A Minimal Book Example

Yihui Xie

2020-04-07

Contents

Chapter 1

Installation

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

# Development version from GitHub
# Install devtools if not previously installed.
# install.packages("devtools")
# If there are errors (converted from warning) during installation related to packages built unde
# they can be ignored by setting the environment variable R_REMOTES_NO_ERRORS_FROM_WARNINGS="true

# Sys.setenv(R_REMOTES_NO_ERRORS_FROM_WARNINGS="true")
# devtools::install_github("jameswcraig/tidyvpc")
```

Chapter 2

Introduction

When deriving a Visual Predictive check (VPC) you must:

- Have both observed and simulated datasets that include x & y variables, typically TIME & DV.
- Compute Prediction Intervals on Simulated versus Observed Data

When deriving a VPC you may want to:

- Stratify over variables in your model.
- Censor data below LLOQ.
- Perform prediction correction (pcVPC).

The tidyvpc package makes these steps fast and easy:

- By providing readable syntax using the `%>%` operator from magrittr.
- It uses efficient backend computation, taking advantage of `data.table` parallelization.
- By providing traditional binning methods and new binless methods using additive quantile regression and loess for pcVPC.
- By using ggplot2 graphics engine to visualize the results of the VPC.

This document introduces you to tidyvpc's set of tools, and shows you how to apply them to tidyvpcobj to derive VPC.

All of the tidyvpc functions take a tidyvpcobj as the first argument, with the exception of the first function `observed()` in the piping chain, which takes a `data.frame` or `data.table` of the observed dataset. Rather than forcing the user to either save intermediate objects or nest functions, tidyvpc provides the `%>%` operator from magrittr. The result from one step is then "piped" into the next step, with the final function in the piping chain always `vpcstats()`. You can use the pipe to rewrite multiple operations that you can read left-to-right, top-to-bottom (reading the pipe operator as "then").

2.1 Data

To explore the functionality of tidyvpc, we'll use an altered version of obs_data(`vpc::simple_data$obs`) & sim_data(`vpc::simple_data$sim`) from the vpc package. These datasets contains all necessary variables to explore the functionality of tidyvpc including:

- DV (y variable)
- TIME (x variable)
- NTIME (nominal time for binning on x-variable)
- GENDER (gender variable for stratification, "M", "F")
- STUDY (study for stratification, "Study A", "Study B")
- PRED (prediction variable for pcVPC)
- MDV (Missing DV)

```r
library(tidyvpc)
obs_data <- tidyvpc::obs_data
sim_data <- tidyvpc::sim_data
head(obs_data)
```

```
##   ID      TIME   DV AMT DOSE MDV NTIME GENDER   STUDY
## 1  1 0.0000000  0.0 150  150   1  0.00      M Study A
## 2  1 0.2157624 37.3   0  150   0  0.25      M Study A
## 3  1 0.4694366 62.2   0  150   0  0.50      M Study A
## 4  1 0.8271844 74.1   0  150   0  1.00      M Study A
## 5  1 1.7724895 75.1   0  150   0  1.50      M Study A
## 6  1 1.7142415 58.3   0  150   0  2.00      M Study A
```

2.1.1 Preprocessing data

First we'll need to subset our data by filtering `MDV == 0` which removes rows where both `DV == 0` & `TIME == 0`.

```r
obs_data <- as.data.table(obs_data)
sim_data <- as.data.table(sim_data)
obs_data <- obs_data[obs_data$MDV == 0,]
sim_data <- sim_data[sim_data$MDV == 0,]
```

Next we'll add the prediction variable from the first replicate of simulated data into our observed data.

```r
obs_data$PRED <- sim_data$PRED[sim_data$REP == 1]
```

Now that we have our data loaded in memory, proceed to the next chapter to learn about using the various functions in the `tidyvpc` package.

Chapter 3

Functions

3.1 observed()

The `observed()` function is always the first function used in the VPC piping chain and is used to specify the observed dataset and corresponding variables. There are three arguments that are required in order to use `observed`. The first argument is either a `data.frame` or `data.table`, the second argument is the name of x-variable in the observed data, and the third argument is the name of the y-variable. Note variable names should be unquoted.

```
vpc <- observed(obs_data, x = TIME, y = DV)
```

3.2 simulated()

The `simulated()` function is used to specify the simulated dataset and corresponding variables. There are two arguments that are required in order to use `simulated()`. Since the function is "piped" in after the `observed()` function, the first argument is the tidyvpcobj and should not be included, followed by the name of the simulated data, then the name of y-variable in the simulated data. Variable names should be unquoted and x-variable should not be included as it is recycled from the `observed()` function.

```
vpc <- observed(obs_data, x = TIME, y = DV) %>%
  simulated(sim_data, y = DV)
```

3.3 binning()

The `binning()` function provides the binning method to derive the vpc and should be inputted as a character string in the `bin` argument. Binning methods include: “ntile”, “pam”, “sd”, “equal”, “pretty”, “quantile”, “kmeans”, “jenks”, “centers”, “breaks”. Some methods such as “ntile” and “pam” will require you to specify the number of bins using the `nbins` argument i.e. `nbins = 9`.

If using `bin = "centers"` or `bin = "breaks` you must also provide the centers/breaks argument as numeric vector in the function i.e. `centers = c(1,3,5,7)`.

You can also bin directly on x-variable. If using this type of binning, the bin argument should be the unquoted variable name that you used in the `observed()` function i.e. `bin = NTIME` for the Nominal Time variable in the data.

Binning on x-variable, NTIME

```r
vpc <- observed(obs_data, x=TIME, y=DV) %>%
    simulated(sim_data, y=DV) %>%
    binning(bin = NTIME)
```

Binning with “ntile”

```r
vpc <- observed(obs_data, x = TIME, y = DV) %>%
  simulated(sim_data, y = DV) %>%
  binning(bin = "ntile", nbins = 9)
```

Binning with “breaks”

```r
vpc <- observed(obs_data, x = TIME, y = DV) %>%
  simulated(sim_data, y = DV) %>%
  binning(bin = "breaks", breaks = c(1,5,7,9,10))
```

3.4 binless()

Binless methods utilize additive quantile regression (AQR) in place of traditional binning. Use the `binless()` function instead of `binning()` to derive a binless VPC. By default, `binless()` performs AQR at the 5%, 50%, and 95% quantiles but you can change this using the `qpred` argument which takes a numeric vector of length 3 i.e. `qpred =  c(.1, .5, .9)` for the 10%, 50%, 90% quantiles.

The lambda smoothing parameters for each quantile are optimized by default with AIC as indicated by the `optimize = TRUE` argument, however, if you would