krishna paksha	Waning phase of the moon
Purnima	Full moon
Panchami	Fifth lunar day
Ayanamsa	Related to the precession of equinoxes
Rashi	Zodiac
Lagna	Sun sign
Masa	Month
Samvatsara	Year

1 Introduction

Calendars are believed to reflect culture and religious activities. Many daily activities and traditional events are planned with the help of a calendar. They help us gain better access to long-term memory and life history information. Although the Gregorian calendar is followed worldwide for national and international activities, many religions across the world rely on their religious calendar for their cultural and traditional activities.

The `VedicDateTime` [1] is an R package that provides a platform for the Vedic calendar system. It provides several functionalities to facilitate the conversion between Gregorian and Vedic calendar systems and helps examine its impact in the time series analysis domain. It aims to analyze various natural time series influenced by both sun and moon with the help of a (lunisolar) calendar. It implements five important parts of the Vedic calendar and provides conversion from a Gregorian date or a Julian day to Vedic calendar entities. A few more entities in the Vedic calendar system and related conversions are also available in this package. Researchers can use this package to analyze the time series concerning different time factor s and explore its impact.

In this package, different entities of the Vedic calendar are implemented. This package implements efficient and reliable functions for computing *Panchanga's* for a given Gregorian date. Section 2 presents the motivation for this package. A brief introduction to the Vedic calendar and the terminology of the Vedic calendar system is presented in Sect. 3. The package is described, in detail, in Sect. 4.

2 Motivation

Calendars are integral parts of our life and culture. Over time, various calendar systems have been developed in different countries, reflecting their traditions and religious events [2–4]. Gregorian, Vedic, Chinese, and Hebrew calendars are some of the existing ones [5]. These calendars follow either the solar cycle, the moon cycle, or both. Various researchers are interested to develop software for these calendar systems. Table 1 shows different software packages available in `R` and Python for different calendars. In the Gregorian calendar, the number of days in each month of a year is fixed without a justified logic behind it. Whereas, the Vedic calendars are based on the motion of both the sun and moon [6]. The Vedic calendar length corresponds to the solar year, which is the time taken by Earth to revolve around the

Table 1 R and Python packages for different calendar systems

Package Name	Description of Calendar system	Language
`Pyluach` [7]	A Python package for manipulating Hebrew (Jewish) calendar dates and Hebrew-Gregorian conversion. (URL)	Python
`Lunisolar` [8]	A Python library that handles conversions between the Chinese calendar and the Gregorian calendar. (URL)	Python
`Pyroj` [9]	A python package for converting Gregorian and Solar dates to Kurdish date. (URL)	Python
`sanatantime` [10]	It is a Python module to convert christian system time to vedic sanatan time. (URL)	Python
`jalcal` [11]	R package `jalali`, also known as Persian, Solar Hijri and Hijri Shamsi calendar is the official calendar of Iran and Afghanistan. (URL)	R
`jcalendaR` [12]	`jcalendaR` is an R package for converting between current Japanese calendar, old Japanese calendar that Kyureki, and Julian and (proleptic) Gregorian calendar. (URL)	R
`ethiopianDate` [13]	This is an R package for converting Gregorian dates to the Ethiopian calendar and vice-versa (URL)	R
`maya` [14]	`maya` provide R functions to convert between the Mayan calendar dates and Gregorian dates. (URL)	R

sun and the months are lunar, which is the time taken by the moon to revolve around the Earth. These lunar months and solar years are well balanced in the Vedic calendar.

The time factor is an important parameter in time-series analysis. In [15], patterns and trends in univariate time-series data are identified by considering multiple time scales simultaneously. And the authors of [15] claimed that such clustering of time series with different time scales provides insight into both standard and exceptional patterns. In [16], calendar effects on monthly time-series are analyzed. They analyzed the effects of changing months and days of the week and suggested that for effective analysis of time series the calendar effects should be removed. This implies that only the Gregorian calendar is not sufficient to analyze the patterns and trends in the time series.

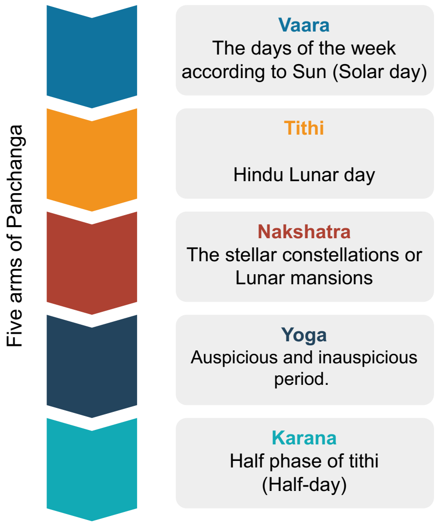
There exists a great influence of both sun and moon on various behavioral activities on Earth [17], e.g., lunar rhythms are responsible for tidal effects on seas, and major climate fluctuations [18], and the sun is responsible for variations in temperature and wind. These activities are studied and analyzed by forming a time series. For experimental analysis of such natural time series, the time factor generally refers to Gregorian (solar) calendar. However, to analyze the activities that have the impact of the sun and moon both, the time factor must be dependent on the sun and moon both i.e. lunisolar. If the natural time series are formatted using the Vedic (lunisolar) calendar format, then it may be possible to extract both lunar (moon) and solar (sun) based patterns within the time series more efficiently. In [19], Bokde analyzed the time factor of temperature time series using the Gregorian and Vedic calendars. The correlation-based analysis in [19] shows that the Vedic calendar can reveal interesting facts and may improve the time-series analysis accuracies.

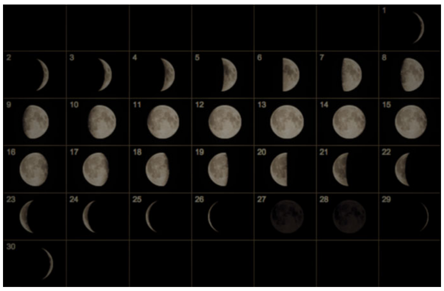
Time series are widely used in many fields ranging from weather forecasting, and forecasting financial time series to wind power and speed prediction. Instead of relying only on the Gregorian calendar system, the `VedicDateTime` package will help researchers to explore a new time factor. As shown in Table 1 various software packages for different calendar systems are available. However, the software package for the Vedic calendar system is not available. So the `VedicDateTime` package presented in this paper will facilitate the researchers to analyze and compare their results with reference to different time factors.

3 Panchanga of Vedic calendar

In order to understand how `VedicDateTime` package works, the concepts related to the Vedic calendar must be clear. *Ayanamsha* and *Panchanga* are important entities in this calendar system. In order to explain how the Vedic calendar works, some basic facts from astronomy must be recalled. As we all know, the Earth revolves anti-clockwise around the sun in an elliptical orbit. And the time taken by the earth to arrive at one vernal equinox from the other vernal equinox after one revolution is called a tropical year. At the end of the tropical year, if we consider the position of the earth with respect to a fixed star in the zodiac, the earth appears to lie 50.26 s of celestial longitude to the west of its original position. In order to arrive at the same position with respect to a fixed star of the zodiac, the time taken is called a sidereal year. This is the result of the precession of the earth's axis due to its wobble in a clockwise direction. The precession of the equinoxes increases by 50.2 s every year. The distance between the Vernal equinox and the 1st point of Aries (*Mesha*) on the fixed zodiac is progressively increasing. The distance at any given point of time is called *Ayanamsha*.

The *Panchanga* means 'five arms', it consists of the five most important parts, i.e., *Tithi, Vaara, Nakshatra, Yoga, and Karana* [20]. The five arms of *Panchanga* are shown in Fig. 1a. The calculations of these five entities are based on either moon or sun or both. The first arm, the *Vaara* or the day of the week is based on the sun alone; *Tithi* and *Karana* are based on the moon alone; *Nakshatra* and *Yoga* are based upon both moon and sun. This makes the Vedic calendar a true lunisolar calendar. A brief description of these five entities is listed below.


Five arms of Panchanga
Vaara
The days of the week according to Sun (Solar day)
Tithi
Hindu Lunar day
Nakshatra
The stellar constellations or Lunar mansions
Yoga
Auspicious and inauspicious period.
Karana
Half phase of tithi (Half-day)
(a)



(b)

Fig. 1 (1a) *Panchanga* and (1b) *Tithi* of Vedic calendar

1. *Tithi*: A lunar day is termed as *tithi*. The moon illuminates because it reflects part of the light received from the sun. The different phases of the moon are due to the relative positions of the sun and the moon with respect to the Earth. When the moon traverses 12° in longitude relative to the sun, a *tithi* is completed. In one month, fifteen *tithis* are in the waxing phase (known as *Shukla paksha*) of he moon while the end *tithi* is the full moon day (called *Purnima*). The remaining fifteen *tithis* are in the waning phase (known as *Krishna paksha*) with the end of tithi being the new moon day *Amavasya*). These phases together constitute a lunar month, which is of thirty *tithis*. Names are associated with each of these *tithis*. Thirty *tithis* or phases of the moon in a lunar month are hown in Fig. 1b.
2. *Vara*: As in the Gregorian calendar, *vaara* is the day of the week. The names of the seven *varas* are derived from the names of the seven *Grahas* i.e. planets which are considered as heir lords. The *varas* and their corresponding names as per the Gregorian calendar are presented in Table 2.
3. *Nakshatra*: As per Vedic terminology, *nakshatra* or constellation means a sky map or a star map. There are a total of 27 *nakshatras* and each of them has a specific name. The Moon resides in each of these constellations for a day [21]. Figure 2 shows the location of *nakshatras* and zodiac. The position of any planet or star in the sky is fixed with reference to the zodiac. And the earth is encircled by zodiacs so it covers 360 degrees. This 360 degrees span is equally divided into a span of 13 degrees 20 min, each one starting from zero Aries, and each *nakshtra* is placed on it [22] as shown in Fig. 2. Each *nakshatra* spans 13 degrees 20 min where the moon resides for one day. Therefore, the absolute longitude of the moon is used to calculate the *nakshatra* of the day. The location of *nakshatra* with respect to zodiacs is shown in Fig. 2.
4. *Yoga*: There are 27 *yogas*, each of them measuring 13 degrees and 20 min of the arc. *Yoga* is the sum of sidereal (*Nirayana*) longitudes of sun and moon in the multiples of 13 degrees 20 min. When the sidereal longitudes of the sun and moon are added and they are divided by 13 degrees 20 min, the quotient plus one gives the respective *yoga*.
5. *Karana*: A *Karana* is half of a tithi or when the moon traverses 6° in longitude relative to the sun. In 30 *tithis* of a lunar month, there are 60 *Karanas* or half-*tithis*.

4 Introduction to R package VedicDateTime

The `VedicDateTime` package provides functions to convert Gregorian date-time to Vedic date-time based on the calculations of the *Panchanga*. These functions rely on the

Table 2 Names of *varas* as per Vedic calendar and their corresponding names in the Gregorian calendar

Sr. No.	*Varas* name	Name of the day as per Gregorian calendar
1	*Ravivara*	Sunday
2	*Somvara*	Monday
3	*Mangalvara*	Tuesday
4	*Budhvara*	Wednesday
5	*Guruvara*	Thusday
6	*Shukrvara*	Friday
7	*Shanivara*	Saturday

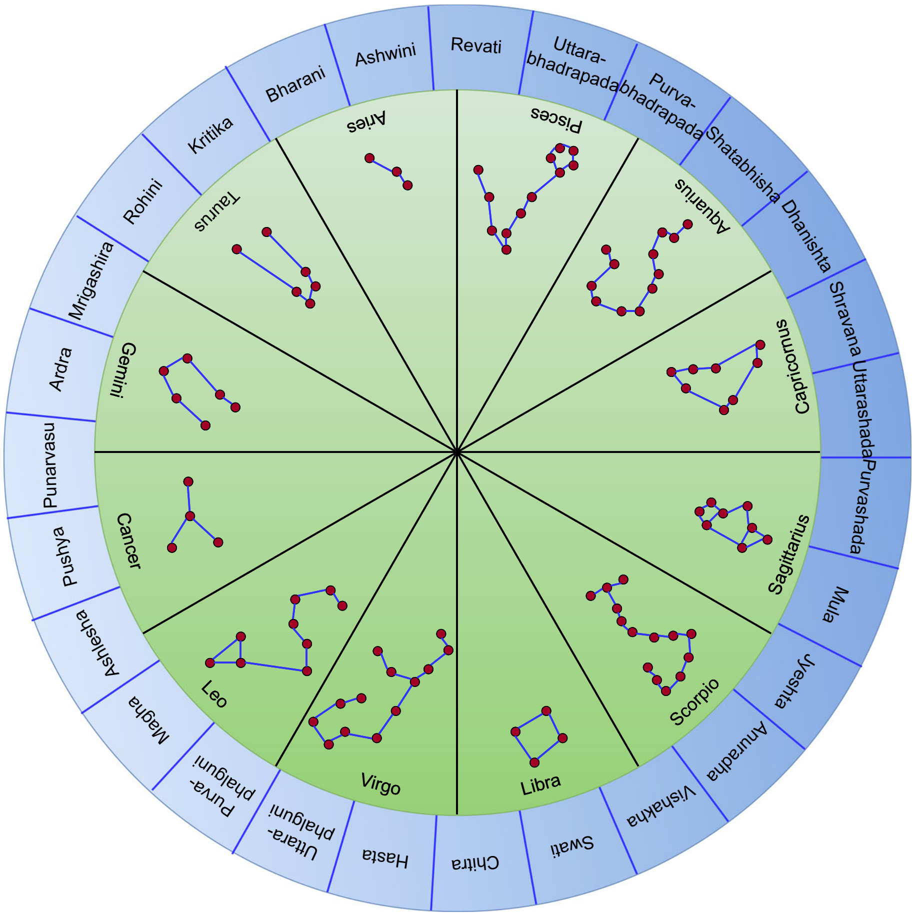


Fig. 2 Location of *Nakshatra* as per Vedic calendar

`gregorian_to_jd( )` function, which converts a Gregorian date to a Julian day. The package includes functions such as `tithi`, `vaara`, `nakshatra`, `yoga`, and `karana`. These functions require the input parameters of the date (Julian day number) and the location (latitude, longitude, and time zone).

1. `tithi()`

   Listing 1: Syntax of `tithi()` function.

```
tithi(jd,place)
tithi( gregorian_to_jd(date), place)
```

   The function tithi(jd, place) calculates the *tithi* (a phase of the moon) for a given Julian day and place. The tithi at sunrise is determined by the difference between lunar and solar longitudes divided by 12. The fractional part of the *tithi* indicates the remaining degrees to the next *tithi*, which helps determine when that *tithi* ends. The time required to complete the remaining degrees is found using Inverse Lagrange's interpolation with lunar and solar longitudes at 0.25 day intervals from sunrise. The resulting *tithi* and its ending time are added to the output vector.

Listing 2: Use of `tithi()` function to convert a Julian day number into the corresponding *tithi*.

```
# Julian day number
jd <- 2459778
# Latitude, Longitude, and timezone of the location
place <- c(15.34, 75.13, +5.5)
tithi(jd,place)
#> [1] 20 20 55 35
```

In Listing 2, the first number in the output represents a specific *tithi*. There are 30 *tithis* in total, and in this case, the number 20 represents *Krishna Paksha Panchami*. The following three numbers indicate the time at which this *tithi* ends (20 h, 55 min, and 35 s). Sometimes, two *tithis* can occur on a given date. In such cases, the *tithi* of the next day is checked to determine if the current *tithi* is skipped. If it is skipped, the skipped *tithi* is included in the output along with its ending time.

Listing 3: Use of `tithi()` function to convert a Gregorian date into the corresponding *tithi*.

```
tithi(gregorian_to_jd(17,6,2022),c(15.34, 75.13, +5.5))
#> [1] 18  6 11 26 19 26 59 58
```

For the given input parameters in Listing 3, the `tithi( )` function returns 8 numbers as output. The first 4 numbers represent the first *tithi* and its ending time, while the next 4 numbers represent the second *tithi* and its ending time. If the timings exceed 24:00, it means that they indicate a time past midnight. To determine the *tithi* name associated with a *tithi* number, `get_tithi_name(jd, place)` function is used. The *tithi* calculation process is illustrated in a flow chart (Fig. 3a).

2. `Vaara()`
   Vaara is the day of the week, very similar to the day in the Gregorian calendar. There are seven *vaaras* in one week. *Vaara* of a Julian day is calculated as follows.

Listing 4: Syntax of `vaara()` function.

```
vaara(jd)
```

Listing 5: Use of `vaara()` function to convert a Julian day into the corresponding day of the week (*Vaara*).

```
vaara(2459778)
#> [1] 1
```

The output of Listing 5 is 1, which is *’Ravivara’* as per the Vedic calendar. The name associated with the Julian day is obtained using `get_vaara_name( )` function as shown in Listings 6 and 7.

Listing 6: Use of `get_vaara_name()` function to convert a Julian day into the name of the corresponding day of the week (*Vaara*).

```
get_vaara_name(2459778)
#> [1] "Ravivar"
```

Listing 7: Use of `get_vaara_name()` function to convert a Gregorian day into the name of the corresponding day of the week (*Vaara*).

```
get_vaara_name(gregorian_to_jd(6,8,2022))
#> [1] "Shanivar"
```

3. nakshatra(): The function calculates the *Nakshatra* of a given day based on the moon's position. It involves determining the moon's sidereal longitude by subtracting the *Ayanamsa* (related to the precession of equinoxes) from its tropical(*Sayana*) longitude. The *Ayanamsa* used is the *Lahiri Ayanamsa*. The sunrise for the specified date and location is obtained using the Swiss Ephemeris, providing a Julian day number. To calculate the *Nakshatra*, the *Nirayana* longitude of the moon is computed by subtracting the *Ayanamsa* from its *Sayana* longitude. This *Nirayana* longitude is then multiplied by 27 and divided by 360 to obtain the *Nakshatra* as a decimal number. The ending time of the current *Nakshatra* is determined by checking for the starting time of the next *Nakshatra* using

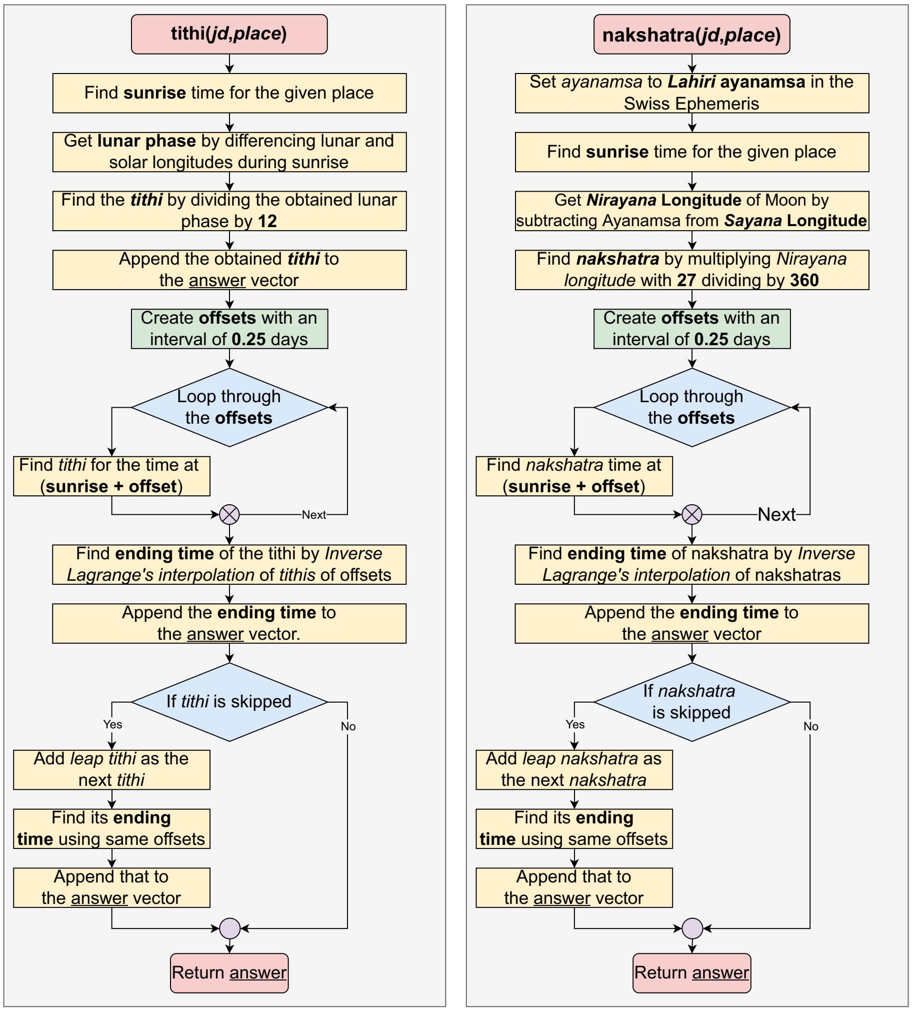

tithi(jd,place)
Find sunrise time for the given place
Get lunar phase by differencing lunar and solar longitudes during sunrise
Find the tithi by dividing the obtained lunar phase by 12
Append the obtained tithi to the answer vector
Create offsets with an interval of 0.25 days
Loop through the offsets
Find tithi for the time at (sunrise + offset)
Next
Find ending time of the tithi by Inverse Lagrange's interpolation of tithis of offsets
Append the ending time to the answer vector.
If tithi is skipped
Yes
No
Add leap tithi as the next tithi
Find its ending time using same offsets
Append that to the answer vector
Return answer
nakshatra(jd,place)
Set ayanamsa to Lahiri ayanamsa in the Swiss Ephemeris
Find sunrise time for the given place
Get Nirayana Longitude of Moon by subtracting Ayanamsa from Sayana Longitude
Find nakshatra by multiplying Nirayana longitude with 27 dividing by 360
Create offsets with an interval of 0.25 days
Loop through the offsets
Find nakshatra time at (sunrise + offset)
Next
Find ending time of nakshatra by Inverse Lagrange's interpolation of nakshatras
Append the ending time to the answer vector
If nakshatra is skipped
Yes
No
Add leap nakshatra as the next nakshatra
Find its ending time using same offsets
Append that to the answer vector
Return answer
(a)
(b)

Fig. 3 Flowchart of (a) *Tithi* and (b) *Nakshatra* calculation

Lagrange's inverse interpolation of *Nirayana* lunar longitudes at 0.25 intervals of the day. The *Nakshatra* and its ending time are added to the output vector. In some cases, two *Nakshatras* may occur on a given date. To account for skipped *Nakshatras*, the function checks the *Nakshatra* of the next day. If a *Nakshatra* is skipped, it is included in the output along with its ending time. This process is summarized in the flowchart shown in Fig. 3b.

Listing 8: Syntax of `nakshatra()` function.

```
nakshatra(jd, place)
```

Listing 9: Use of `nakshatra()` function to convert a Julian day into the corresponding (*Nakshatra*).

```
# Julian day number
jd <- 2459778
# Latitude, Longitude, and timezone of the location
place <- c(15.34, 75.13, +5.5)
nakshatra(jd,place)
#> [1] 25 24 24  1
```

In Listing 9, the first number represents the number associated with the respective *nakshatra*. 25 represents *Purvabhadrapada*. This *nakshatra* ends on the next day at 0 h, 24 min, and 1 s, which is represented by the rest of the numbers.

4. yoga():
There are 27 *yogas*, each of them measures 13 degrees, 20 min of an arc. To calculate *yoga*, the sidereal longitudes of the sun and moon are added and divided by 13 degrees 20 min. The quotient of the division plus one gives the number which denotes the respective Yoga as shown in Listing 11.

Listing 10: Syntax of `yoga()` function.

```
yoga(jd, place)
```

Listing 11: Use of `yoga()` function to convert a Julian day into the corresponding (*Yoga*).

```
# Julian day number
jd <- 2459778
# Latitude, Longitude, and timezone of the location
place <- c(15.34, 75.13, +5.5)
yoga(jd,place)
#> [1]  5 27 26 12
```

For given input parameters, `yoga( )` returns a number associated with the respective *yoga* and the remaining three numbers the time at which *yoga* ends.

5. karana():
There are four fixed *karanas* that occur only once in a lunar month and the remaining seven *karanas* recur eight times during the rest of the lunar month. For a given date and place, firstly *tithi* is calculated, and then, using *tithi*, the corresponding *karanas* are calculated as shown in Listing 13.

Listing 12: Syntax of `karana()` function.

```
karana(jd, place)
```

Listing 13: Use of `karana()` function to convert a Julian day into the corresponding (*karanas*).

```
# Julian day number
jd <- 2459778
# Latitude, Longitude, and timezone of the location
place <- c(15.34, 75.13, +5.5)
karana(jd,place)
#> [1] 39 40
```

For given input parameters, `karana( )` returns two numbers associated with the respective two *karanas*. The names of the *karana* are obtained using `get_karana_name( )` as shown in Listing 14.

Listing 14: Use of `get_karana_name()` function to convert a Julian day into the names of the corresponding (*karanas*).

```
# Julian day number
jd <- 2459778
# Latitude, Longitude, and timezone of the location
place <- c(15.34, 75.13, +5.5)
get_karana_name(jd,place)
#> [1] "Kaulava-Taitila"
```

The remaining functions available in `VedicDateTime` package are listed in Table 3.

Table 3 Five arms of *Panchanag* in Vedic calendar

Sr. No.	Function name	Description
1	`tithi( )`	Tithi can be called as a lunar day. When the moon traverses 12^0 in longitude relative to the sun, a tithi is completed. As per vedic calendar, there are 30 tithis. This function returns one of the tithi corresponding to the input Julian day or Gregorian day
2	`vaara( )`	Vaara is the day of the week, very similar to the Gregorian calendar. $vara(jd)$ returns one of the seven varas in a week for a given Julian day
3	`nakshatra( )`	There are 27 nakshatras recognised in the Vedic astrology. The absolute longitude of the Moon is used to calculate the Nakshatra of the day. This function returns four numbers. The first one denotes the nakshatra's number and the other three numbers denote the time in hr:min:sec at which the corresponding nakshatra ends
4	`yoga( )`	There are 27 Yogas, each measure $13^0$20' of the arc. Yoga is the sum of sidereal (Nirayana) longitudes of the sun and the moon in multiples of 13^0 20'. This function returns one of the 27 yoga along with its end time for the given date and place
5	`karana( )`	A Karana is half of a tithi or when the moon traverses 6^0 in longitude relative to the sun. In 30 tithis of a lunar month, there are 60 Karanas or half-tithis. This function returns a pair of Karanas for the given Julian day and a place

5 Demonstration of case studies using `VedicDateTime`

This section demonstrates the applicability of the VedicDateTime package to analyze the effects of calendar seasonality on economic output and regressor on industrial production of India pre and post-Diwali festival. Calendars have an enormous effect on economic, social, and cultural behaviours but nowhere it is more profound than a country or region's economic output indicators such as industrial output, CPI, stock-market transaction, and import exports. But in a globalized world, all of the economic time series is modelled around the Gregorian calendar system. In a country like India, with multiple regions and festivals from multiple religions as well, a single calendar will not be sufficient to model various seasonal components. For example, during the festival month of Diwali, the consumer consumption rate is usually higher and industrial output is generally lower. In this section, we present two case studies where the VedicDateTime package is used to dig for more insights from the data (Table 4).

Table 4 Functions in `VedicDateTime` Package

Functions	Description
`rashi( )`	The zodiac is divided into 12 parts Rashi(moon sign) represents
`get_rashi_name( )`	the position of the moon on the zodiac at a given time `rashi( )` returns number and get_rashi_name( ) returns the name of the rashi
`lagna( )`	*Lagna* (sun sign) represents the position of the sun on the zodiac
`get_lagna_name( )`	at a given time. Output of `lagna( )` is a number and *lagna's* name is returned by `get_lagna_name( )`
`masa( )`	*Masa* is the lunar month in the Vedic calendar system. Every *masa* has fixed number of *tithis* and 2 *pakshas*
`get_masa_name( )`	Number associated with `masa( )` for the given Julian day is calculated from *tithi*, sunrise of given day. `get_masa_name( )` returns the name of the *masa*
`ritu( )`	*Ritu* means season. There are 6 Ritus which represent the 6
`get_ritu_name( )`	seasons. ritu( ) returns number associated with *ritu* from a *masa* and `get_ritu_name( )` returns the name of the ritu
`samvatsara( )`	*Samvatsara* means Year in Sanskrit. It is a cycle of 60 *Samvatsaras* in *Panchanga* which is of 60 years
`get_samvatsara_name( )`	The function `samvatsar( )` takes two inputs Julian day and *masa* number and returns the number associated with the name of the samvatsara. There are 60 names of the samvatsara and get_name_samvatsara ( ) returns one of them for given input parameters
`sun_longitude( )`	returns solar longitude for a given Julian day number
`moon_longitude( )`	`moon_longitude ( )` returns lunar longitude for a given Julian day number
`sunrise( )`	`Sunrise( )` returns sunrise time for a given date and place is returned
`sunset( )`	`Sunset( )` returns the sunset time for a given date and place
`moonrise( )`	rise time of moon for a given date and place is obtained by `moonrise ( )`
`moonset( )`	Moonset for a given date and place
`gregorian_to_jd( )`	This function converts Gregorian date to Julian day number at 00:00 UTC
`jd_to_gregorian`	Julian day number to Gregorian date conversion is done using `jd_to_gregorian( )`

5.1 Effects of calendar seasonality on economic output

Calendar has an enormous effect on economic, social, and cultural behaviors but nowhere it is profound than a country or region's economic output indicators such as industrial output, CPI, stock-market transaction, import–exports but in a globalized world, all of the economic time-series modelling is modeled around Gregorian calendar system. In a country like India with multiple regions and festivals from multiple religions as well, a single calendar will not be sufficient to model various seasonal components. For example, during the festival month of Diwali, the consumer consumption rate is usually higher and during the month, industrial output is generally lower (Table 5).

There have been significant work published already to identify and remove multiple lagged seasonality from time-series [23] considering technical papers published from the US Bureau of Census [24]. There also have been significant work done to identify and remove seasonality, especially concerning religious festival based on the Gregorian calendar by the Bank of Spain [25]. Similar work has been presented for lunar [26] and luni-solar [27] based calendar system and also in the context of Indian seasonality effect present [28–30].

We have performed a comparative study to understand calendar seasonality creeping in while modelling seasonal decomposition on time-series objects, especially on econometric data sets. To understand if any methodologies can be obtained and to provide a better, quantitative analysis of the seasonality, we used India's Industrial output data after the year 2000 and modelled holiday seasonality with Diwali dates corresponding to that. To understand the effect of seasonality, dates corresponding to India's industrial production data are converted into Vedic calendar dates using [1]. The following are the steps to convert the time series from the Gregorian calendar into the Vedic calendar (details available in supplementary material: Doc1).

- Step 1: View India Industrial Production (IIP) Data from [31]
- Step 2: Convert Gregorian dates of India Industrial Production (IIP) data to Julian dates.
- Step 3: Select the desired location for the date conversion.
- Step 4: Convert Julian dates to Vedic calendar dates.

With this conversion, there will be two different time-series objects of India's industrial production output, one with Gregorian calendar dates and another with Vedic calendar dates. Both time series are modelled and in both the cases, a seasonal decomposition was observed visible along the red line, and [32] was used for seasonality decomposition. Due to the change in the type of calendar, missing values were omitted to keep the frequency of the time series the same for both calendar systems. VedicDateTime [1] was used for all the conversions related to date and time-series objects in this comparative study.

5.2 Regressors of Diwali on industrial production of India

Diwali is the most important festival of India and its timing distorts the monthly time-series of industrial production heavily. Generally, Diwali is celebrated in the month of October according to the Gregorian Calendar but that is not fixed. However, as per the Vedic calendar, Diwali happens to be on the last day of the seventh *masa*, i.e., *Ashwin* and it is fixed unless two *tithis* appear on the same day. Depending on which month Diwali is celebrated, the industrial production index also changes. The standard software packages for seasonal adjustment [32, 33] (developed by the U.S. Census Bureau) or Tramo Seats [34] (developed by the Bank of Spain) have a built-in adjustment procedure for Easter holiday, but not for Diwali. However, all packages allow for the inclusion of user-defined variables. Seasonal is an interface to [32] wherein we can include Diwali dates. The impact of Diwali on industrial production in India is analyzed using the VedicDateTime package. First, Diwali dates are converted into VedicDateTime compatible date-time through the following steps.

- Step 1: View the Diwali holidays.
- Step 2: Convert Gregorian dates of Diwali holidays into Julian dates.
- Step 3: Select the desired location for the date conversion.
- Step 4: Convert Julian dates to Vedic calendar dates.

Next, a time series based on Gregorian dates is converted into Vedic calendar dates (explained in Sect. 5.1). For comparative analysis and better seasonal adjustment, the time series includes pre and post-festival industrial production. The seasonal package allows users to add user-defined regressors to remove seasonality from a time series. We have added both pre and post-Diwali production time series in the main seasonal adjustment. X-13ARIMA-SEATS is used to adjust for the festival seasonal component. Experimental analysis (presented in supplementary material: Doc3B) shows that the seasonal co-efficient shows a minor decline during pre and post-Diwali season across the time series. We have observed an unadjusted vs adjusted seasonal plot and it can be concluded that seasonal adjustment based on the Diwali season removes distortion from the time series (details are available in the supplementary material: Figure 5 in Doc 3B). The amount of distortion present in the series can be seen in Figure 6 in Doc 3B.

6 Conclusions

In conclusion, this paper introduces `VedicDateTime`, a pioneering open-source framework that implements the Vedic lunisolar calendar and facilitates conversions to and from Gregorian dates. This innovative tool presents a unique avenue for researchers to delve into time-series analyses from a fresh perspective, particularly in contexts influenced by both solar and lunar cycles. Significantly, this study serves to underscore the relevance and potential of alternative calendar systems in fields where the Gregorian calendar has been the primary reference point. The Vedic calendar, due to its lunisolar nature, offers an untapped wellspring of insights for research fields influenced by both celestial bodies.

Our explorations took the form of two case studies focusing on the economic outputs of India. The first study utilized the `VedicDateTime` package to investigate calendar seasonality's effects on economic output, while the second examined the impact of the Diwali festival on India's industrial production. In both scenarios, the `VedicDateTime` package emerged as a valuable tool, providing new insights and reinforcing the need for a nuanced understanding of calendar systems and their impacts. These case studies demonstrate that `VedicDateTime` offers practical solutions for modelling and understanding the complexity of time-series data in contexts where multiple calendar systems and cultural factors come into play. It makes the seasonal decomposition and adjustment process more comprehensive and culturally sensitive, thereby allowing for more accurate forecasting and analysis. As we navigate an increasingly globalized world, it's critical that our analytical tools evolve to reflect the diverse cultural, regional, and temporal complexities at play. `VedicDateTime` represents a significant stride in this direction, offering an innovative tool for researchers keen to incorporate these considerations into their work.

7 Limitations and future scopes

Future iterations of the `VedicDateTime` package will build on this foundation, with plans to incorporate a visualization feature that will allow for enhanced user interaction and interpretation of data. As it stands, `VedicDateTime` opens up an exciting new chapter in time-series analysis, bridging the gap between different calendar systems and providing fresh perspectives in the process.

At present, the `VedicDateTime` R package, while offering robust functionality for date conversions and time-series analysis, does not incorporate built-in tools for data visualization within the Vedic calendar framework. This limits the user's ability to directly and visually interact with and analyze their data in a Vedic calendar context. As part of our future roadmap, we intend to surmount this limitation by infusing the package with a dynamic visualization feature. This development is aimed at enabling users to graphically interact with the Vedic calendar and its corresponding Gregorian dates/times within a single, intuitive interface. This visual augmentation is projected to provide a holistic and tangible representation of the data. It will not only enhance the understanding of the Vedic time series but will also elucidate trends and patterns which may otherwise remain obscured in the raw data. By doing so, we aim to further augment the potential and usability of the `VedicDateTime` package in time-series analysis.

Another future scope of work is to add native R datetime object support for `VedicDateTime` library while dealing with calendar conversation which will provide generic class support (S4 class) to manipulate `VedicDateTime` objects as native R datetime objects which can include operation on such objects such as addition, time-delta and time-zone awareness which is currently missing.

Current code v0.1.7

Table 5 Code metadata

No.	Code metadata description	Details
C1	Current code version	v0.1.4
C2	Permanent link to code/repository used for this code version	https://github.com/neerajdhanraj/VedicDateTime
C3	Permanent link to Reproducible Capsule	https://cran.r-project.org/package=VedicDateTime
C4	Legal Code License	GNU General Public License v3.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	R programming language
C7	In views	TimeSeries [36]
C8	If available Link to developer documentation/manual	https://www.neerajbokde.in/viggnette/2022-09-05-VedicDateTime
C9	Support email for questions	*neerajdhanraj@qgg.au.dk*

Table 5 continued

No.	Code metadata description	Details
C10	Logo	VedicDateTime

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11042-023-16553-w.

Acknowledgements The authors would like to thank Google Inc. for its support and funds for the project through Google Summer of Code - 2022.

Funding Open access funding provided by Royal Danish Library, Aarhus University Library.

Data Availability The `VedicDateTime` R package utilized in this study is openly accessible under the General Public License version 3 (GPL-3). The complete source code can be freely downloaded and utilized for research purposes. The datasets employed for the case studies and analyses in this article are also publicly accessible and free to use under their respective open-source licenses. Detailed instructions for accessing the package and the datasets are included in the Supplementary Material. We strongly encourage the scientific community to use and further develop this tool, promoting transparency and reproducibility in research.

Software and hardware dependancy The `VedicDateTime` R package was developed using R Studio (Version: 2023.06.1+524) as the integrated development environment (IDE). The package is designed to work with R (Version 3.1.0 or higher) and draws upon several other R libraries to function optimally. This includes the `swephR` library, which is critical to the core functionality of the package and is listed under Imports. We have also utilized several other libraries such as `knitr`, `rmarkdown`, `testthat` (Version 3.0.0 or higher), `qpdf`, `formatR`, `spelling`, and `tinytex`', primarily for package documentation, testing, and code formatting purposes. These libraries are listed under Suggests, indicating they are not strictly necessary for the package's primary functions but are beneficial for extended functionalities.
The `VedicDateTime` R package has been successfully tested and validated across various operating systems, including diverse flavors of Linux, Windows, and macOS. Its compatibility has been thoroughly examined using different versions and configurations, showcasing its robustness and wide applicability [35]. The package has demonstrated consistent performance and compatibility, thereby affirming its potential to be a reliable tool for data analysis in different computational environments.

Declarations

Competing Interests The authors declare they have no competing interests.

Financial Interests The authors declare they have no financial interests.

References

1. Bokde ND, Patil PK, Sengupta S, Lorenzo AEF (2022) VedicDateTime: Vedic Calendar System, r package version 0.1.1. https://CRAN.R-project.org/package=VedicDateTime
2. Griffith RTH (2010) The hymns of the Atharva Veda
3. Tilak LB (1955) The Orion or Researches into the Antiquity of the Vedas
4. Chakrabarti V (1999) Indian architectural theory and practice: Contemporary uses of vastu Vidya. Routledge Curzon
5. Reingold EM, Dershowitz N (2018) Calendrical calculations: The ultimate edition. Cambridge University Press
6. Kumar A (2019) Ancient Hindu science: Its transmission and impact on world 420 cultures. Morgan & Claypool
7. List MS (2022) Pyluach: A Python package for manipulating Hebrew (Jewish) calendar dates and Hebrew-Gregorian conversion. Python Software Foundation. https://pypi.org/project/pyluach/
8. Yen LW (2013) lunisolar: Converting Gregorian and Solar dates to Kurdish date. Python Software Foundation. https://pypi.org/project/lunisolar/
9. Heris D (2020) pyroj: Converting Gregorian and Solar dates to Kurdish date. Python Software Foundation. https://pypi.org/project/pyroj/
10. Tripathi A (2020) sanatantime: Python module to convert christian system time to vedic sanatan time. Python Software Foundation. https://pypi.org/project/sanatantime/
11. Jalilian A (2021) jalcal: Conversion Between Jalali (Persian or Solar Hijri) and Gregorian Calendar Dates, r package version 0.1.0. https://CRAN.R-project.org/package=jalcal
12. indenkun (2021) jcalendar: Interconversion between the japanese calendar system and the western calendar. https://github.com/indenkun/jcalendaR
13. Callan D (2021) ethiopiandate: Convert between gregorian and ethiopian calendars. https://github.com/d-callan/ethiopianDate
14. Ruiz E (2021) maya: Converts between mayan and gregorian calendars. https://github.com/edgararuiz-zz/maya
15. Van Wijk JJ, Van Selow ER (1999) Cluster and calendar based visualization of time series data. In: Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis' 99), IEEE, pp. 4–9
16. Cleveland WS, Devlin SJ (1982) Calendar effects in monthly time series: modeling and adjustment. J Am Stat Assoc 77(379):520–528
17. Ramana BV (1992) Astrology in Predicting Weather and Earthquakes
18. Dorminey B (2009) Without the moon, would there be life on earth. Sci Am 21
19. Bokde ND (2023) Natural time-series analysis and vedic hindu calendar system. arXiv preprint arXiv:2111.03441
20. Gangooly P (1989) The Surya siddhanta the Surya siddhanta: A text-book of Hindu-astronomy
21. Basham AL (1974) The wonder that was India. Fontana Press
22. Ramakumar KL (2023) Panchangam calculations
23. De Livera AM, Hyndman RJ, Snyder RD (2011) Forecasting time series 460 with complex seasonal patterns using exponential smoothing. J Am Stat Assoc 106(496):1513–1527
24. Shiskin J, Young A, Musgrave J, of the Census USB (1965) The X-11 Variant of the Census Method II Seasonal Adjustment Program. Technical paper, U.S. Government Printing Office. https://books.google.ae/books?id=BFIfiGmatUoC
25. Maravall A, Sánchez FJ (2000) An application of tramo-seats; model selection and out-of-sample performance. the swiss cpi series. In: COMPSTAT: Proceedings in Computational Statistics 14th Symposium held in Utrecht, The Netherlands. Springer, pp. 121–130

26. Lin J-L, Liu T-S (2023) Modeling Lunar Calendar Holiday Effects in Taiwan. https://www.census.gov/library/working-papers/2002/adrm/lin-01.html
27. Reingold EM, Dershowitz N (2018) Calendrical Calculations: The Ultimate Edition. Cambridge University Press
28. Bhattacharya R, Pandey R, Patnaik I, Shah A (2016) Seasonal adjustment of indian macroeconomic time-series. Tech rep
29. Sanyal A, Mitra P, McElroy TS, Roy A (2017) Holiday effects in indian manufacturing series. Statistics 04
30. Kumar H, Dawar M (2017) Seasonality in the indian stock markets: A study of calendar effects. MUDRA: Journal of Finance and Accounting 4(1):1–19
31. Sax C, Eddelbuettel D (2023) Seasonal adjustment by x-13arima-seats in r 87. https://doi.org/10.18637/jss.v087.i11
32. Bureau UC (2013) X-13ARIMA-SEATS Seasonal Adjustment Program. US Census Bureau. https://www.census.gov/data/software/x13as.html
33. Sax C, Eddelbuettel D (2018) Seasonal adjustment by x–13arima-seats in r. J Stat Softw 87:1–17
34. Maravall A, Sánchez FJ (2000) An application of TRAMO-SEATS; model selection and out-of-sample performance. The Swiss CPI series, Physica-Verlag HD, pp. 121–130. https://doi.org/10.1007/978-3-642-57678-2_11. http://dx.doi.org/10.1007/978-3-642-57678-2_11
35. CRAN (2023) CRAN Package Check Results for Package VedicDateTime. https://cran.r-project.org/web/checks/check_results_VedicDateTime.html
36. Hyndman RJ, Killick R (2023) CRAN Task View: Time Series Analysis. Version 2023-06-27. https://CRAN.R-project.org/view=TimeSeries