Visualising big data (in R)

Hadley Wickham

@hadleywickham

Chief Scientist, RStudio

June 2013

Motivation

Data

- Every US commercial domestic flight 2000-2011: ~76 million flights
- >100 variables. I'll focus on 4: delay, distance, flight time and speed.
- (Total database: ~11 Gb)

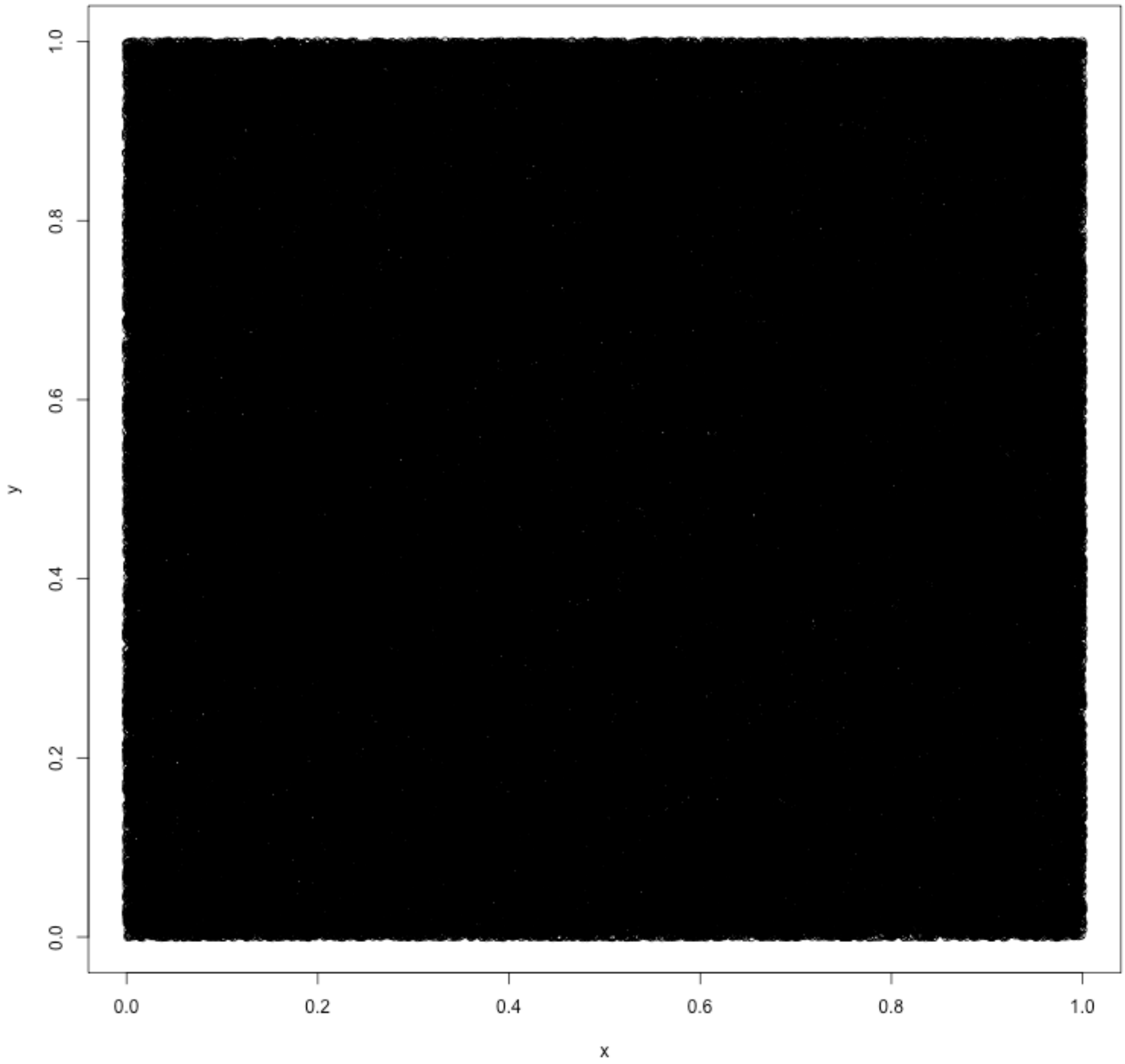
```
library(ggplot2)
library(bigvis)


# Can't use data frames :(
dist <- readRDS("dist.rds")
delay <- readRDS("delay.rds")
time <- readRDS("time.rds")
speed <- dist / time * 60


# There's always bad data
time[time < 0] <- NA
speed[speed < 0] <- NA
speed[speed > 761.2] <- NA
```

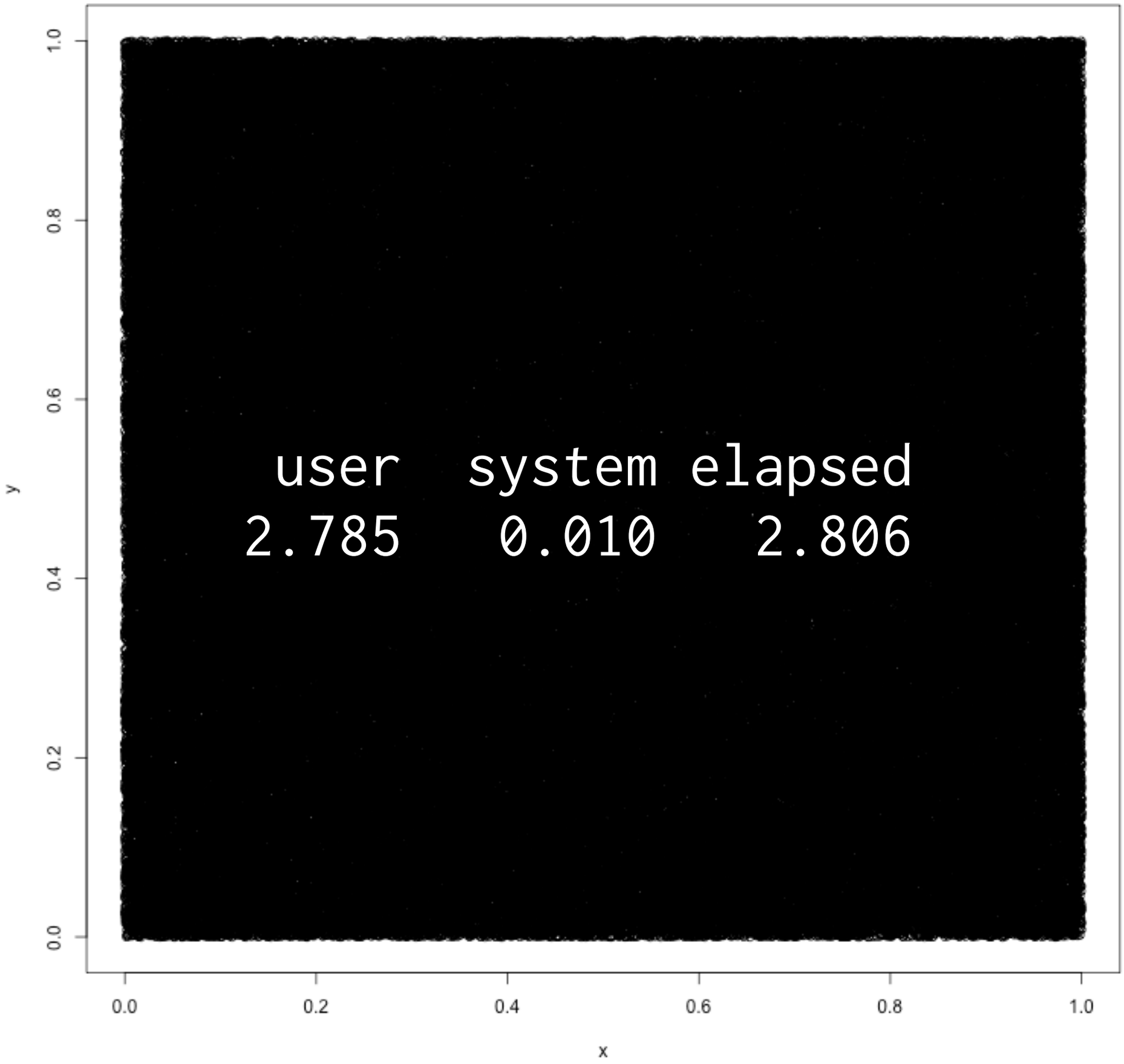
```
qplot(dist, speed, colour = delay) +
  scale_colour_gradient2()
```

One hour later...

```
qplot(dist, speed, colour = delay) +
  scale_colour_gradient2()
```

```
x <- runif(2e5)
y <- runif(2e5)
system.time(plot(x, y))
```


y
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x


user system elapsed
2.785 0.010 2.806
0.0
0.2
0.4
0.6
0.8
1.0
x
y

Motivating principles

- Support exploratory analysis (e.g. in R)
- Efficient
  - 1d: 3,000; 2d: 3,000,000
- Fast on commodity hardware
  - 100,000,000 in <5s
  - 10^8 obs = 0.8 Gb, ~20 vars in 16 Gb

Process

- Condense (bin & summarise)
- Smooth
- Visualise

Related work

- W. Härdle and D. Scott. *Smoothing in low and high dimensions by weighted averaging using rounded points*. Computational Statistics, 7:97–128, 1992.
- J. Fan and J. S. Marron. *Fast implementations of nonparametric curve estimators*. Journal of Computational and Graphical Statistics, 3 (1): 35–56, 1994.
- M. Wand. *Fast computation of multivariate kernel estimators*. Journal of Computational and Graphical Statistics, 3 (4):433–445, 1994.

Condense

Bin

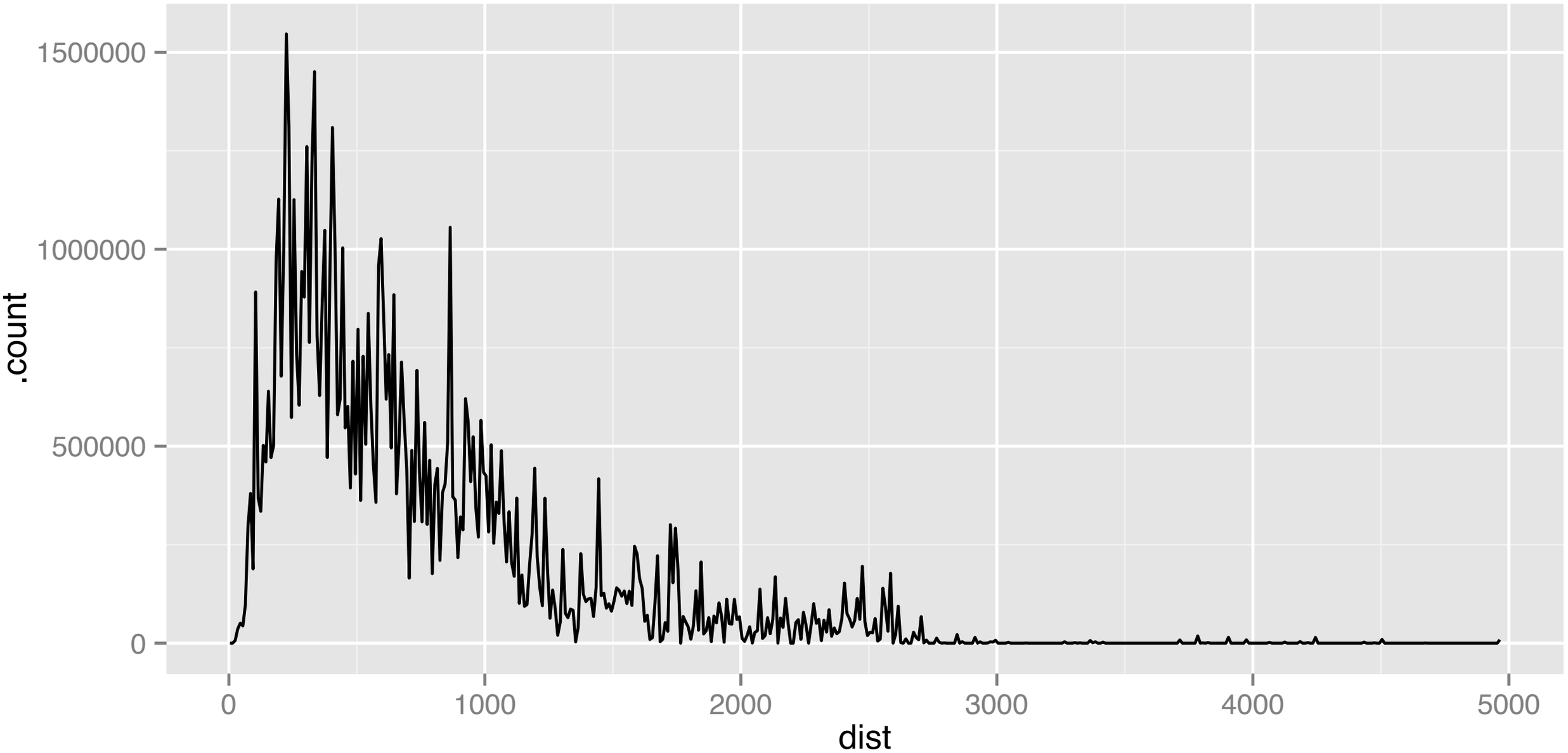
$$\left\lfloor \frac{x - \text{origin}}{\text{width}} \right\rfloor$$

Fixed bins

- V. fast to compute.
- Not obviously worse for density estimation
- No automatic bin width estimation: err on the side of too many & fix up later

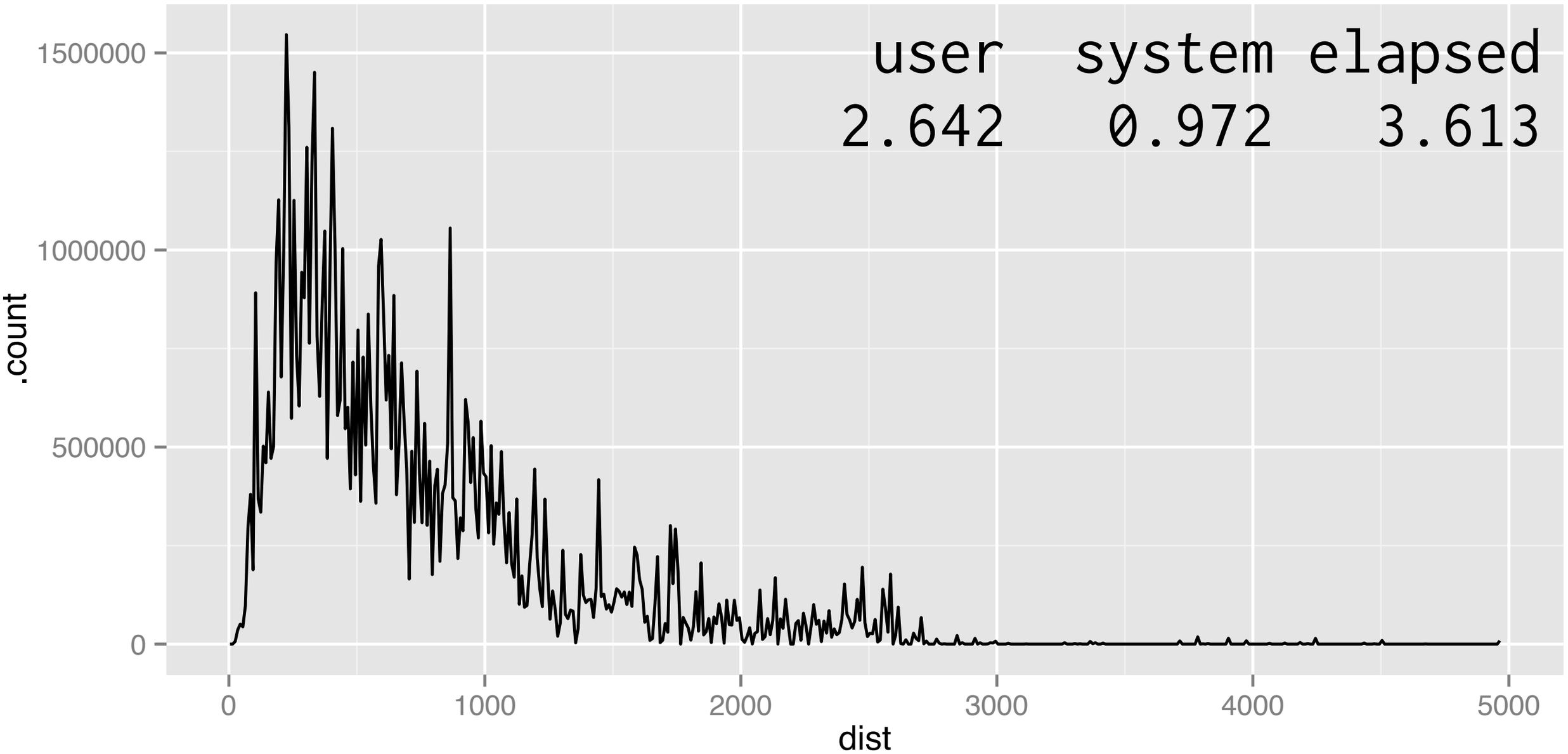
Summarise

Count	Histogram, KDE
Mean	Regression, Loess
Std. dev.	
Quantiles	Boxplots, Quantile regression smoothing

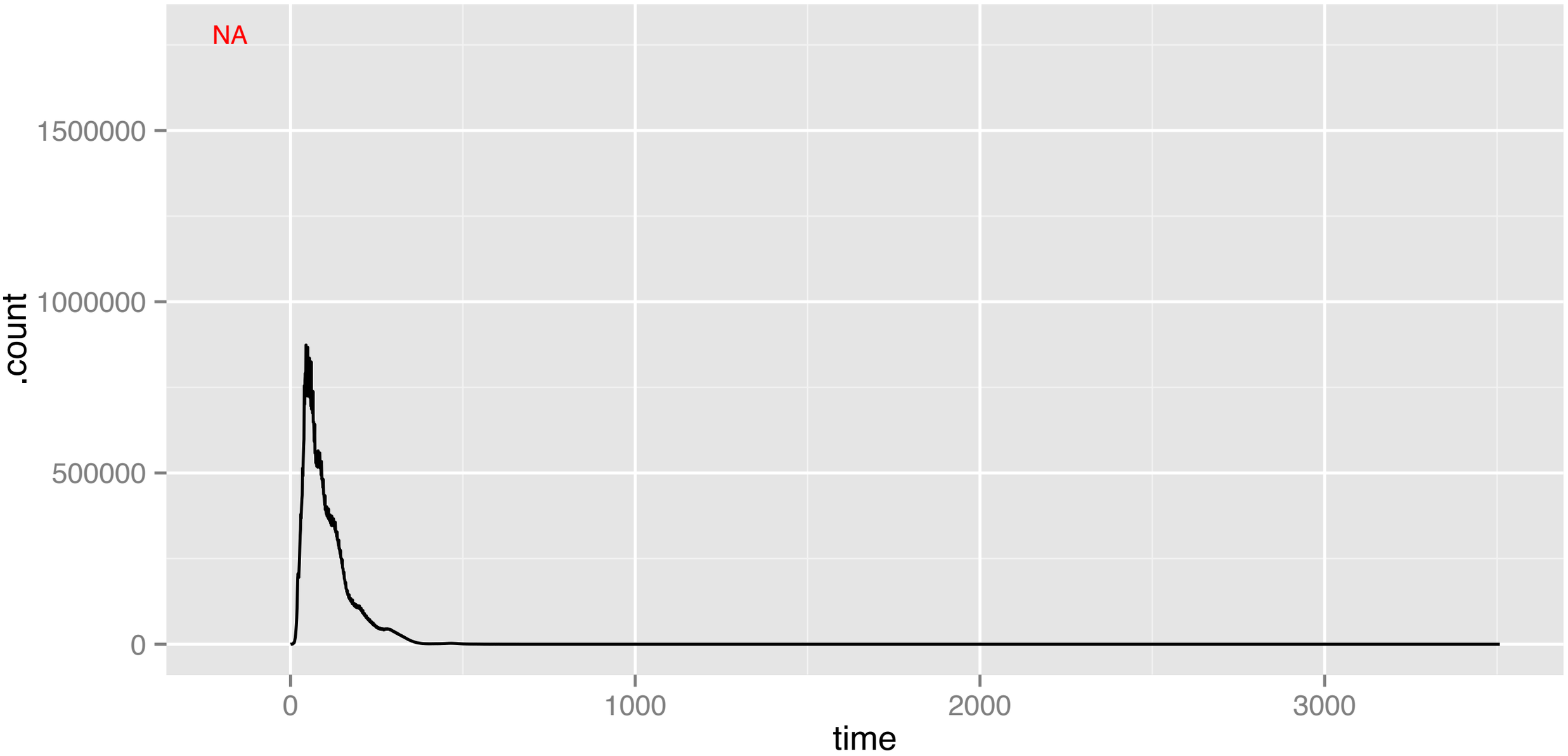



```
dist_s <- condense(bin(dist, 10))
autoplot(dist_s)
```

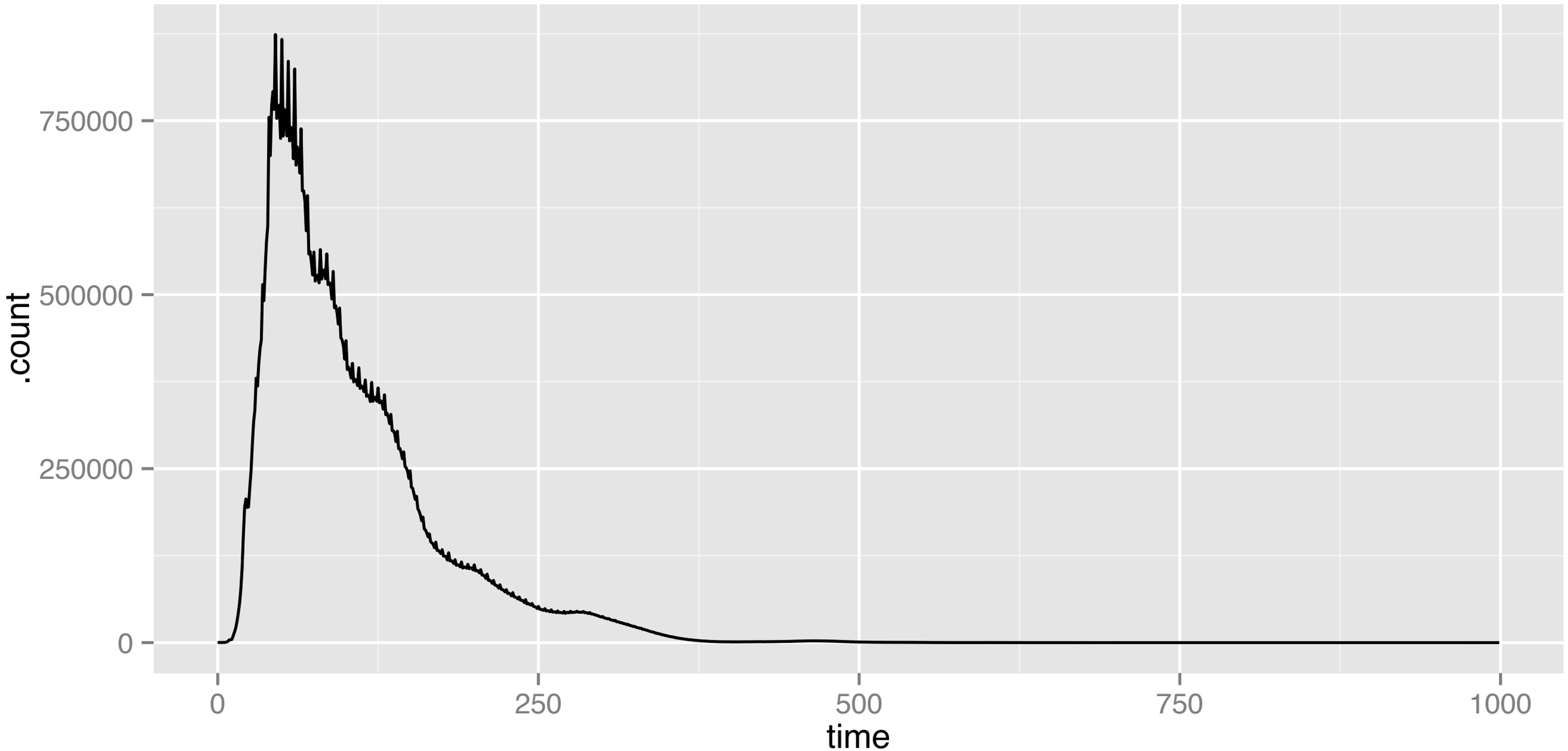
```
   user  system elapsed
  2.642   0.972   3.613
```



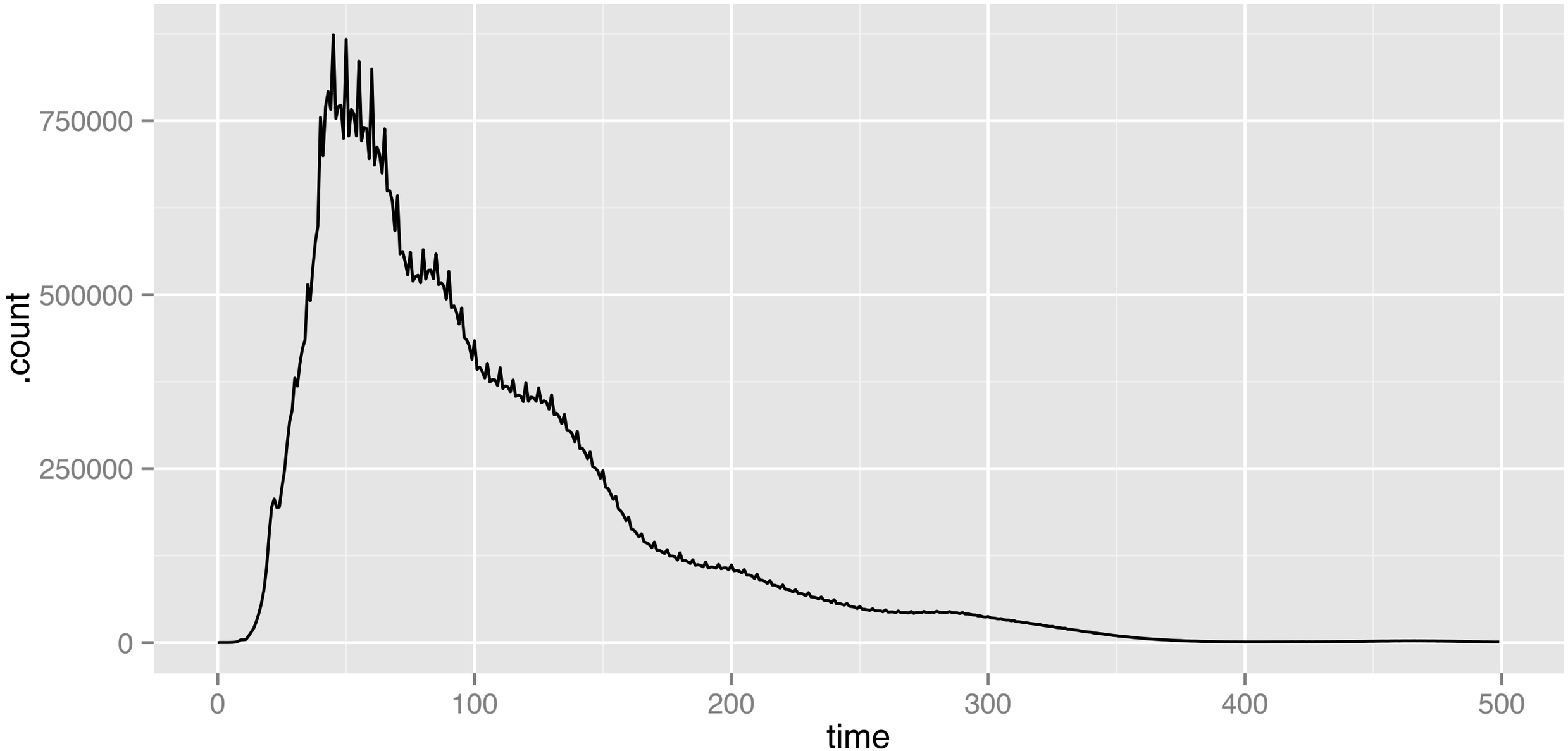

```
dist_s <- condense(bin(dist, 10))
autoplot(dist_s)
```



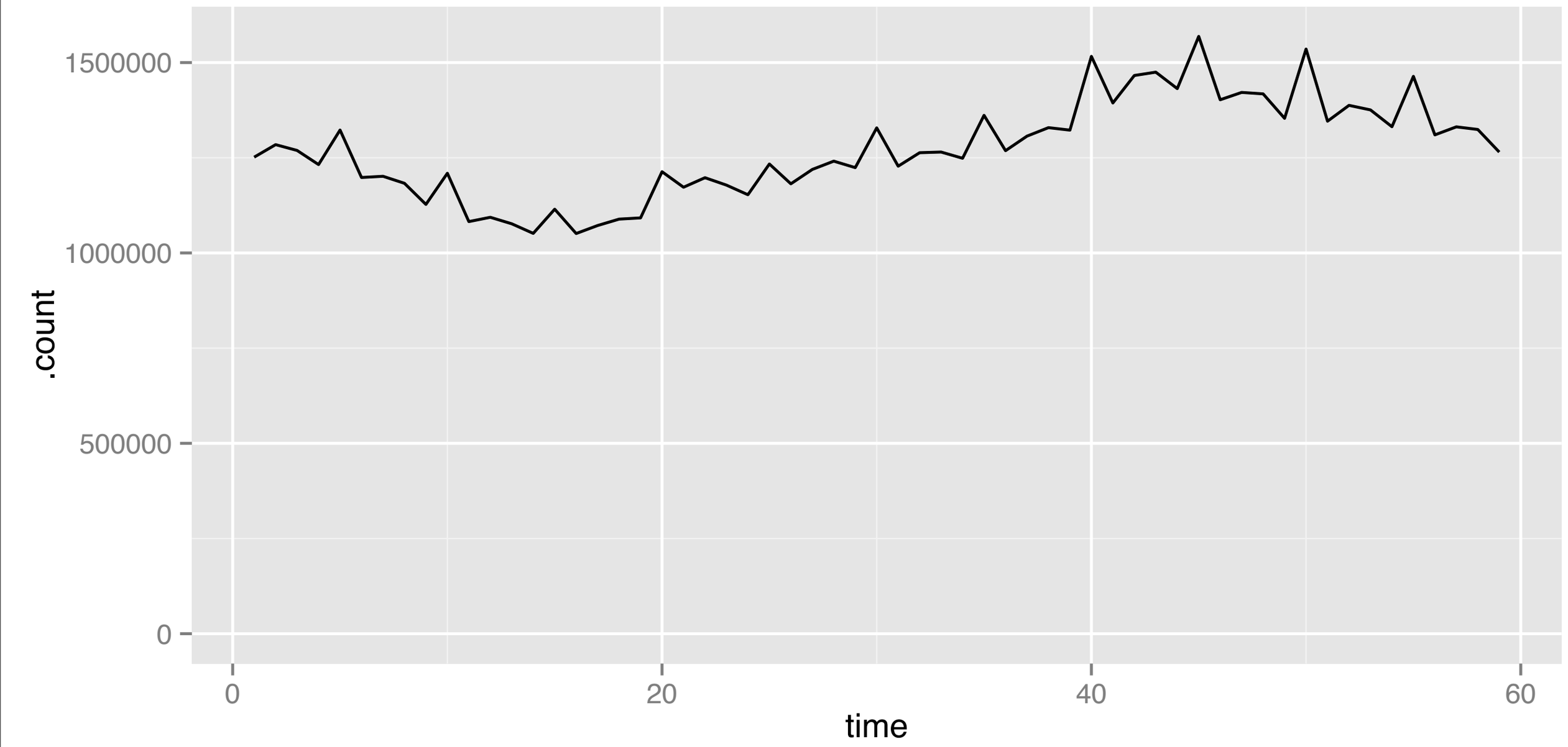

```
time_s <- condense(bin(time, 1))
autoplot(time_s)
```



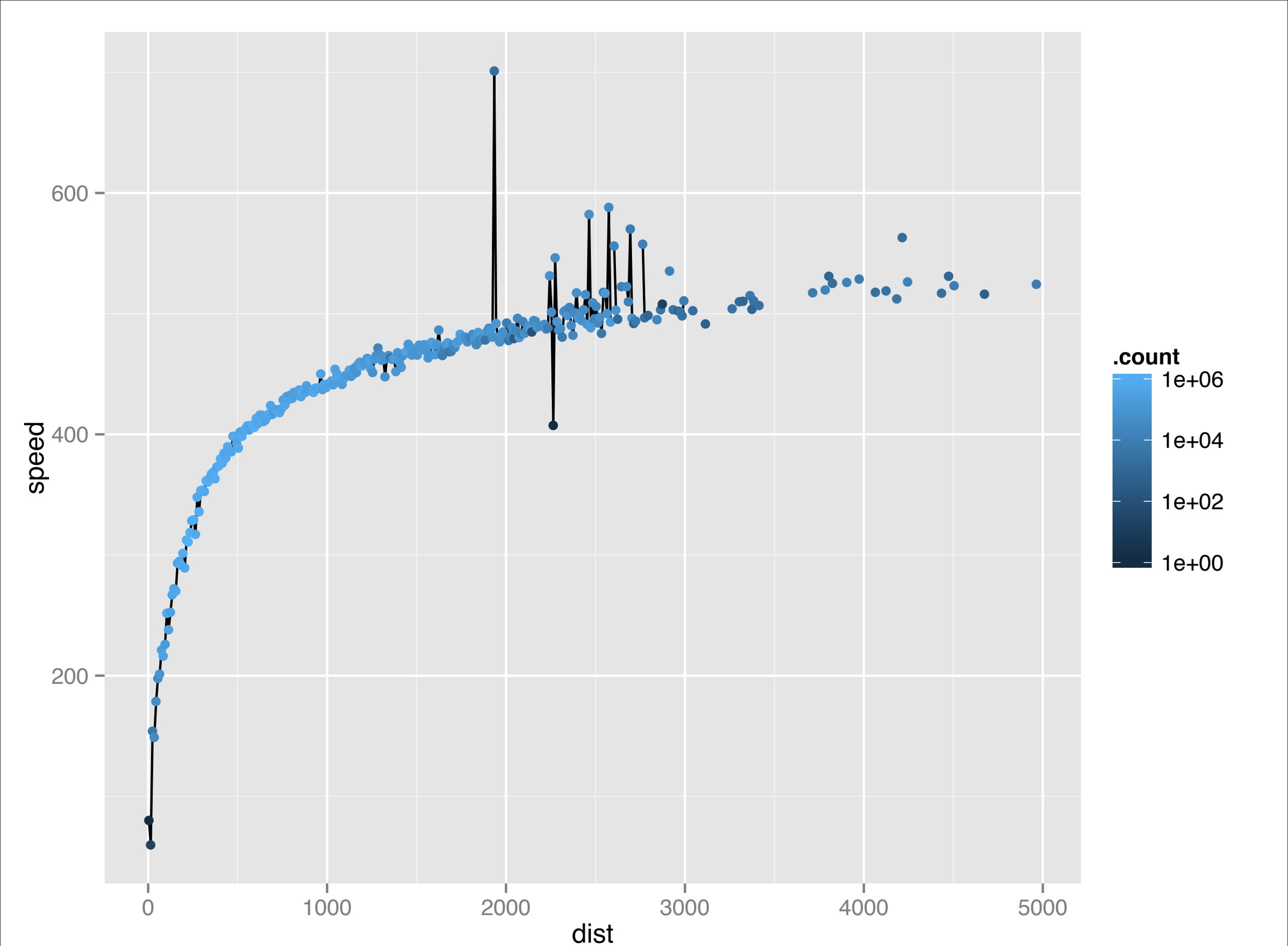

```
autoplot(time_s, na.rm = TRUE)
```

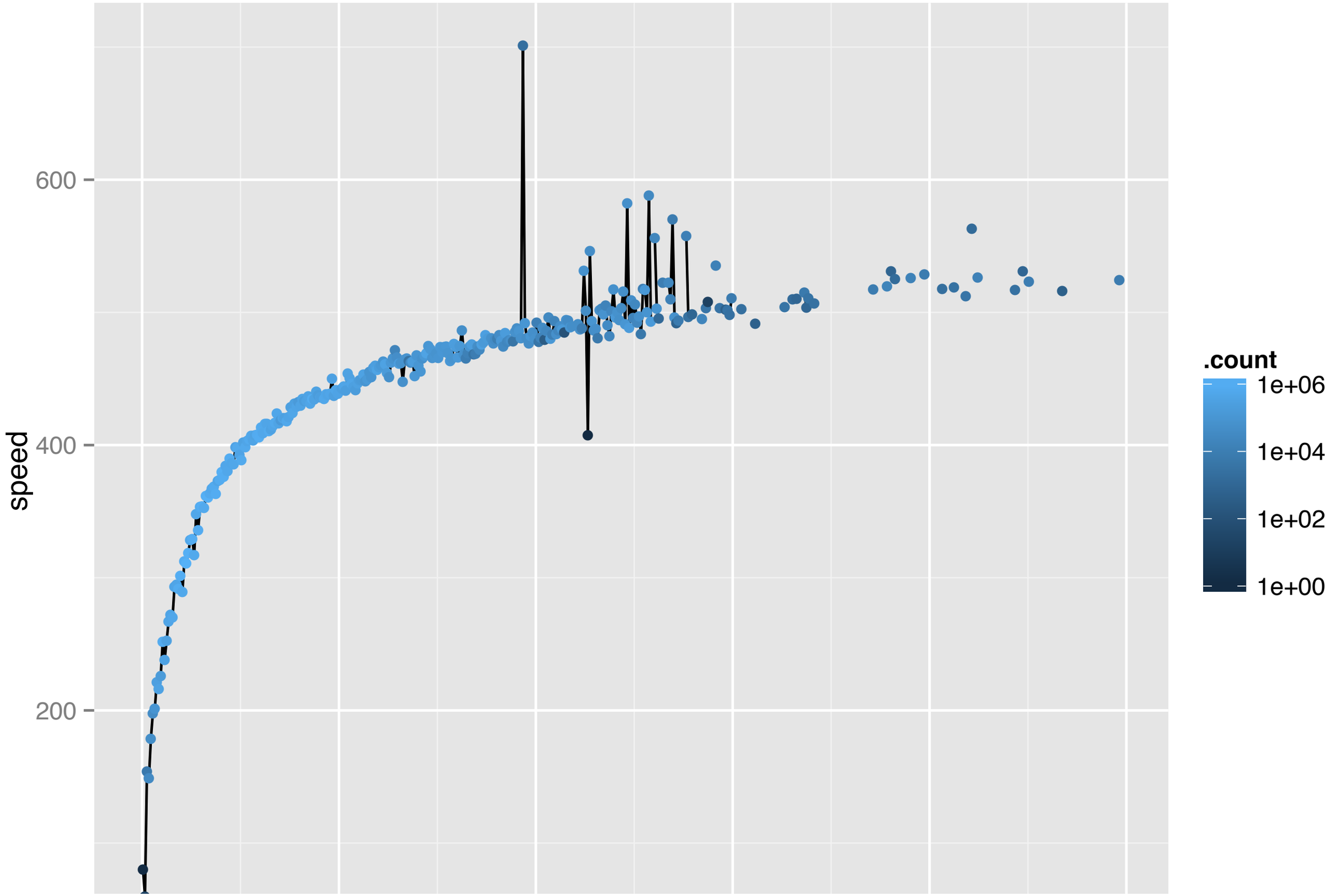



```
autoplot(time_s[time_s < 500, ])
```

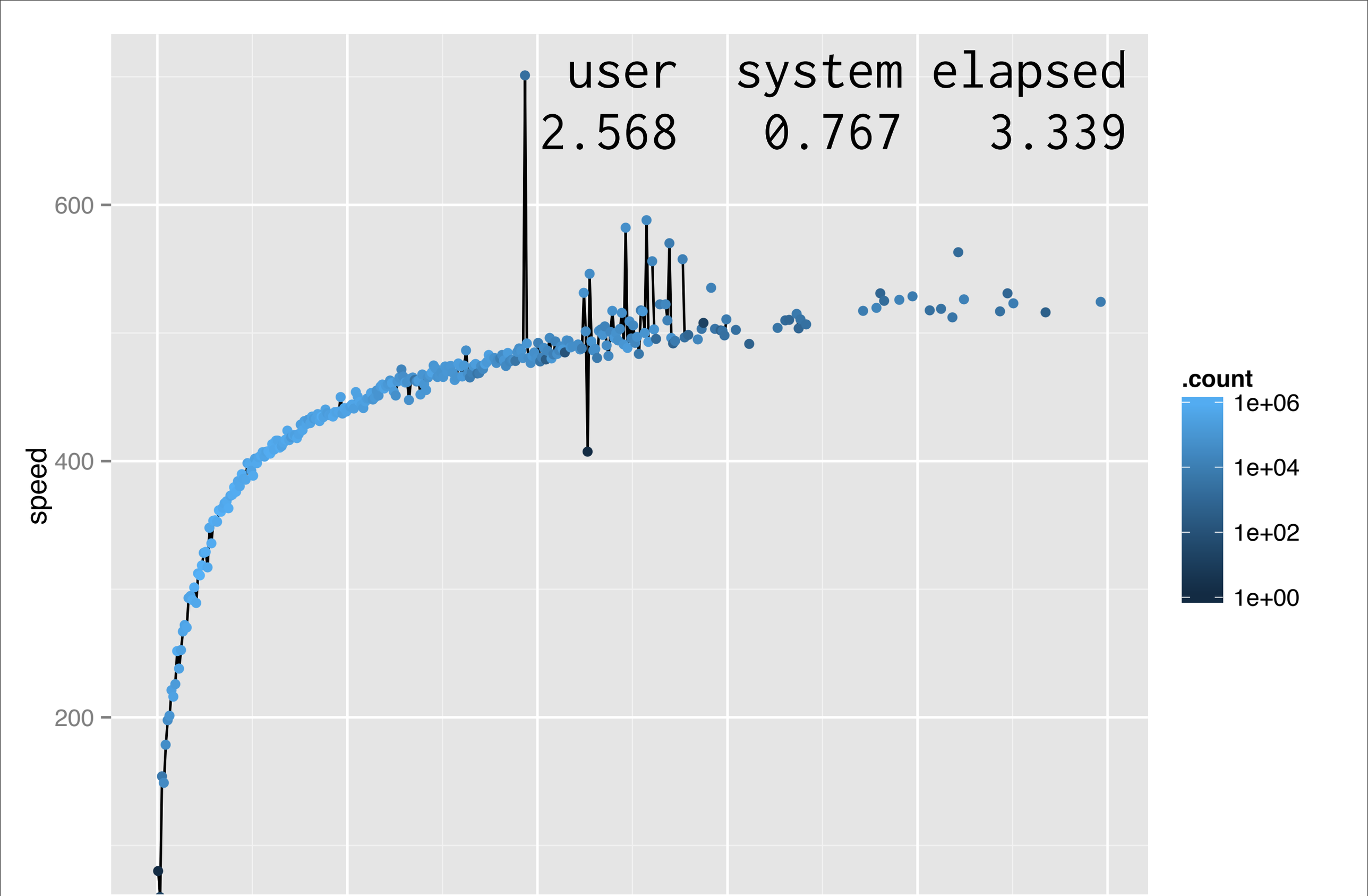



```
autoplot(time_s %% 60)
```


speed
600
400
200
0
1000
2000
3000
4000
5000
dist
.count
1e+06
1e+04
1e+02
1e+00




```
sd1 <- condense(bin(dist, 10), z = speed)
autoplot(sd1) + ylab("speed")
```



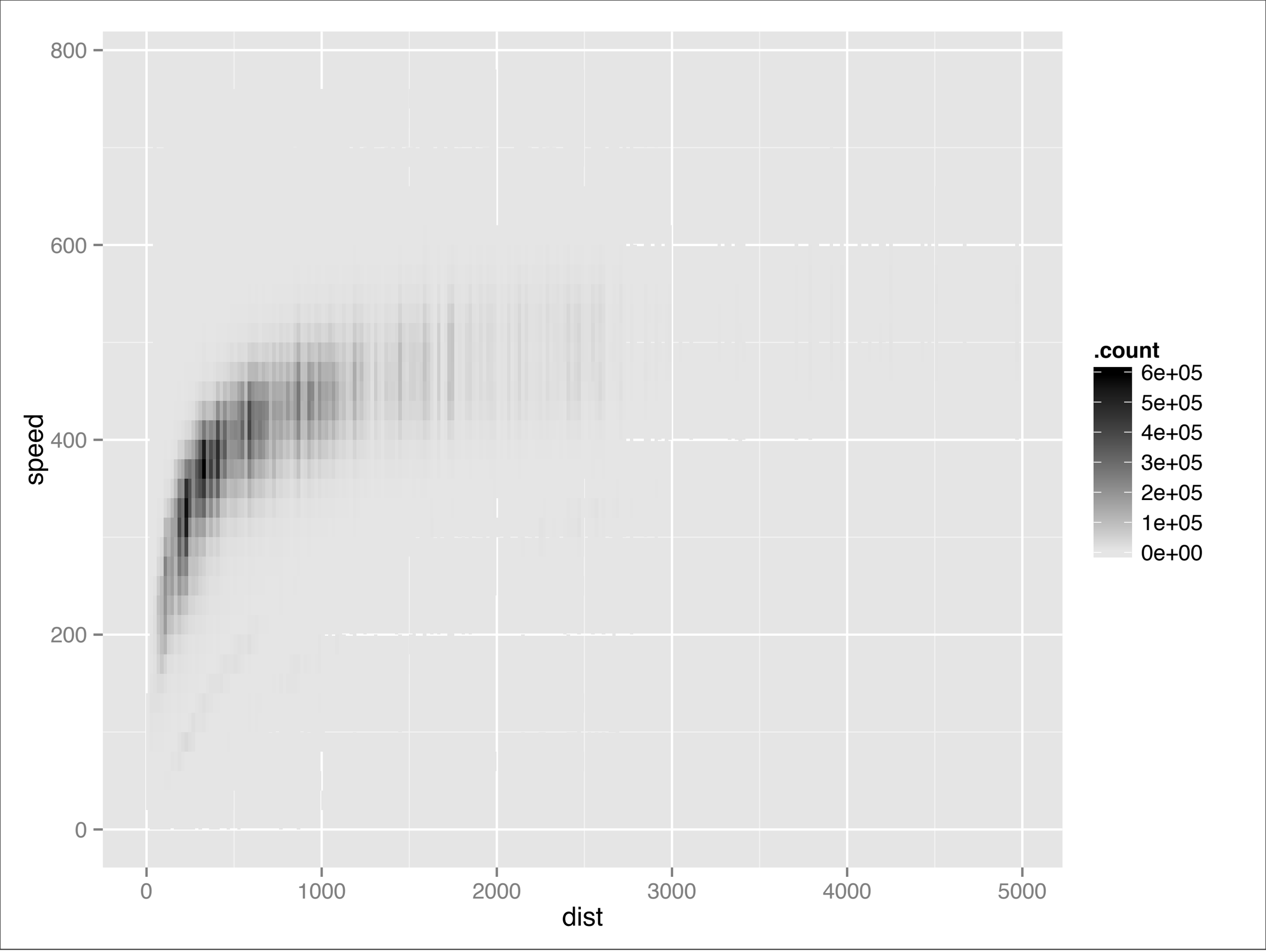

```
sd1 <- condense(bin(dist, 10), z = speed)
autoplot(sd1) + ylab("speed")
```

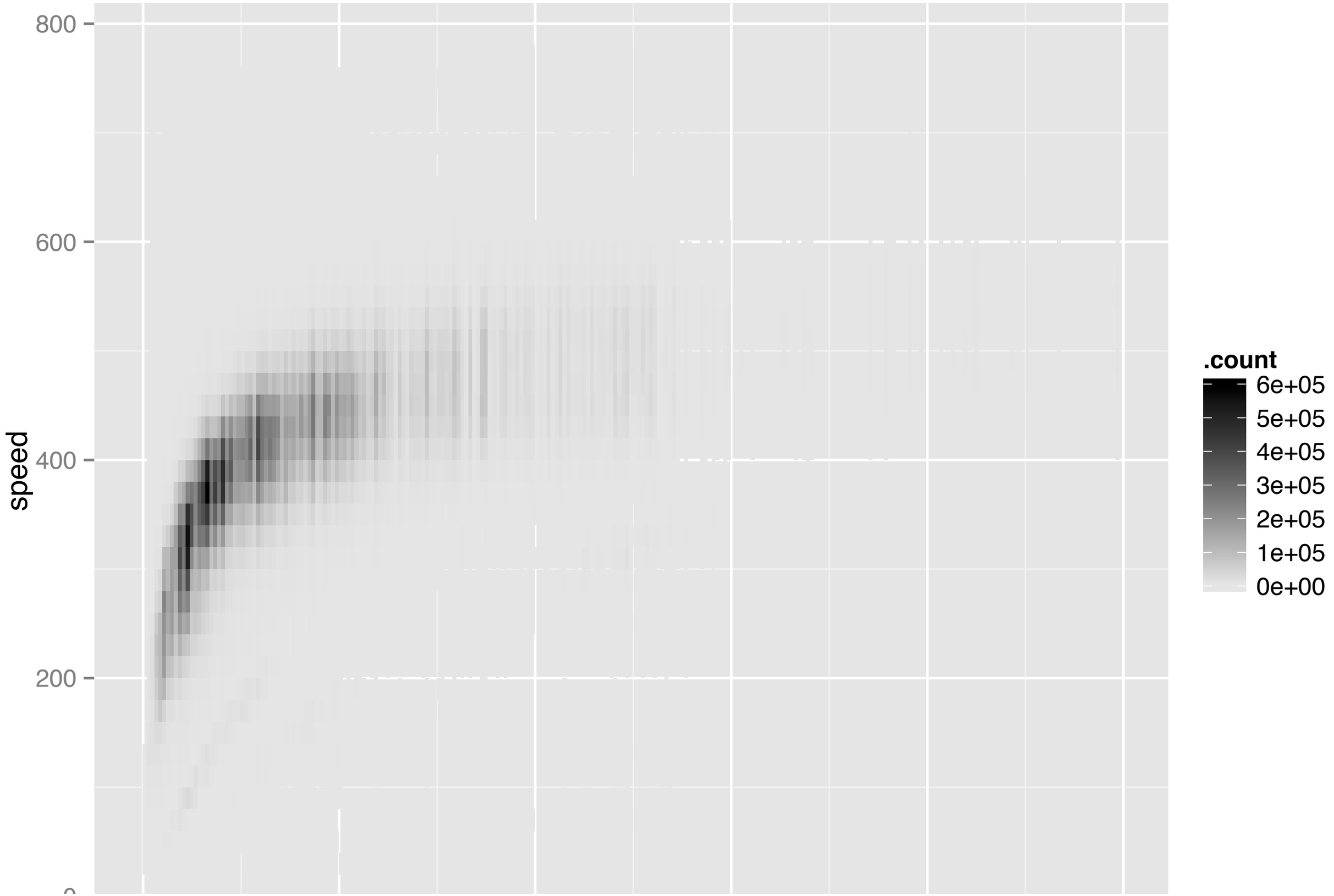
Distributive	1 value
Algebraic	m values
Holistic	f(n) values

Advantage of algebraic & distributive is that they can be re-aggregated which makes them trivially parallelisable & re-binnable

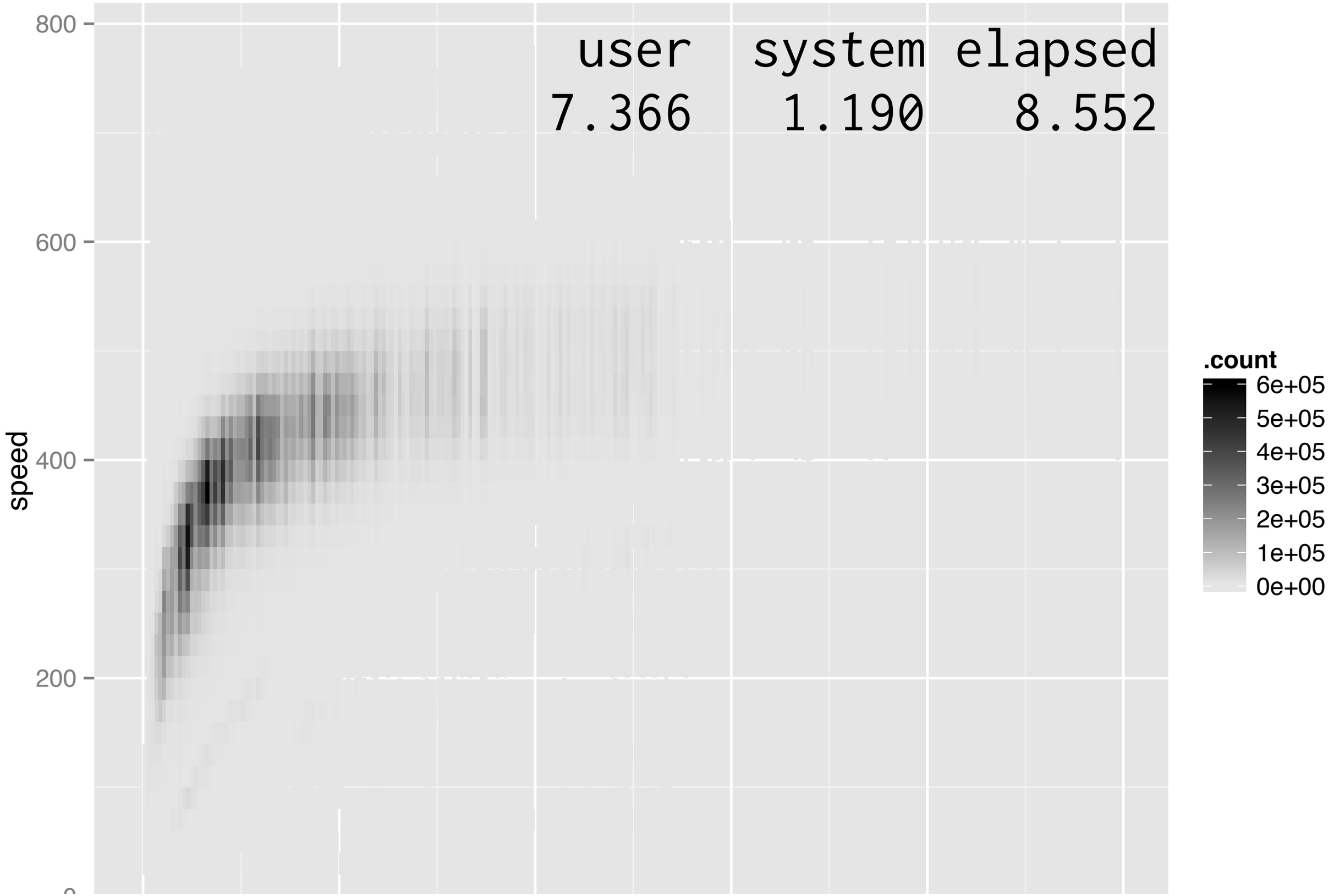
Distributive	sum, count, min, max
Algebraic	mean, sd
Holistic	quantile, cardinality

Advantage of algebraic & distributive is that they can be re-aggregated which makes them trivially parallelisable & re-binnable


speed
800
600
400
200
0
0
1000
2000
3000
4000
5000
dist
.count
6e+05
5e+05
4e+05
3e+05
2e+05
1e+05
0e+00




```
sd2 <- condense(bin(dist, 20), bin(speed, 20))
autoplot(sd2)
```

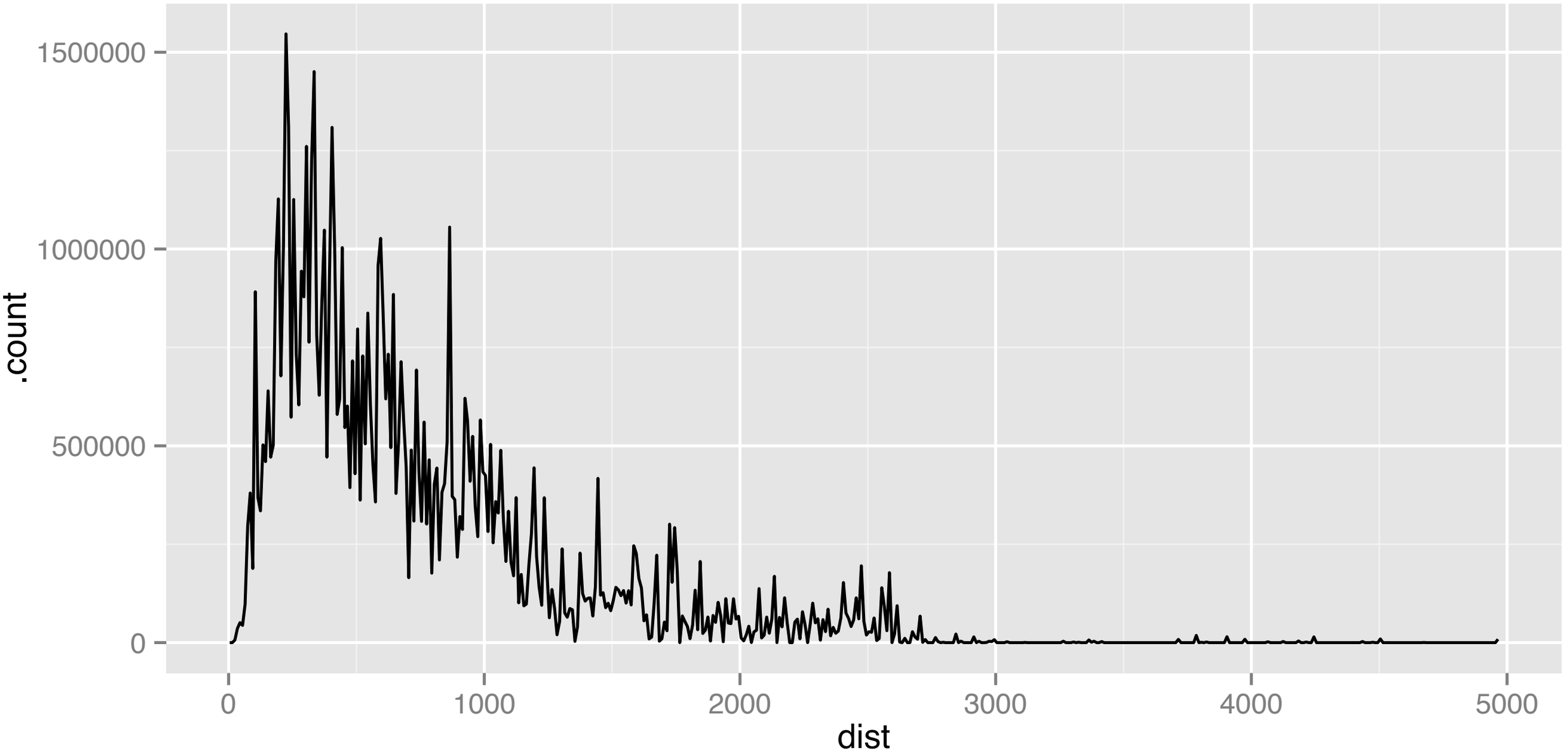



```
sd2 <- condense(bin(dist, 20), bin(speed, 20))
autoplot(sd2)
```

Smooth

Why smooth?

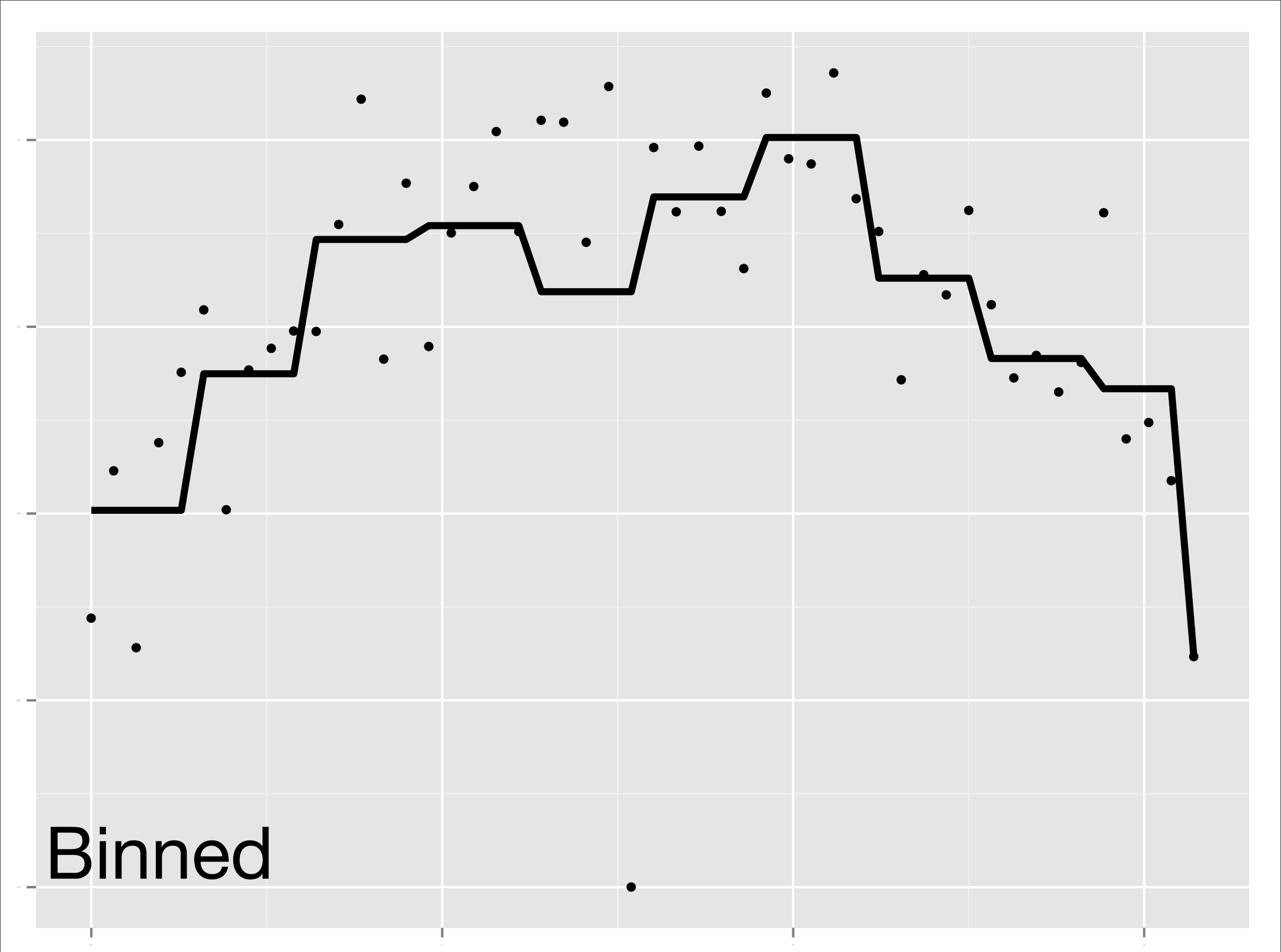
- Fix over binning
- Dampen effect of outliers
- Focus on main trends

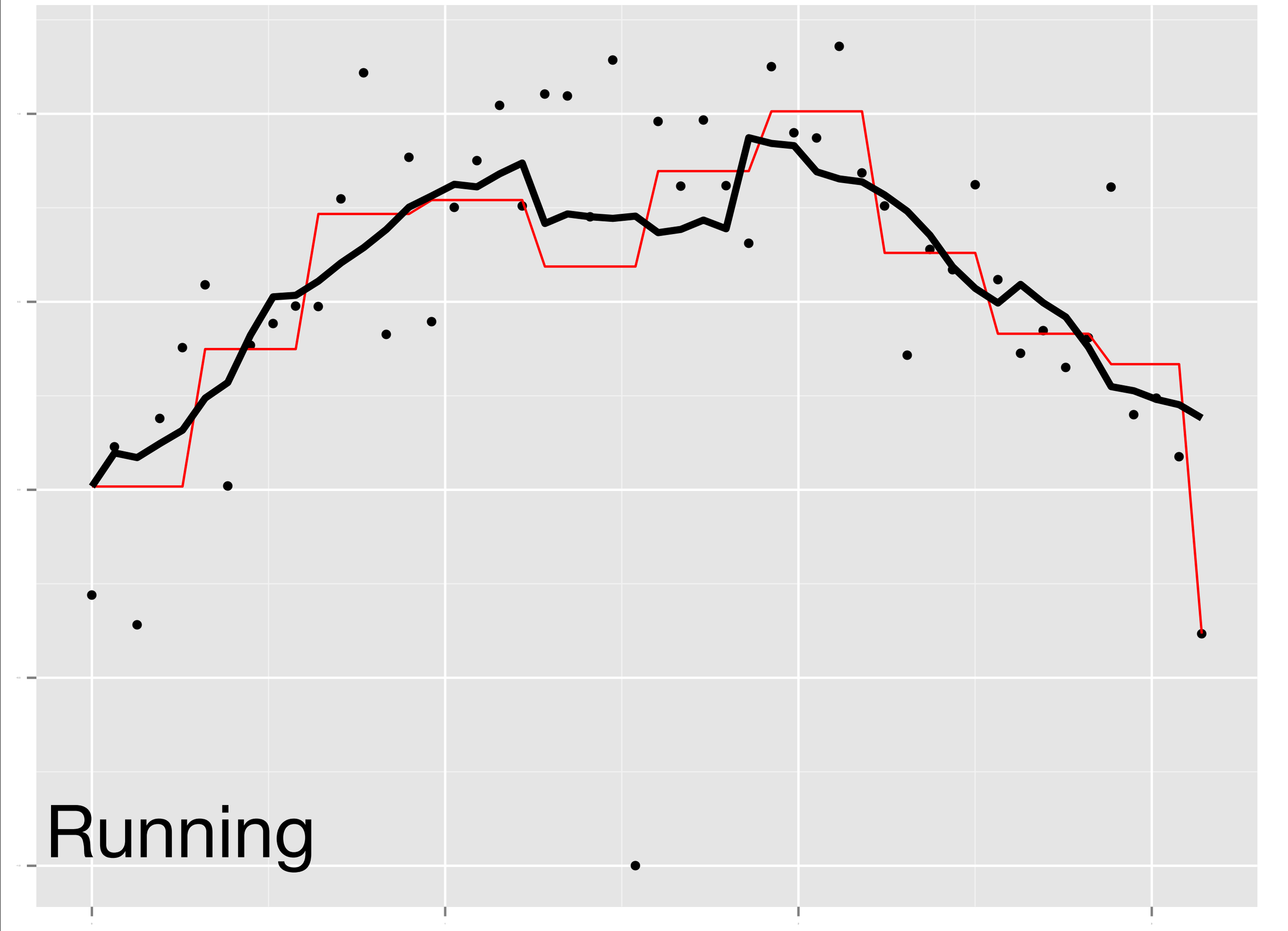



```
dist_s <- condense(bin(dist, 10))
autoplot(dist_s)
```

Demo

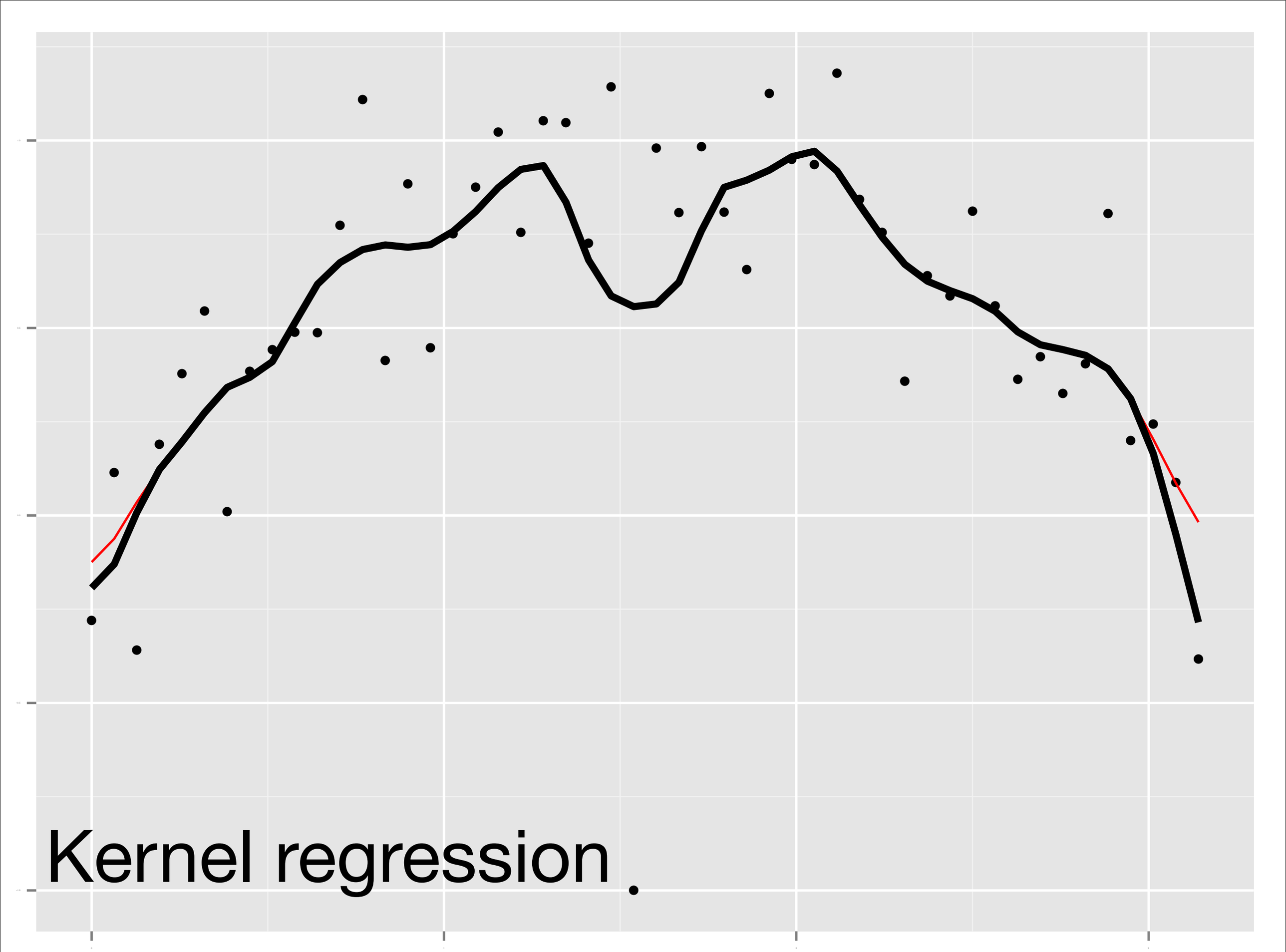
```
shiny::runApp("smooth/", 8000)
```

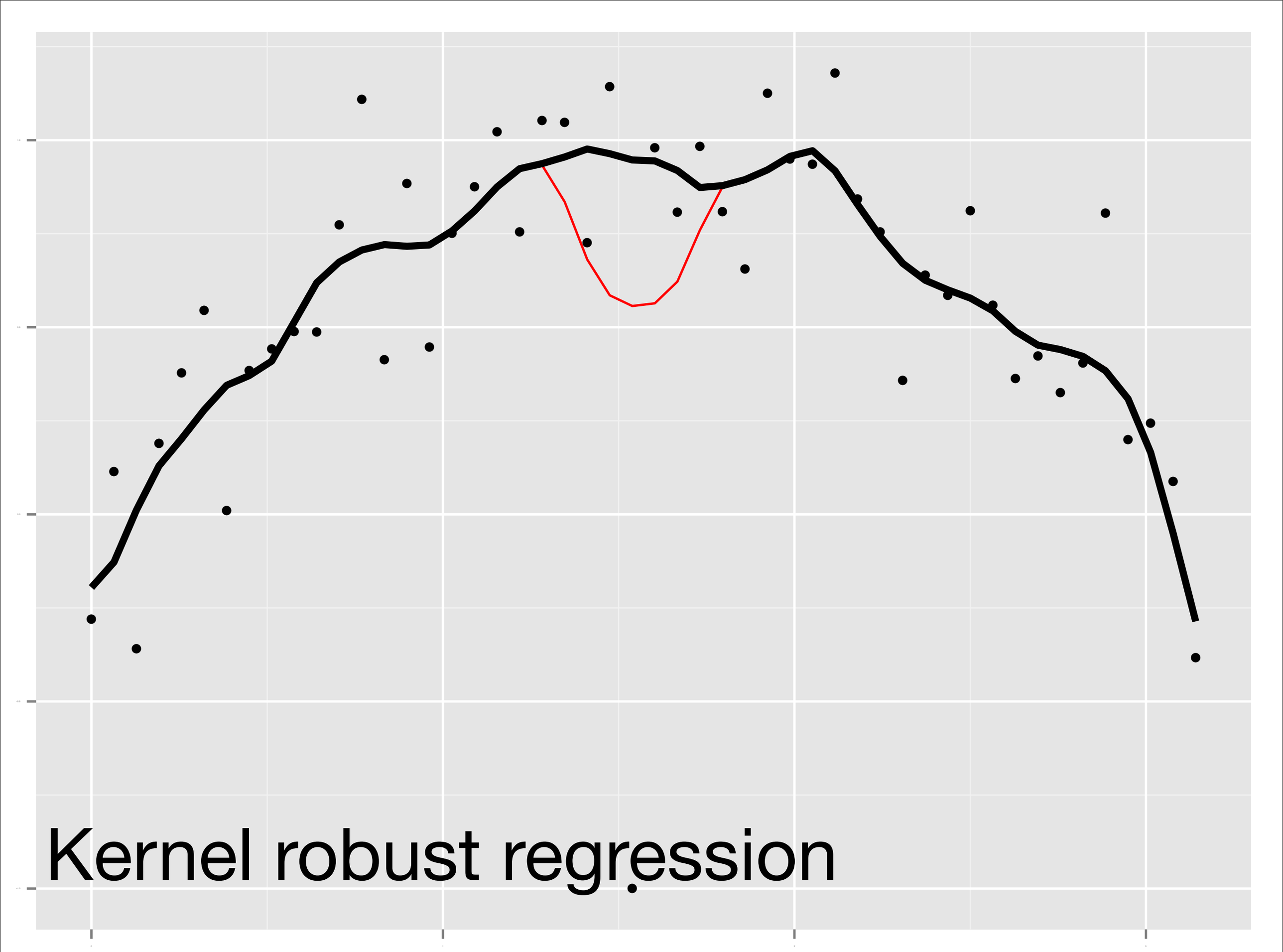

Binned


Running

$$K(x) = \left(1 - |x|^3\right)^2 I_{|x|<1}$$

Kernel mean


Kernel regression


Kernel robust regression

Locally constant (Nadaraya-Watson, kernel mean)	Convolution (=fast)

Locally constant (Nadaraya-Watson, kernel mean)	Convolution (=fast)
Locally linear (Kernel regression/ smooth)	Better boundary behaviour

Locally constant (Nadaraya-Watson, kernel mean)	Convolution (=fast)
Locally linear (Kernel regression/ smooth)	Better boundary behaviour
Locally linear (robust) (loess)	Better resistance to outliers

“Best” bandwidth?

- Estimate using leave-one out cross-validation of rmse
- Not “optimal” for visualisation, but a good place to start.
- Possible to compute in one pass for locally constant smooths. (May be possible for others with enough thought)

Visualise

Challenges

- Prepare for outliers
- Always display count
- Always display missing values

Studio

Demo

```
shiny::runApp("mt/", 8002)
```

Rcpp

Dirk Eddelbuettel, Romain Francois,
& JJ Allaire

```
library(Rcpp)
cppFunction('int one() {
   return 1;
}')
one()
```

Generate C++ file

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
int one() {
  return 1;
}
```

Expose C++ to C

```
#include <Rcpp.h>

RcppExport SEXP sourceCpp_86581_one() {
BEGIN_RCPP
    Rcpp::RNGScope __rngScope;
    int __result = one();
    return Rcpp::wrap(__result);
END_RCPP
}
```

Converts C++ object to R object

Compile C & C++ code to a DLL

```
/Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB -o
'sourceCpp_36763.so' 'file5907496612f3.cpp'
clang++ -I/Library/Frameworks/R.framework/Resources/include -
I/Library/Frameworks/R.framework/Resources/include/x86_64 -
DNDEBUG  -I/usr/local/include  -I"/Users/hadley/R/Rcpp/
include"    -fPIC  -g -O2  -c file5907496612f3.cpp -o
file5907496612f3.o
g++ -arch x86_64 -dynamiclib -Wl,-headerpad_max_install_names
-undefined dynamic_lookup -single_module -multiply_defined
suppress -L/usr/local/lib -o sourceCpp_36763.so
file5907496612f3.o /Users/hadley/R/Rcpp/lib/x86_64/libRcpp.a -
F/Library/Frameworks/R.framework/.. -framework R -Wl,-
framework -Wl,CoreFoundation
```

Dynamically link DLL

```
`.sourceCpp_86581_DLLInfo` <-
dyn.load('/tmp/Rtmpt0ZCNp/
sourcecpp_2cf047b27139/
sourceCpp_91998.so')
```

Connect to C function in DLL to R

```
one <-
Rcpp:::sourceCppFunction(function() {},
FALSE, `.sourceCpp_86581_DLLInfo`,
'sourceCpp_86581_one')

rm(`.sourceCpp_86581_DLLInfo`)
```

```
cppFunction("int one() {
  return 1;
}")
cppFunction("int one() {
  return 1;
}")
# Doesn't need to recompile!
one()
```

Why C++?

- Modern, high-performance language
- Precise control over memory allocation and copying.
- Excellent built-in libraries (e.g. STL)
- With Rcpp, much easier than C/Fortran
- Not too hard to learn

Google for:

“Rcpp”

“Rcpp gallery”

“Rcpp hadley”

Shiny

Joe Chen & Winston Chang

```
library(shiny)
runApp("smooth/")
```

ui.r

```
shinyUI(pageWithSidebar(
  headerPanel("Smoothing"),
  sidebarPanel(sliderInput(inputId = "h",
    label = "Bandwidth (in multiples of binwidth):",
    min = 1, max = 20, value = 1, step = 0.1)),
  mainPanel(plotOutput(
    outputId = "plot", height = "300px"))
))
```

server.r

```
library(bigvis)
library(ggplot2)
library(plyr)

dist <- readRDS("dist.rds")
dist_s <- condense(bin(dist, 10))

shinyServer(function(input, output) {
  output$plot <- renderPlot({
    n <- as.numeric(input$h)
    if (n <= 1) {
      print(autoplot(dist_s))
    } else {
      print(autoplot(smooth(dist_s, n * 10)))
    }
  })
})
```

Why shiny?

- Create web apps easily with knowing html, js, css, ...
- You describe connections between UI and data & shiny takes care of managing the updates
- (Eventually) easily deploy locally or in the cloud

Google for:

“shiny”

“shiny mailing list”

“shiny tutorial”

Conclusions

Performance

- "Interactive" exploration of 100,000,000 observations is possible in R
- Key is use of C++ and extreme care with memory allocation/copying
- RCpp makes this v. easy

Future work

- Multi-core + out-of-memory
- In database, where possible
- More summary statistics

Google for:

“bigvis”