Image analysis

Getting and Cleaning Data

Source: Google

PNAS

Using deep learning and Google Street View to estimate the demographic makeup of neighborhoods across the United States

Timnit Gebru[a,1], Jonathan Krause[a], Yilun Wang[a], Duyun Chen[a], Jia Deng[b], Erez Lieberman Aiden[c,d,e], and Li Fei-Fei[a]

[a]Artificial Intelligence Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305; [b]Vision and Learning Laboratory, Computer Science and Engineering Department, University of Michigan, Ann Arbor, MI 48109; [c]The Center for Genome Architecture, Department of Genetics, Baylor College of Medicine, Houston, TX 77030; [d]Department of Computer Science, Rice University, Houston, TX 77005; and [e]The Center for Genome Architecture, Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005

https://www.pnas.org/content/114/50/13108

The cars in this image of a Brooklyn neighborhood can reveal a lot about the residents there. // Google Street View

Google Street View Can Reveal How Your Neighborhood Votes

LINDA POON DEC 6, 2017

https://www.citylab.com/transportation/2017/12/google-street-view-data-demographics-cars-research/547436/

The magick package: Advanced Image-Processing in R

2018-05-11

The new magick package is an ambitious effort to modernize and simplify high-quality image processing in R. It wraps the ImageMagick STL which is perhaps the most comprehensive open-source image processing library available today.

The ImageMagick library has an overwhelming amount of functionality. The current version of Magick exposes a decent chunk of it, but being a first release, documentation is still sparse. This post briefly introduces the most important concepts to get started.

Installing `magick`

On Windows or OS-X the package is most easily installed via CRAN.

```
install.packages("magick")
```

The binary CRAN packages work out of the box and have most important features enabled. Use `magick_config` to see which features and formats are supported by your version of ImageMagick.

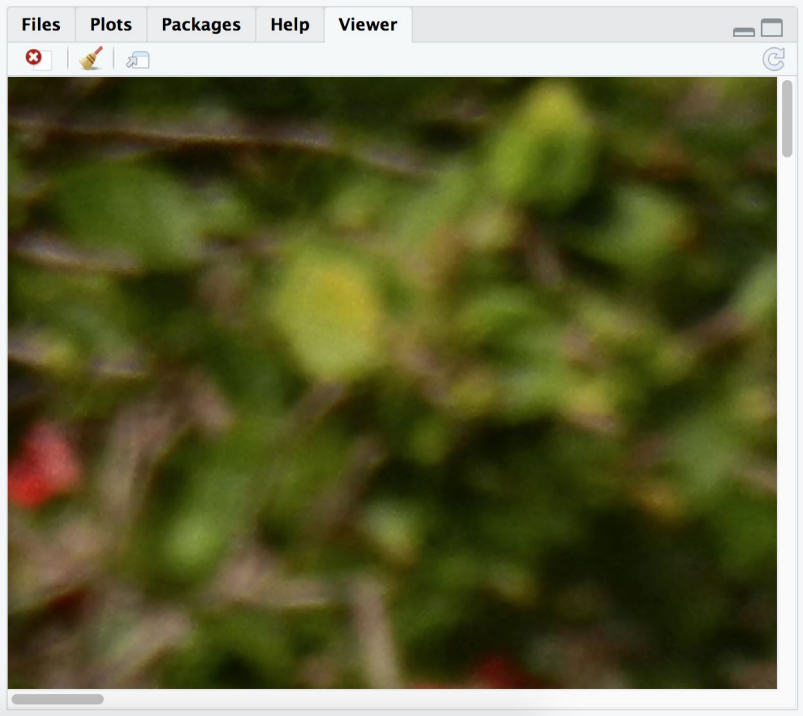
```
library(magick)
```

```
## Linking to ImageMagick 6.9.9.39
## Enabled features: cairo, fontconfig, freetype, lcms, pango, rsvg, webp
## Disabled features: fftw, ghostscript, x11
```

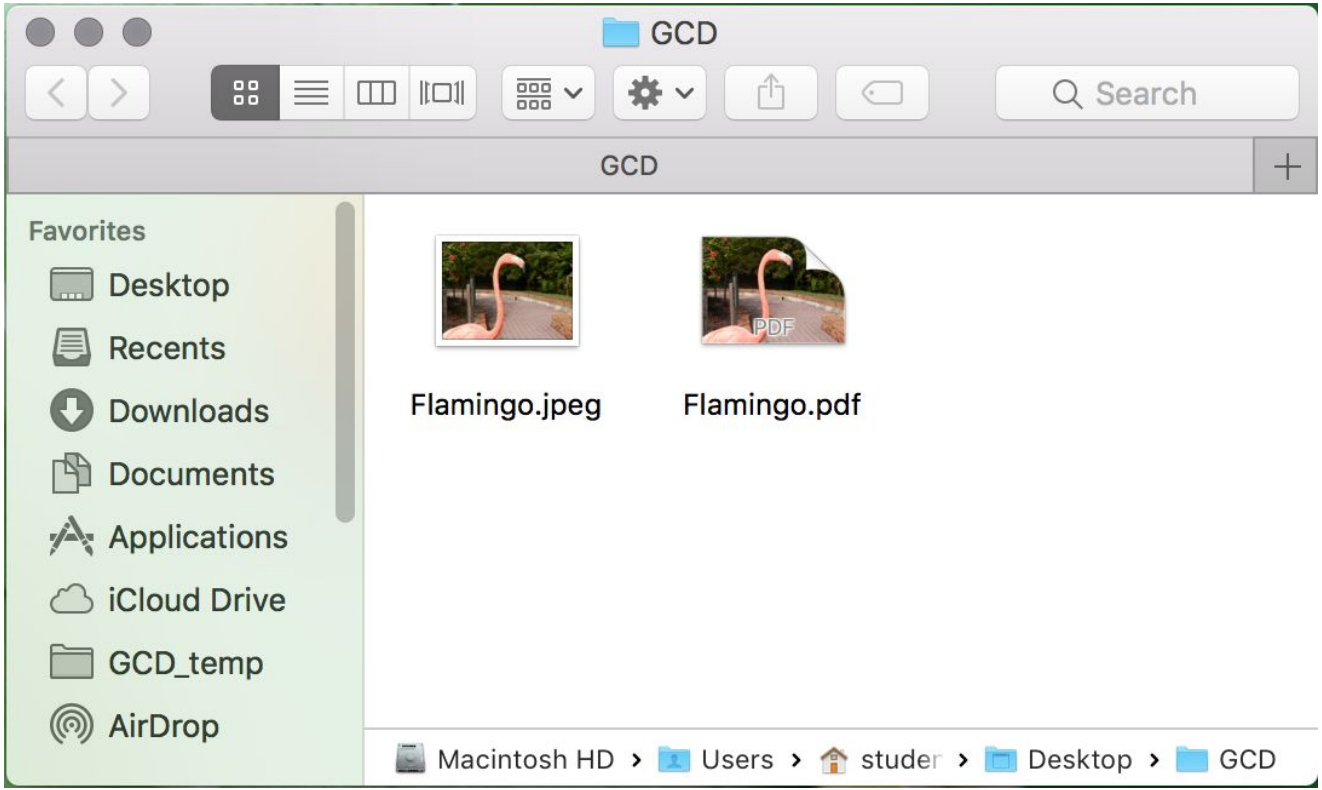
https://cran.r-project.org/web/packages/magick/vignettes/intro.html

```r
# Download the flamingo image from
# https://drive.google.com/file/d/1Y8z2ukKaa63S_XpbDHFw7YcmUFDhkk9Q/view?usp=sharing

# We will have to specify the path to the image file
# In this example, our image file is stored in a directory called "GCD" on the Desktop

flamingo <- image_read('~/Desktop/GCD/Flamingo.jpeg')

# Call the flamingo object
flamingo
```

```
# A tibble: 1 x 7
  format width height colorspace matte filesize density
  <chr>  <int>  <int> <chr>      <lgl>    <int> <chr>
1 JPEG    4000   3000 sRGB       FALSE  2809245 300x300
```



```r
# Write out the image of the flamingo as a PDF to a folder on your desktop called GCD
image_write(flamingo, path = "~/Desktop/GCD/Flamingo.pdf", format = "pdf")
```



The magick package: Advanced Image-Processing in R

2018-05-11

The new magick package is an ambitious effort to modernize and simplify high-quality image processing in R. It wraps the ImageMagick STL which is perhaps the most comprehensive open-source image processing library available today.

The ImageMagick library has an overwhelming amount of functionality. The current version of Magick exposes a decent chunk of it, but being a first release, documentation is still sparse. This post briefly introduces the most important concepts to get started.

Installing `magick`

On Windows or OS-X the package is most easily installed via CRAN.

```
install.packages("magick")
```

The binary CRAN packages work out of the box and have most important features enabled. Use `magick_config` to see which features and formats are supported by your version of ImageMagick.

```
library(magick)
```

```
## Linking to ImageMagick 6.9.9.39
## Enabled features: cairo, fontconfig, freetype, lcms, pango, rsvg, webp
## Disabled features: fftw, ghostscript, x11
```

https://cran.r-project.org/web/packages/magick/vignettes/intro.html

%>%

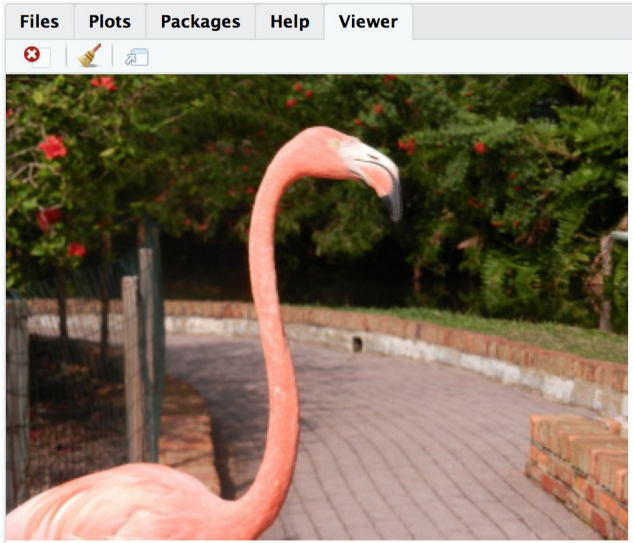
Ceci n'est pas une pipe.

```
# Specify you want to change the width to 400 pixels
image_scale(flamingo, "400x")
```

```
# A tibble: 1 x 7
  format width height colorspace matte filesize density
  <chr>  <int>  <int> <chr>      <lgl>    <int> <chr>
1 JPEG     400    300 sRGB       FALSE        0 300x300
```

```
# Specify you want to change the height to 400 pixels
image_scale(flamingo, "x400")
```

```
# A tibble: 1 x 7
  format width height colorspace matte filesize density
  <chr>  <int>  <int> <chr>      <lgl>    <int> <chr>
1 JPEG     533    400 sRGB       FALSE        0 300x300
```



By specifying the width to 400 pixels, the height becomes 300 pixels

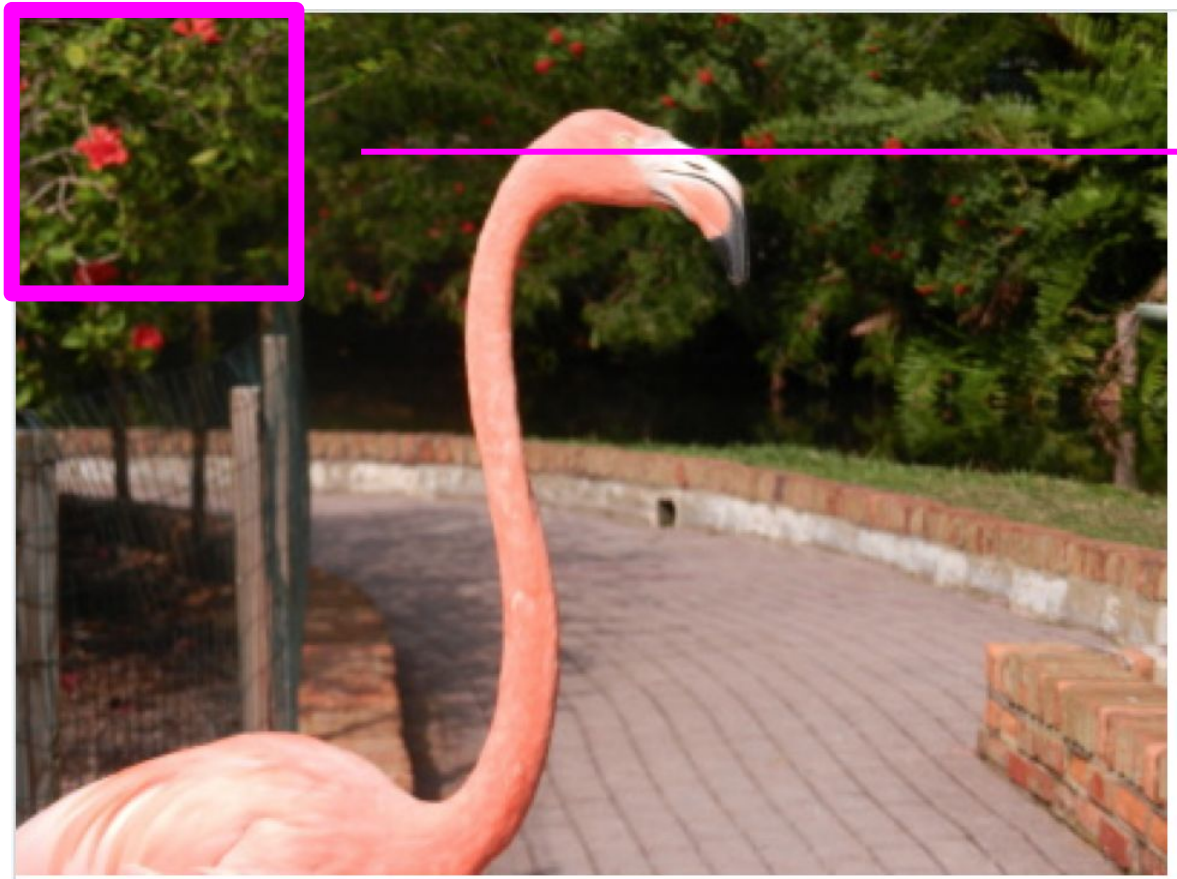
Width = 400 pixels

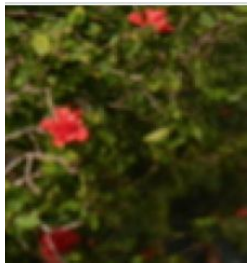
```r
# Use image_rotate() with a positive
# integer to rotate clockwise
image_scale(flamingo, "400x") %>%
  image_rotate(10)
```

```r
# Use image_rotate() with a negative
# integer to rotate anti-clockwise
image_scale(flamingo, "400x") %>%
  image_rotate(-10)
```

```r
# Rotating your image 350 degrees is the
# same as anti-clockwise rotating 10 degrees
image_scale(flamingo, "400x") %>%
  image_rotate(350)
```

```
# To keep the upper-left 100x100 pixels
image_scale(flamingo, "400x") %>%
  image_crop("100x100")
```



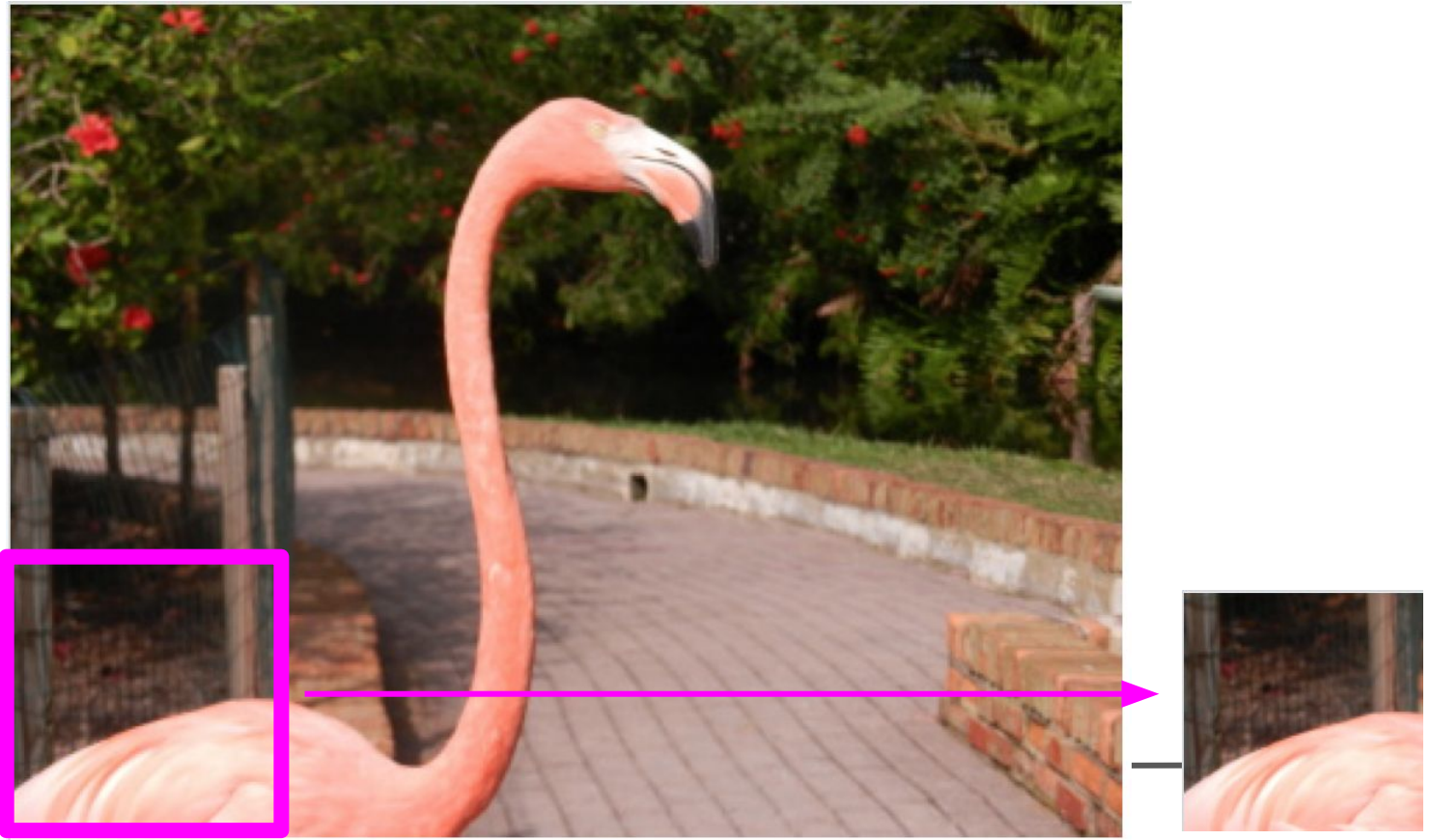


```
# To keep the bottom-left 100x100 pixels
image_scale(flamingo, "400x") %>%
  image_crop("100x100+0+200")
```

The height of the image is 300 pixels.

We want the bottom 100 pixels.

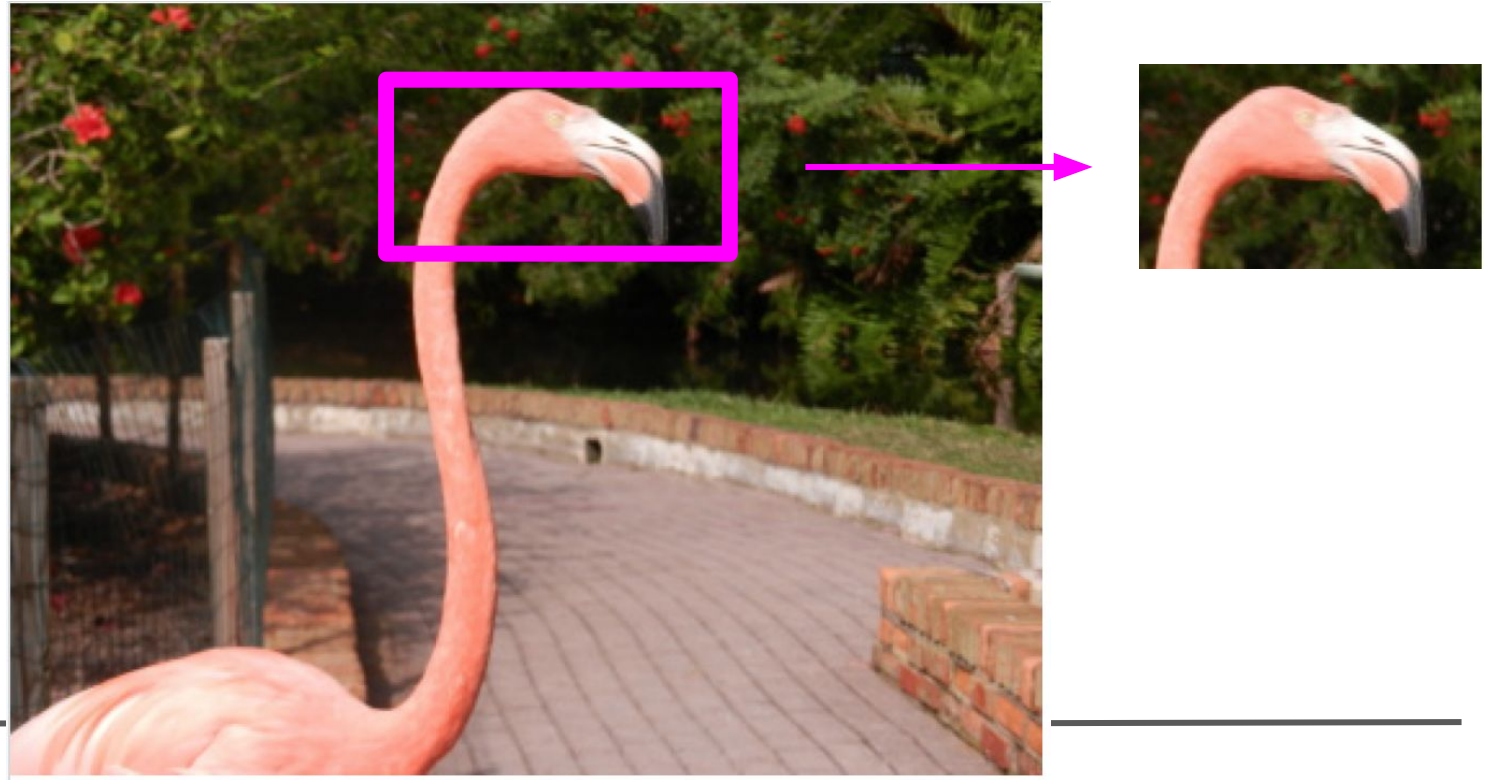
We need to set a Y-offset of 200 pixels



```
# To crop out the pixels surrounding the flamingo head,
# you might end up with dimensions that look something like this
image_scale(flamingo, "400x") %>%
  image_crop("125x75+150+25")
```

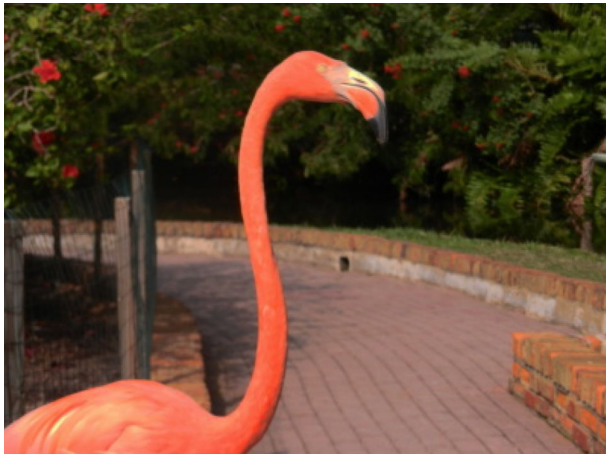
The flamingo head is approximately 125 pixels wide and 75 pixels tall.

It starts approximately 150 pixels in from the left (X-offset) and 25 pixels from the top (Y-offset).

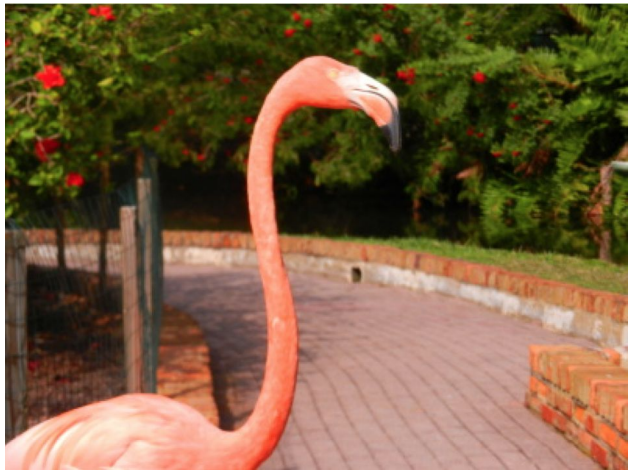


```
# Let's increase the brightness
image_scale(flamingo, "400x") %>%
  image_modulate(brightness = 120)
```

```
# Let's decrease the brightness
image_scale(flamingo, "400x") %>%
  image_modulate(brightness = 80)
```

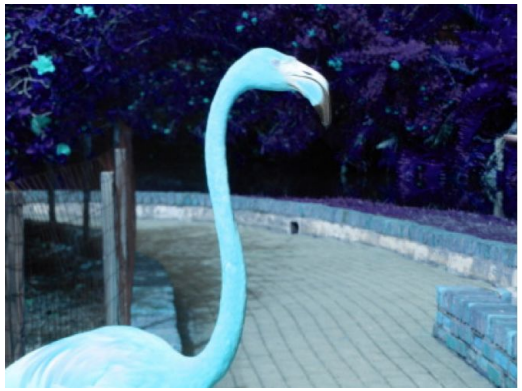


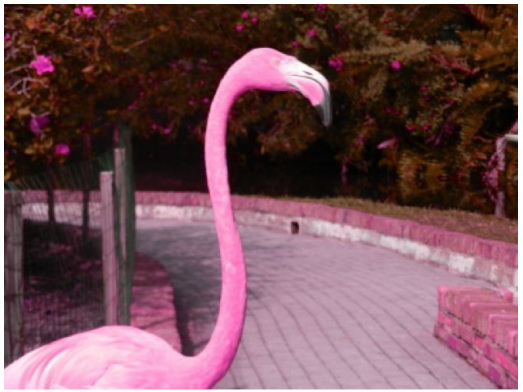
```
# Let's (really) oversaturate our image
image_scale(flamingo, "400x") %>%
  image_modulate(saturation = 150)
```

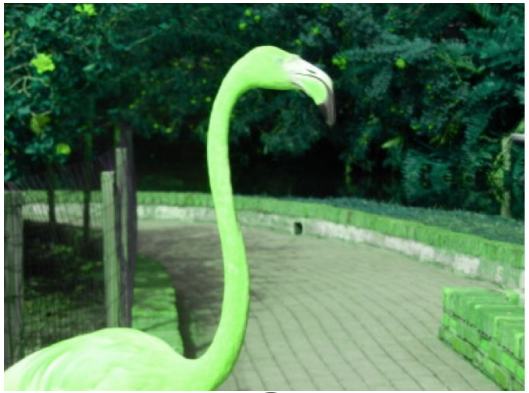


```
# Let's desaturate our image
image_scale(flamingo, "400x") %>%
  image_modulate(saturation = 80)
```

```
# Blue
image_scale(flamingo, "400x") %>%
  image_modulate(hue = 0)

# Purple-ish
image_scale(flamingo, "400x") %>%
  image_modulate(hue = 25)

# Magenta-ish
image_scale(flamingo, "400x") %>%
  image_modulate(hue = 50)

# Pink!
image_scale(flamingo, "400x") %>%
  image_modulate(hue = 75)

# The original, unchanged
image_scale(flamingo, "400x") %>%
  image_modulate(hue = 100)

# Yellow
image_scale(flamingo, "400x") %>%
  image_modulate(hue = 125)

# Green
image_scale(flamingo, "400x") %>%
  image_modulate(hue = 150)

# More green?
image_scale(flamingo, "400x") %>%
  image_modulate(hue = 175)

# Back to blue!
image_scale(flamingo, "400x") %>%
  image_modulate(hue = 200)
```







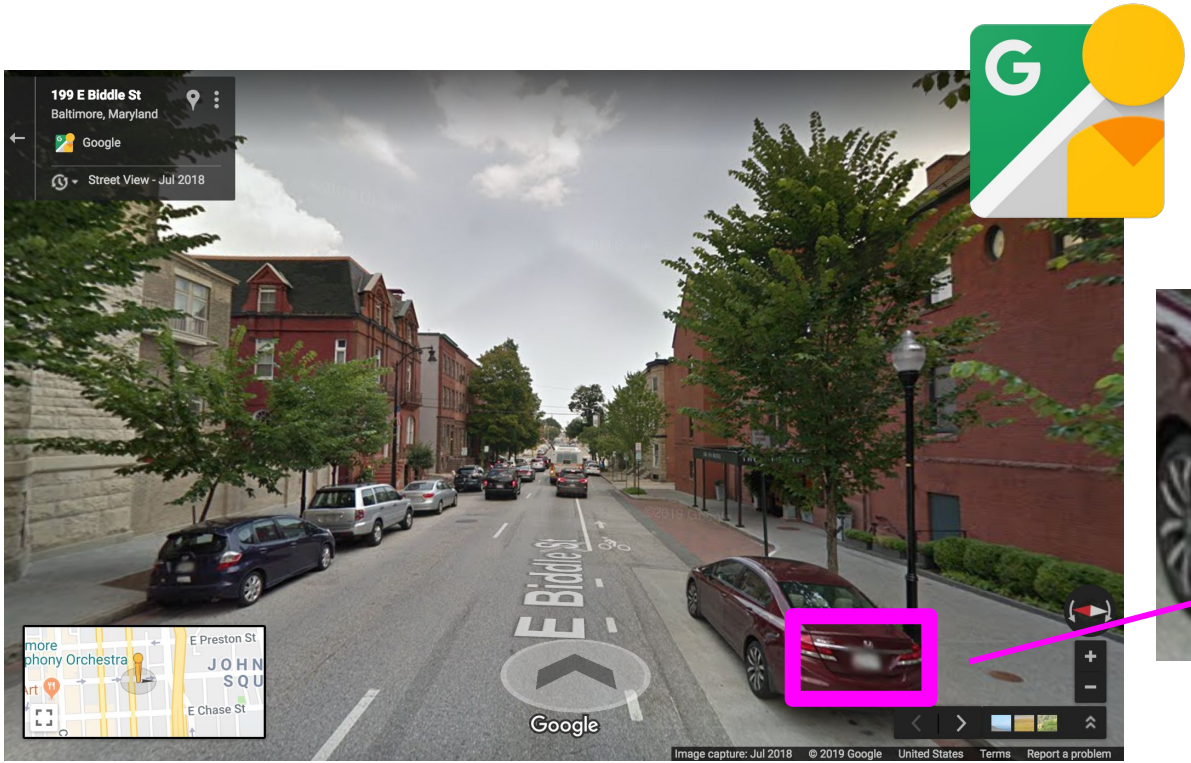
```
> flamingo
# A tibble: 1 x 7
  format width height colorspace matte filesize density
  <chr>  <int>  <int> <chr>      <lgl>    <int> <chr>
1 JPEG    4000   3000 sRGB       FALSE  2809245 300x300
```

sRGB stands for “standard Red, Green, Blue”

```
# Change the colorspace to gray
image_scale(flamingo, "400x") %>%
  image_quantize(colorspace = 'gray')

# A tibble: 1 x 7
  format width height colorspace matte filesize density
  <chr>  <int>  <int> <chr>      <lgl>    <int> <chr>
1 JPEG     400    300 Gray       FALSE        0 300x300
```

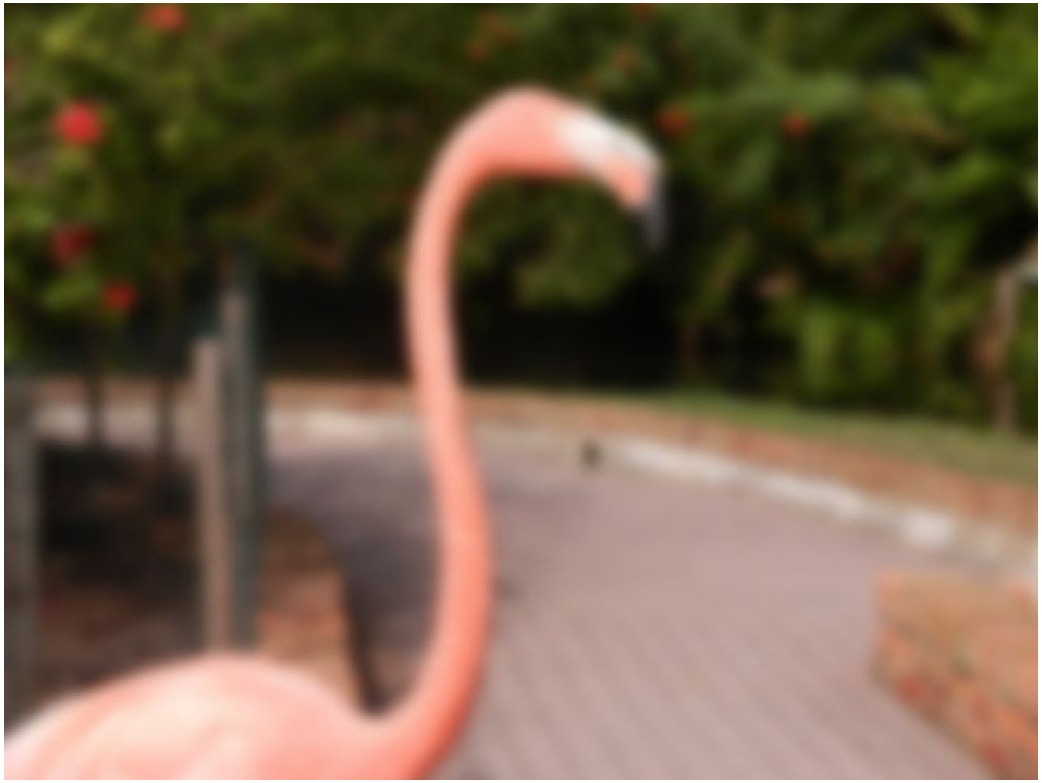
```
# Original
image_scale(flamingo, "400x")
```

```
# Increase the contrast
image_scale(flamingo, "400x") %>%
  image_contrast(2)
```



Source: Google

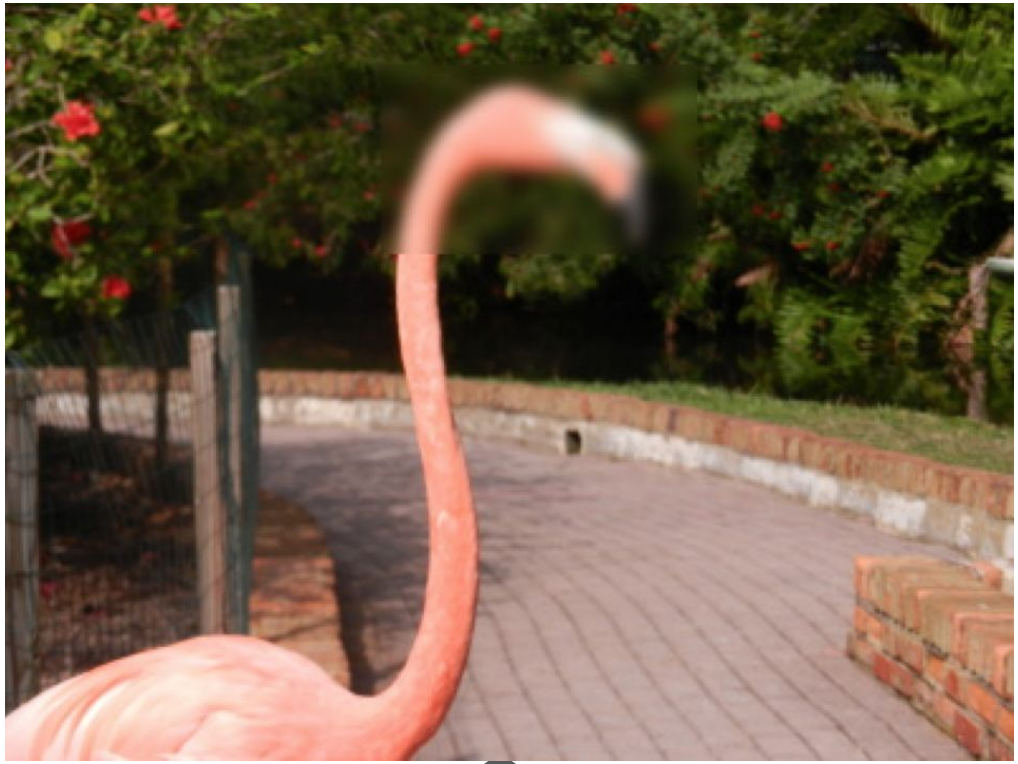
```
# Let's blur our flamingo!
image_scale(flamingo, "400x") %>%
  image_blur(0, 3)
```



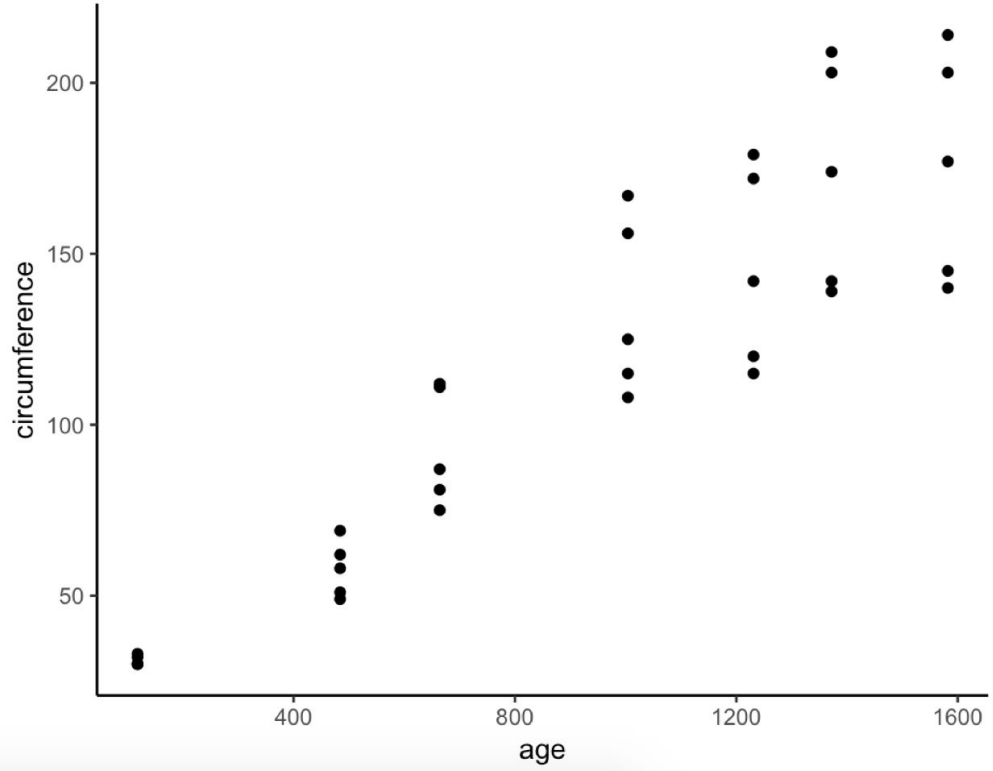
```r
# Crop out the blurred flamingo head and save to a new image object
fl_head <- image_scale(flamingo, "400x") %>%
  image_blur(0, 3) %>%
  image_crop("125x75+150+25")

# Look at the saved object
fl_head
```

```
# A tibble: 1 x 7
  format width height colorspace matte filesize density
  <chr>  <int>  <int> <chr>      <lgl>    <int> <chr>
1 JPEG     125     75 sRGB       FALSE        0 300x300
```

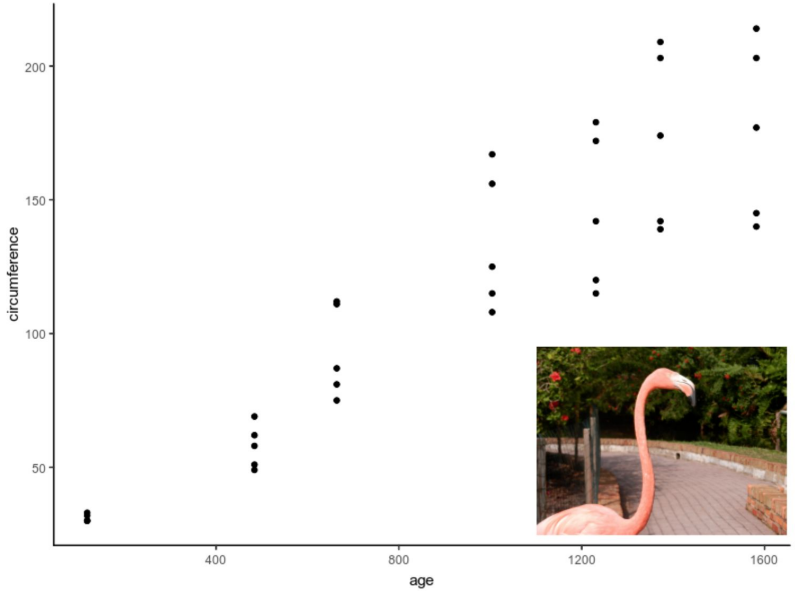
```
# Combine the original image with the blurred flamingo head
# using image_composite()
image_composite(image_scale(flamingo, "400x"), fl_head, offset = "+150+25")
```



```
ggplot(Orange, aes(x = age, y = circumference)) +
  geom_point() +
  theme_classic()
```



```
orange_plot <- image_graph(width = 800, height = 600, res = 100)
ggplot(Orange, aes(x = age, y = circumference)) +
  geom_point() +
  theme_classic()
dev.off()

image_composite(orange_plot, image_scale(flamingo, "250x"),
                offset = "+540+350")
```



The plot is 800 pixels wide by 600 pixels tall.

We specify that we want our flamingo image to be 250 pixels wide. This makes it 188 pixels tall.

We know our X-offset is going to have to be approximately 800-250 = 550. Our Y-offset is going to be approximately 600-188 = 412, but we don't want to block the X-axis, so our Y-offset will have to be less than 400.

Summarizing: Image analysis

Getting and Cleaning Data