Overview of the vietnameseConverter package

Juergen Niedballa (niedballa@izw-berlin.de)

2021-10-15

Contents

Introduction

This package helps you read and use data from Vietnamese sources in R. Such data often use Vietnamese legacy character encodings such as TCVN (Vietnam Standards / Tieu chuan Viet Nam) and are still in use today. Such data are not read correctly in R (which doesn't seem to support these Vietnamese encodings). To correct this problem and make such data available in R, this package converts data in legacy Vietnamese encodings to the correct Unicode characters. The main function is `decodeVN`.

The package currently supports conversion between the following encodings:

- Unicode (UTF-8)
- TCVN3
- VPS
- VISCII

It converts between any of these encodings, but most commonly one would want to convert legacy encodings to their correct Unicode characters. The

package supports character vectors and data frames (the character columns thereof).

Background

For those interested here is some background on character sets and encodings.

A *character set* is a collection of characters used in languages.

A *coded character set* assigns a unique number to each character in a character set. The unique values in a coded character sets are known as *code points* (e.g. "U+0041 " corresponcs to capital letter A in Unicode).

Unicode (also known as Universal Coded Character Set) is a modern coded character set that contains over 100,000 characters from most writing systems of the world.

Character encodings are the method by which a coded character set is converted to binary. *UTF-8* is a commonly used encoding for Unicode characters (UTF stands for Unicode Transformation Format).

Historically, the Vietnamese language could not be represented well using 128 characters in the ASCII standard. Hence, several character encodings were developed for the Vietnamese language with the aim of representing Vietnamese characters while fitting into 1 byte (=8-bit, allowing up to 255 characters). To achieve that, some some code points were reassigned and differ from today's standards like Unicode. Thus, when reading data that use those Vietnamese encodings on systems that assume e.g. UTF-8 encoding (Unicode), we get gibberish text (also known as mojibake).

Let's take a Vietnamese string that is supposed to be:

"Quảng Trị, An Đôn, Thừa Thiên Huế"

If it is encoded using a legacy Vietnamese encoding, it might displayed as something like:

"Qu¶ng TrÞ, An §«n, Thõa Thiªn HuÕ"

"Quäng TrÎ, An Ðôn, ThØa Thiên Hu‰"

"Quäng Tr¸, An Ðôn, Th×a Thiên Huª"

when it is read in R or other software (depending on the specific encoding of the data).

These days, almost all Vietnamese computer systems use Unicode and there is no need to use the legacy encodings anymore. Nevertheless, historic data may still be encoded in these legacy encodings and require conversion.

This package fixes this problem by converting the garbled strings back to their original (with or without the diacritcs). When saving the output in R, it will use standard encodings and data will read correctly in the future.

You can check the output from this package using e.g. this website http://www.enderminh.com/minh/vnconversions.aspx.

If you want to know more about the technicalities of encodings and character sets, see https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/.

Installation

The package can be installed from CRAN:

```r
install.packages("vietnameseConverter")
```

and loaded into R via:

```r
library(vietnameseConverter)
```

Example applications

Disclaimer: Printing Unicode characters from R function output in package vignettes is difficult. Often R will print the Unicode escape characters (the code point) instead of the actual characters, e.g. <U+1EBF> instead of "ế".

This is a limitation in vignettes. I'll try to circumvent it in this vignette as much as possible, e.g. by using DT::data.table() and by copy/pasting the correct console output below the output with Unicode characters. Even though it doesn't print nicely in this vignette, the converted data are correct (as you can see if you use data.table(), View(), or print vectors in the R console.

Run the code from this vignette in your R console to see the actual output, or try the examples in `?decodeVNN`

Vectors

Let’s consider a simple vector with garbled characters:

```
string_garbled <- c("Qu¶ng TrÞ", "An §«n", "Thõa Thiªn HuÕ")
string_garbled
```

```
## [1] "Qu¶ng TrÞ"      "An §«n"         "Thõa Thiªn HuÕ"
```

It can be fixed using the `decodeVN` function:

```
tmp <- decodeVN(string_garbled)
tmp
```

```
## [1] "Qu<U+1EA3>ng Tr<U+1ECB>" "An Đôn"         "Th<U+1EEB>a Thiên Hu<U+1EBF>"
```

In your R console, the Unicode characters will be printed correctly as:

```
## [1] "Quảng Trị"      "An Đôn"         "Thừa Thiên Huế"
```

By default, the output contains character with diacritics (accents). We can also get output without the diacritics (plain ASCII letters) by setting argument `diacritics = FALSE`:

```
decodeVN(string_garbled, diacritics = FALSE)
```

```
## [1] "Quang Tri"      "An Don"         "Thua Thien Hue"
```

By default, decodeVN() attempts to convert TCVN3 to Unicode characte (as shown in this example)r. If your data are in different encodings, set argument `from`.

For example, a string in VPS encoding is converted via:

```
string_garbled_vps <- c("Quäng TrÎ",  "An ñôn", "ThØa Thiên Hu‰")
decodeVN(string_garbled_vps, from = "VPS")
```

```
## [1] "Qu<U+1EA3>ng Tr<U+1ECB>" "An Đôn"         "Th<U+1EEB>a Thiên Hu<U+1EBF>"
```

Which in your R cosole will be

```
## [1] "Quảng Trị"      "An Đôn"         "Thừa Thiên Huế"
```

There is also an argument `to`, which by default is set to `to = "Unicode"`. If you want output that simulates data encoded in one of the supported Vietnamese encodings, set argument `to` accordingly, e.g.

```
string_tcvn_to_vps <- decodeVN(x = string_garbled, from = "TCVN3", to = "VPS")
string_tcvn_to_vps
```

```
## [1] "Quäng TrÎ"      "An ñôn"         "ThØa Thiên Hu‰"
```

This string is now garbled the VPS-way. It can still be converted to Unicode:

```
decodeVN(string_tcvn_to_vps, from = "VPS")
```

```
## [1] "Qu<U+1EA3>ng Tr<U+1ECB>" "An Đôn"         "Th<U+1EEB>a Thiên Hu<U+1EBF>"
```

which is:

```
## [1] "Quảng Trị"      "An Đôn"         "Thừa Thiên Huế"
```

Data frames

We can also convert entire data frames at once. It will only convert character columns and leave the other columns untouched. If you have factor columns, convert them to character first.

The package contains a list containing several data frames which simulate the problems with different encodings. These data frames show what the problems with different encodings look like when the data are loaded in an environment that assumes UTF-8 / native encoding). The data frame is based on the table of provinces in https://en.wikipedia.org/wiki/Provinces_of_Vietnam.

Loading the list of sample data.

```
data(vn_samples)
```

vn_samples is a list of four data frames. See `?vn_samples` for details.

The first item $Unicode shows the correct characters.

```
head(vn_samples$Unicode)
```

```
##           Province_city Administrative_center Area_km2 Density_perkm2 HDI_2012
## 1      B<U+1EAF>c Giang      B<U+1EAF>c Giang  3895.59            463    0.711
## 2 B<U+1EAF>c K<U+1EA1>n B<U+1EAF>c K<U+1EA1>n  4859.96             65    0.685
## 3       Cao B<U+1EB1>ng       Cao B<U+1EB1>ng  6700.26             79    0.653
## 4             Hà Giang             Hà Giang  7929.48            108    0.586
## 5       L<U+1EA1>ng Son       L<U+1EA1>ng Son  8310.09             94    0.707
## 6        Phú Th<U+1ECD>       Vi<U+1EC7>t Trì  3534.56            414    0.715
```

Note: when printing a data frame in the R colsole, the Unicode characters are not displayed properly, which is a limitation of the print method for data frames in R (it's the same issue as mentioned in the disclaimer above). The characters are correct though and show correctly when printing the columns as vectors:

```
head(vn_samples$Unicode$Province_city)
```

```
## [1] "B<U+1EAF>c Giang" "B<U+1EAF>c K<U+1EA1>n" "Cao B<U+1EB1>ng" "Hà Giang"  "L<U+1EA1>ng Son" "Phú T
```

Which again is not formatted nicely in the vignette (but would be formatted correctly in your console). The string is:

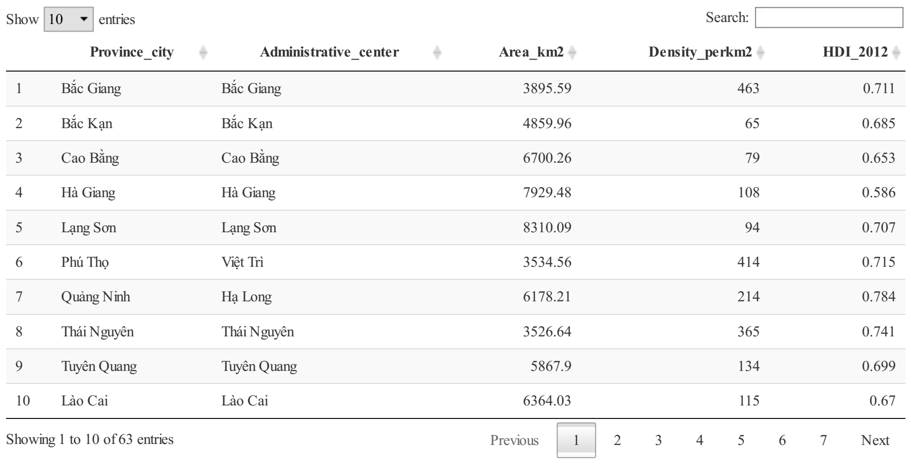
```
## [1] "Bắc Giang" "Bắc Kạn"   "Cao Bằng"  "Hà Giang"  "Lạng Sơn"  "Phú Thọ"
```

It also shows correctly when using `View` (not shown here, it works in interactive sessions):

```
View(vn_samples$Unicode)
```

It is also shown correctly when using `DT::datatable`:

```
DT::datatable(vn_samples$Unicode)
```

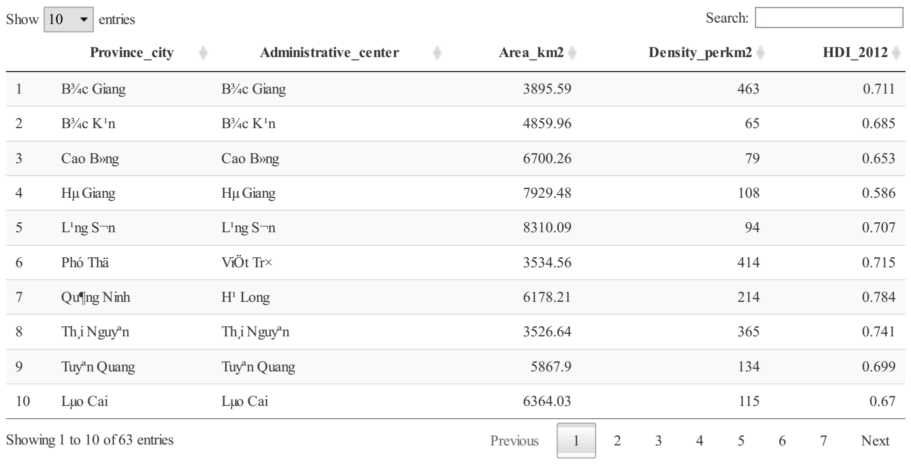

Show 10 entries Search:

	Province_city	Administrative_center	Area_km2	Density_perkm2	HDI_2012
1	Bắc Giang	Bắc Giang	3895.59	463	0.711
2	Bắc Kạn	Bắc Kạn	4859.96	65	0.685
3	Cao Bằng	Cao Bằng	6700.26	79	0.653
4	Hà Giang	Hà Giang	7929.48	108	0.586
5	Lạng Sơn	Lạng Sơn	8310.09	94	0.707
6	Phú Thọ	Việt Trì	3534.56	414	0.715
7	Quảng Ninh	Hạ Long	6178.21	214	0.784
8	Thái Nguyên	Thái Nguyên	3526.64	365	0.741
9	Tuyên Quang	Tuyên Quang	5867.9	134	0.699
10	Lào Cai	Lào Cai	6364.03	115	0.67

Showing 1 to 10 of 63 entries

The other data frames in `vn_samples` show garbled text from several Vietnamese encodings. Here's the example in TCVN3 encoding:

```
DT::datatable(vn_samples$TCVN3)
```


Show 10 entries Search:

	Province_city	Administrative_center	Area_km2	Density_perkm2	HDI_2012
1	B¾c Giang	B¾c Giang	3895.59	463	0.711
2	B¾c K¹n	B¾c K¹n	4859.96	65	0.685
3	Cao B»ng	Cao B»ng	6700.26	79	0.653
4	Hµ Giang	Hµ Giang	7929.48	108	0.586
5	L¹ng S¬n	L¹ng S¬n	8310.09	94	0.707
6	Phó Thä	ViÖt Tr×	3534.56	414	0.715
7	Qu¶ng Ninh	H¹ Long	6178.21	214	0.784
8	Th¸i Nguyªn	Th¸i Nguyªn	3526.64	365	0.741
9	Tuyªn Quang	Tuyªn Quang	5867.9	134	0.699
10	Lµo Cai	Lµo Cai	6364.03	115	0.67

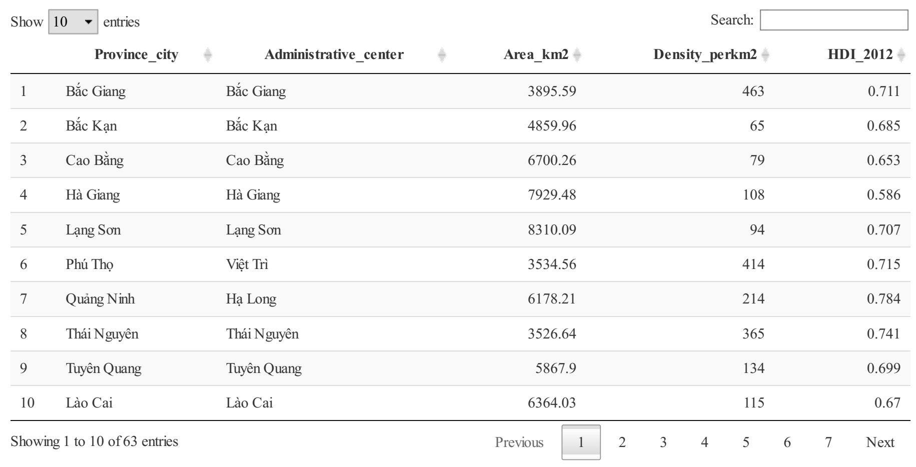
Showing 1 to 10 of 63 entries

One can easily convert the entire data frame:

```r
# take data frames out of list for easier readability
df_unicode <- vn_samples$Unicode
df_tcvn3 <- vn_samples$TCVN3

# conversion from TCVN3 to Unicode (default)
df_tcvn3_converted <- decodeVN(df_tcvn3)

# print output
DT::datatable(df_tcvn3_converted)
```



Show 10 entries Search:

	Province_city	Administrative_center	Area_km2	Density_perkm2	HDI_2012
1	Bắc Giang	Bắc Giang	3895.59	463	0.711
2	Bắc Kạn	Bắc Kạn	4859.96	65	0.685
3	Cao Bằng	Cao Bằng	6700.26	79	0.653
4	Hà Giang	Hà Giang	7929.48	108	0.586
5	Lạng Sơn	Lạng Sơn	8310.09	94	0.707
6	Phú Thọ	Việt Trì	3534.56	414	0.715
7	Quảng Ninh	Hạ Long	6178.21	214	0.784
8	Thái Nguyên	Thái Nguyên	3526.64	365	0.741
9	Tuyên Quang	Tuyên Quang	5867.9	134	0.699
10	Lào Cai	Lào Cai	6364.03	115	0.67

Showing 1 to 10 of 63 entries

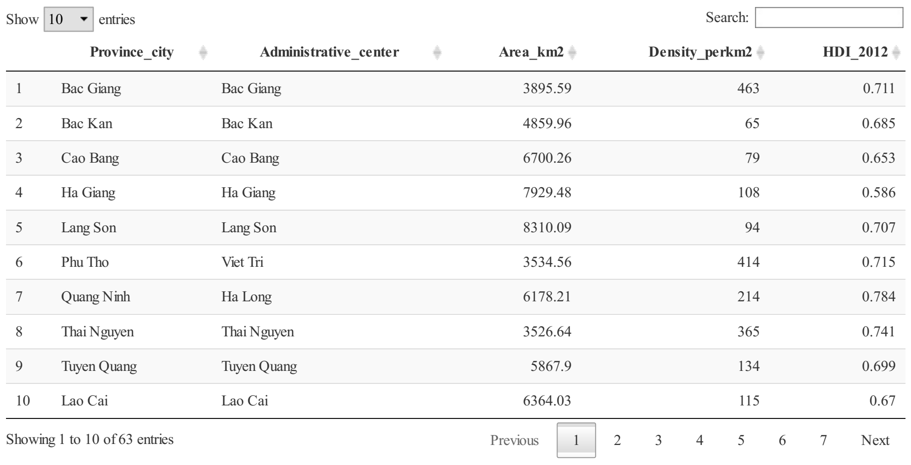
After conversion it is identical to the original with Unicode characters

```r
all.equal(df_unicode, df_tcvn3_converted)
```

```
## [1] TRUE
```

Again, we can choose to return characters without accents by setting `diacritics = FALSE`:

```r
df_tcvn3_converted2 <- decodeVN(df_tcvn3, diacritics = FALSE)
DT::datatable(df_tcvn3_converted2)
```


Show 10 entries Search:

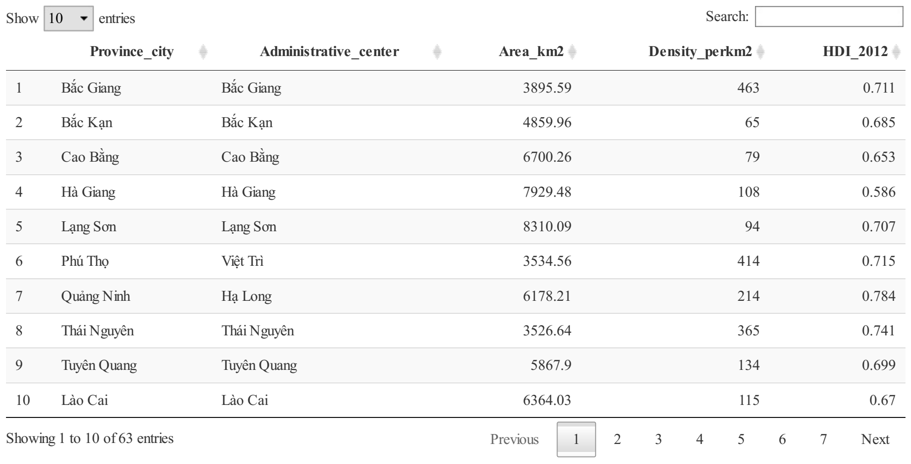
	Province_city	Administrative_center	Area_km2	Density_perkm2	HDI_2012
1	Bac Giang	Bac Giang	3895.59	463	0.711
2	Bac Kan	Bac Kan	4859.96	65	0.685
3	Cao Bang	Cao Bang	6700.26	79	0.653
4	Ha Giang	Ha Giang	7929.48	108	0.586
5	Lang Son	Lang Son	8310.09	94	0.707
6	Phu Tho	Viet Tri	3534.56	414	0.715
7	Quang Ninh	Ha Long	6178.21	214	0.784
8	Thai Nguyen	Thai Nguyen	3526.64	365	0.741
9	Tuyen Quang	Tuyen Quang	5867.9	134	0.699
10	Lao Cai	Lao Cai	6364.03	115	0.67

Showing 1 to 10 of 63 entries Previous 1 2 3 4 5 6 7 Next

We can also use the `from` and `to` arguments for conversions between other encodings.

For example, to convert VISCII-encoded data to Unicode, use:

```r
df_viscii <- vn_samples$VISCII
DT::datatable(decodeVN(df_viscii, from = "VISCII"))
```

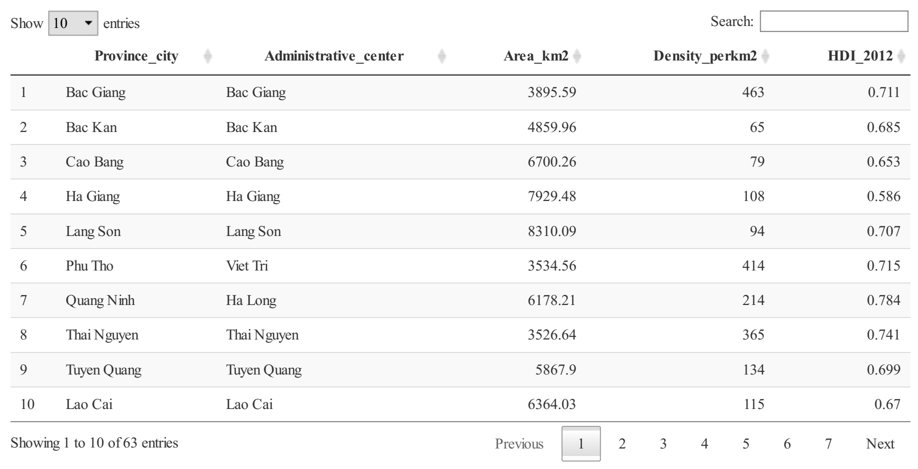

Show 10 entries Search:

	Province_city	Administrative_center	Area_km2	Density_perkm2	HDI_2012
1	Bắc Giang	Bắc Giang	3895.59	463	0.711
2	Bắc Kạn	Bắc Kạn	4859.96	65	0.685
3	Cao Bằng	Cao Bằng	6700.26	79	0.653
4	Hà Giang	Hà Giang	7929.48	108	0.586
5	Lạng Sơn	Lạng Sơn	8310.09	94	0.707
6	Phú Thọ	Việt Trì	3534.56	414	0.715
7	Quảng Ninh	Hạ Long	6178.21	214	0.784
8	Thái Nguyên	Thái Nguyên	3526.64	365	0.741
9	Tuyên Quang	Tuyên Quang	5867.9	134	0.699
10	Lào Cai	Lào Cai	6364.03	115	0.67

Showing 1 to 10 of 63 entries Previous 1 2 3 4 5 6 7 Next

The same thing without diacritics:

```
DT::datatable(decodeVN(df_viscii, from = "VISCII", diacritics = FALSE))
```



Show 10 entries

Search:

	Province_city	Administrative_center	Area_km2	Density_perkm2	HDI_2012
1	Bac Giang	Bac Giang	3895.59	463	0.711
2	Bac Kan	Bac Kan	4859.96	65	0.685
3	Cao Bang	Cao Bang	6700.26	79	0.653
4	Ha Giang	Ha Giang	7929.48	108	0.586
5	Lang Son	Lang Son	8310.09	94	0.707
6	Phu Tho	Viet Tri	3534.56	414	0.715
7	Quang Ninh	Ha Long	6178.21	214	0.784
8	Thai Nguyen	Thai Nguyen	3526.64	365	0.741
9	Tuyen Quang	Tuyen Quang	5867.9	134	0.699
10	Lao Cai	Lao Cai	6364.03	115	0.67

Showing 1 to 10 of 63 entries

Previous 1 2 3 4 5 6 7 Next

Spatial data

If you are working with spatial data, it is easiest to temporarily convert them to non-spatial data (a regular data frame), run decodeVN(), and make the object spatial again.

For `sf` objects, the workflow can be as follows (data are not included, code is for illustration only, and converts from TCVN3 to Unicode).

```
library(sf)
sf_object <- st_read(...)                        # load sf data set
sf_object_geometry <- st_geometry(sf_object)     # temporarily store geometry column
sf_object_no_geom <- st_drop_geometry(sf_object)  # remove geometry column
sf_object_decoded <- decodeVN(sf_object_no_geom)        # run decodeVN
sf_object_decoded <- st_set_geometry(sf_object_decoded, sf_object_geometry)  # assign geometry column t
sf_object_decoded
```