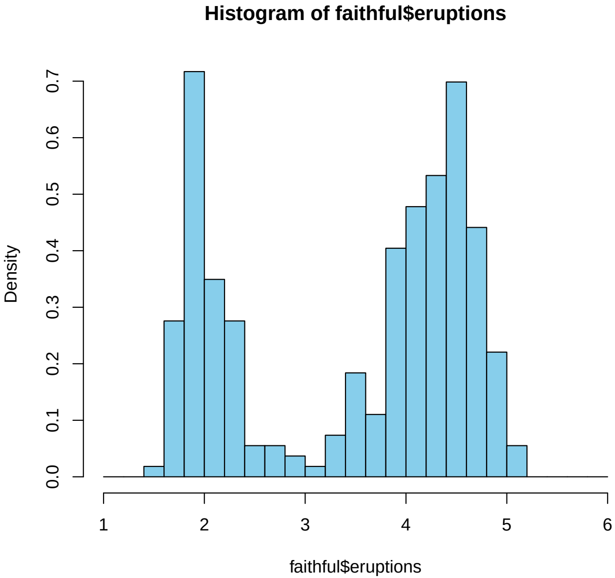
Stat 302
Statistical Software and Its Applications
Density Estimation

Yen-Chi Chen

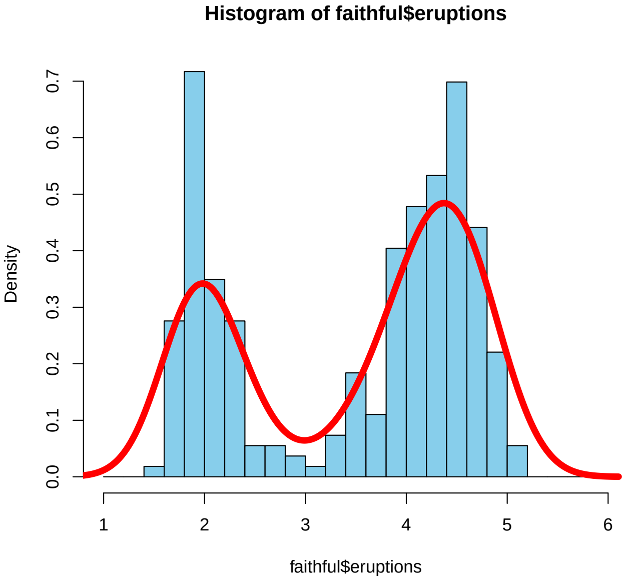
Department of Statistics, University of Washington

Autumn 2016

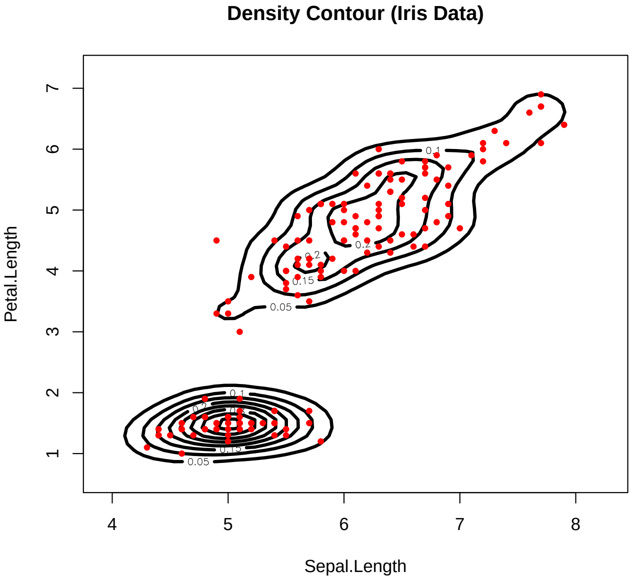
Examples of Density Estimation – 1



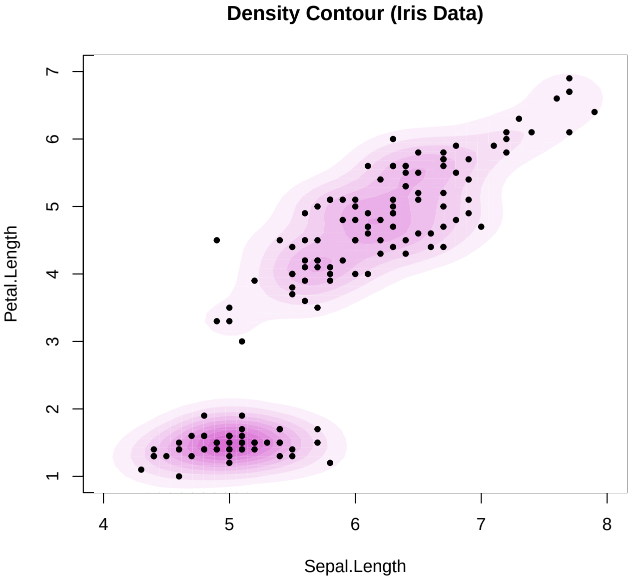
Examples of Density Estimation – 2



Examples of Density Estimation – 3



Examples of Density Estimation – 4



Density Estimation: Overview

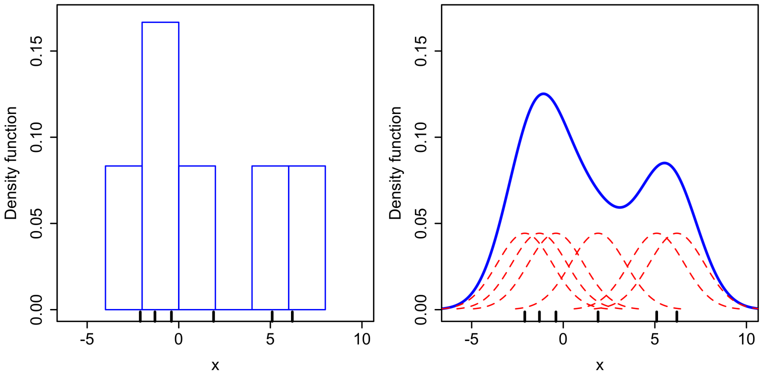
- We has seen a method for density estimation: histogram.
- Today, we will introduce another method: the *kernel density estimator (KDE)*.
- A feature of the KDE is that it will generate a smooth density function.
- The red curve in the second plot and the density contours in the previous plots are all from the KDE.
- And we will also talk about an important concept: *bias-variance tradeoff*.
- The bias-variance tradeoff is related to 'how to choose the bin size' for histogram.

Kernel Density Estimator – 1

- Assume our data is a random sample $X_1, \cdots, X_n \in \mathbb{R}$ from a density function p.
- The goal of density estimation is to find an estimator to the function p.
- Here we will use the KDE, which is defined as

$$\hat{p}_n(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right).$$

- $h > 0$ is a positive constant called the *smoothing bandwidth*.
- h is just like the bin size in the histogram.
- The function $K(x)$ is called the *kernel function*.
- In most cases, we choose $K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ (the Gaussian kernel).
- You can just view $\hat{p}_n$ as replacing each of the data points by a small Gaussian bump and aggregate these bumps.



Credit: wikipedia.

Kernel Density Estimator – 3

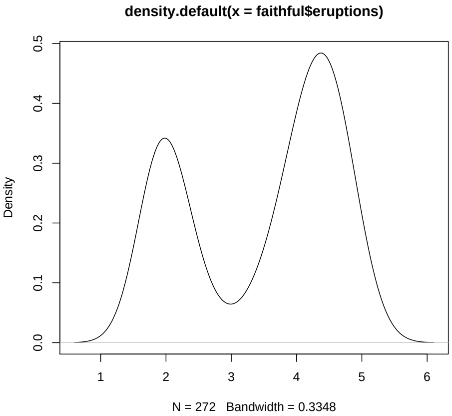
To compute the KDE in one dimensional case, we use the function `density()`.

```
> den<- density(faithful$eruptions)
> names(den)
[1] "x"          "y"          "bw"          "n"
"call"
[6] "data.name" "has.na"
```

- `x`: position of grid points.
- `y`: density value at each grid point.
- `bw`: smoothing bandwidth, the parameter h.

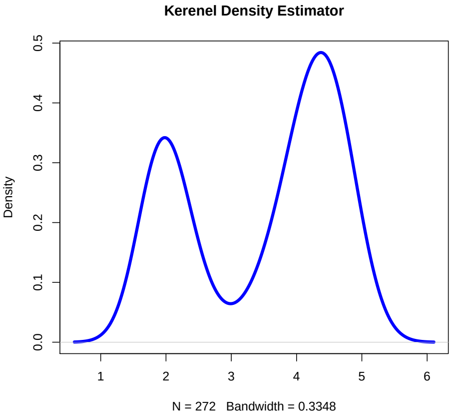
Kernel Density Estimator – 4

```
> plot(den)
```



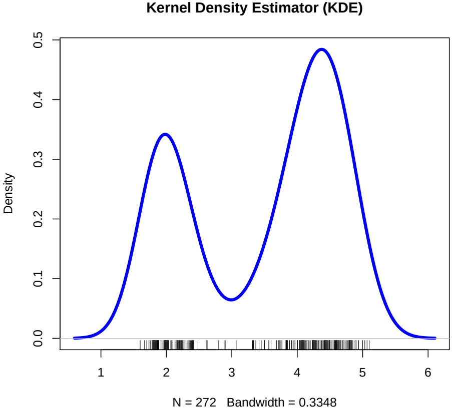
Kernel Density Estimator – 5

```
> plot(den, lwd=4, col="blue",
+        main="Kerenel Density Estimator")
```



You can use the arguments in the scatter plot here.

```
> rug(faithful$eruptions)
```



`rug()`: adding the rug to show data points.

Kernel Density Estimator – 7

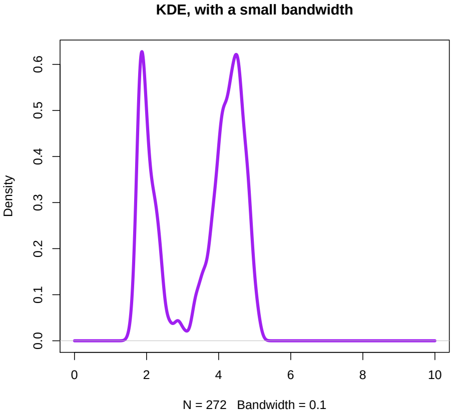
```
> den1<- density(faithful$eruptions, bw=0.1,
+                    n=1024, from=0, to=10)
```

The inputs are as follows:

- `bw`: the smoothing bandwidth (we can choose).
- `n`: total number of grid points.
- `from`: the starting point of the grid.
- `to`: the end point of the grid.

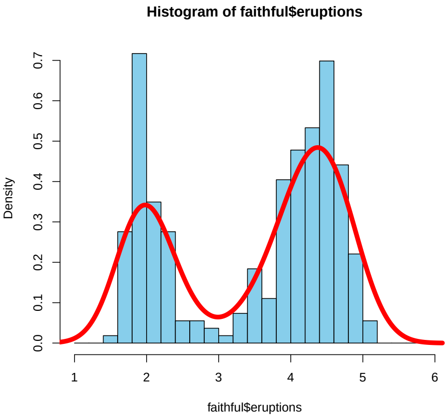
Kernel Density Estimator – 8

```
> plot(den1, main="KDE, with a small bandwidth",
+        lwd=4, col="purple")
```



Adding Density Curves – 1

```
> hist(faithful$eruptions, col="skyblue",
+        probability=T,
+        breaks=seq(from=1, to=6, by=0.2))
> lines(den, lwd=6, col="red")
```

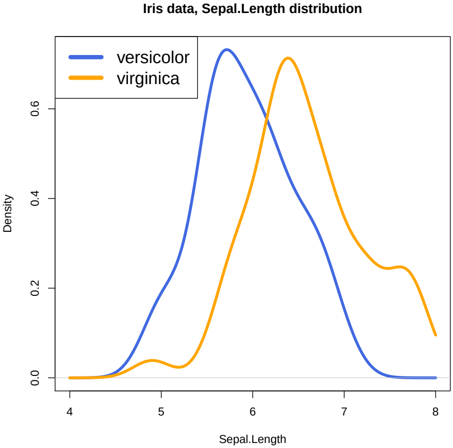


Adding Density Curves – 2

It can be applied to two-sample comparison.

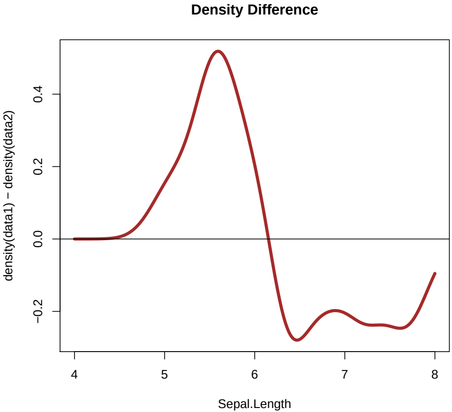
```
> data1 <- iris$Sepal.Length[iris$Species=="versicolor"]
> data2 <- iris$Sepal.Length[iris$Species=="virginica"]
> data1_den <- density(data1, from=4, to=8)
> data2_den <- density(data2, from=4, to=8)
>
> plot(data1_den, col="royalblue",lwd=4,
+      main="Iris data, Sepal.Length distribution",
+      xlab="Sepal.Length")
> lines(data2_den, col="orange",lwd=4)
> legend("topleft",c("versicolor","virginica"),
+        col=c("royalblue","orange"), lwd=6, cex=1.5)
```

Adding Density Curves – 3



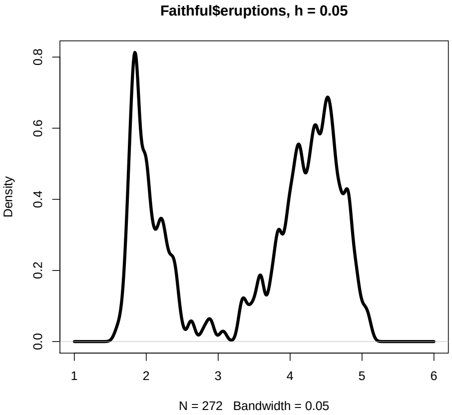
Adding Density Curves – 4

```
> plot(x=data1_den$x,
+       y=data1_den$y-data2_den$y,
+       col="brown",lwd=4,
+       main="Density Difference",
+       xlab="Sepal.Length",
+       type="l", ylab="density(data1) - density(data2)")
> abline(h=0)
```



Smoothing Bandwidth: Undersmoothing

```
> plot(density(faithful$eruptions, bw=0.05,
+                from=1, to=6), lwd=4,
+        main="Faithful$eruptions, h = 0.05")
```

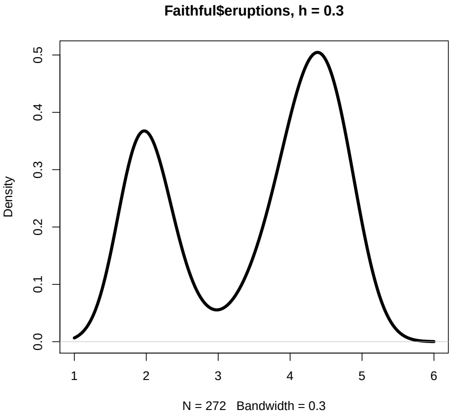


Smoothing Bandwidth: Oversmoothing

```
> plot(density(faithful$eruptions, bw=1,
+                  from=1, to=6), lwd=4,
+        main="Faithful$eruptions, h = 1")
```

Smoothing Bandwidth: Right Amount

```
> plot(density(faithful$eruptions, bw=0.3,
+                  from=1, to=6), lwd=4,
+       main="Faithful$eruptions, h = 0.3")
```



Smoothing Bandwidth and Bias-Variance Tradeoff

- The smoothing bandwidth h matters a lot!
- If you do not specify h, R will automatically choose it for you.
- However, this is still an unsolved problem in statistics.
- There are many new papers about how to choose h; this is known as the *bandwidth selection problem*.
- The phenomena we just observe–undersmoothing and oversmoothing–are related to the so-called *bias-variance tradeoff*.
- When h is small, the variance is large but the bias is small.
- When h is large, the variance is small but the bias is large.

Undersmoothing: Large Variance, Small Bias

Try the following commands:

```
> x_seq <- seq(from=-3, to=3, length.out=500)
> d_seq <- dnorm(x_seq)
> for(j in 1:30){
+   plot(x=x_seq, y=d_seq, type="l", lwd=4,
+        ylim=c(0,0.5), ylab="Density", xlab="X",
+        main="n=1000, h=0.1", col="blue")
+   abline(h=0, col="gray")
+   lines(density(rnorm(1000), bw=0.1,
+         from=-3, to=3), lwd=2, col="red")
+   Sys.sleep(1)
+ }
```

- Blue curve: the theoretical density curve.
- Red curve: the estimated density curve.
- You would see that the red curve fluctuates a lot (large variance).
- However, at least these red curves are fluctuates around the blue curve (small bias).

Oversmoothing: Small Variance, Large Bias

Try the following commands:

```
> x_seq <- seq(from=-3, to=3, length.out=500)
> d_seq <- dnorm(x_seq)
> for(j in 1:30){
+   plot(x=x_seq, y=d_seq, type="l", lwd=4,
+        ylim=c(0,0.5), ylab="Density", xlab="X",
+        main="n=1000, h=1", col="blue")
+   abline(h=0, col="gray")
+   lines(density(rnorm(1000), bw=1,
+                 from=-3, to=3), lwd=2, col="red")
+   Sys.sleep(1)
+ }
```

- Blue curve: the theoretical density curve.
- Red curve: the estimated density curve.
- Now, you would see that the red curve is very stable (small variance).
- However, the red curve deviates a lot from the blue curve (large bias).

Bias-Variance Tradeoff: Comments

- The concept of bias-variance tradeoff appears in many modern statistical problem.
- The basic idea is, if we try to fit a complex model (small h), we have a small bias but suffers from large variance.
- On the hand, if we fix a simple model (large h), we have a large bias but small variance.
- Also related to another problem called 'model selection'.
- One example of model selection is 'in the multiple regression, how to select the variables?'

Kernel Density Estimator in 2D

- Now we discuss how to use the KDE in bivariate case.
- We will use the library `KernSmooth`.
- You may need to install this library first:

```
install.packages("KernSmooth", dependencies = T)
```

- Now execute `library(KernSmooth)` to include this library.
- The 2D KDE is the function `bkde2D()`.

`bkde2D()`: 2D Kernel Density Estimator

```
> data1 <- cbind(iris$Sepal.Length, iris$Petal.Length)
> iris_kde <- bkde2D(data1, bandwidth = 0.25,
+                   gridsize = c(101,101),
+                   range.x=list(c(4,8),c(1,7)))
```

Input variables:

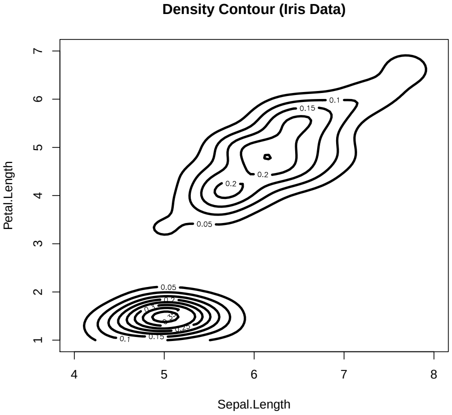
- `bandwidth`: smoothing bandwidth.
- `gridsize`: the size of the 2D grid.
- `range.x`: a list consists of range of two variables.

```
> names(iris_kde)
[1] "x1"   "x2"   "fhat"
```

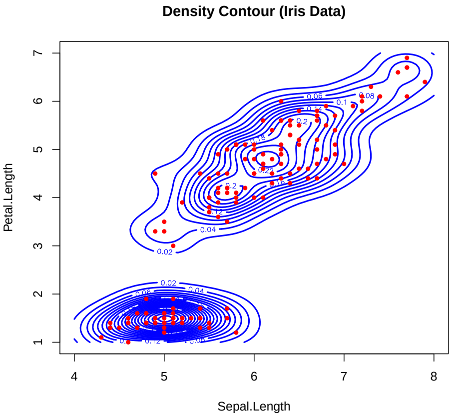
Output–a list consists of the following variables:

- `x1, x2`: the position of grids.
- `fhat`: a matrix consists of estimated density on the grid.

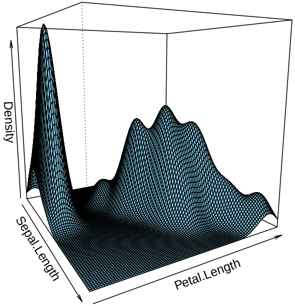
```
> contour(x=iris_kde$x1,y=iris_kde$x2,
+           z=iris_kde$fhat, lwd=3,
+           main="Density Contour (Iris Data)",
+           xlab="Sepal.Length", ylab="Petal.Length")
```



```
> contour(x=iris_kde$x1,y=iris_kde$x2,
+         z=iris_kde$fhat, lwd=2,
+         main="Density Contour (Iris Data)",
+         xlab="Sepal.Length", ylab="Petal.Length",
+         nlevels=20, col=c("blue"))
> points(data1, col="red",pch=20)
```



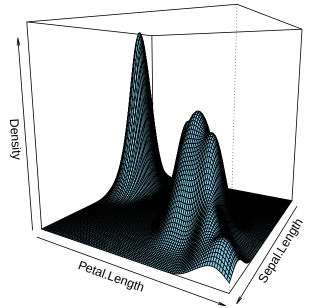
```
> persp(x=iris_kde$x1,y=iris_kde$x2,
+           z=iris_kde$fhat, col="skyblue",
+        xlab="Sepal.Length", ylab="Petal.Length",
+        zlab="Density", theta=60, phi=10)
```



Changes `theta` will rotate the plot.

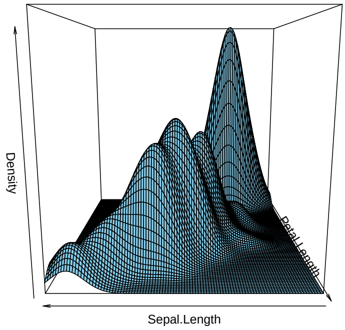
persp(): Making a Perpective Plot – 2

```
> persp(x=iris_kde$x1,y=iris_kde$x2,
+           z=iris_kde$fhat, col="skyblue",
+        xlab="Sepal.Length", ylab="Petal.Length",
+        zlab="Density", theta=120, phi=10)
```



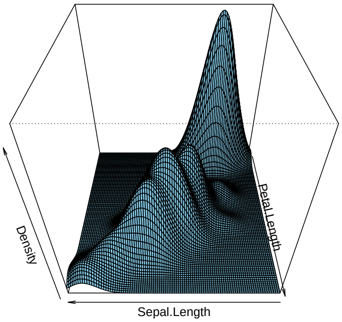
persp(): Making a Perpective Plot – 3

```
> persp(x=iris_kde$x1,y=iris_kde$x2,
+         z=iris_kde$fhat, col="skyblue",
+       xlab="Sepal.Length", ylab="Petal.Length",
+       zlab="Density", theta=180, phi=10)
```



persp(): Making a Perpective Plot – 4

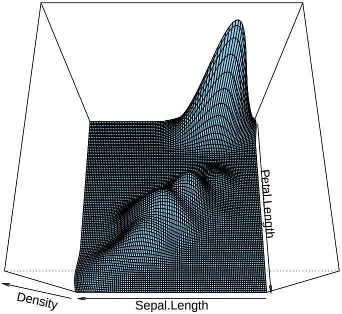
```
> persp(x=iris_kde$x1,y=iris_kde$x2,
+           z=iris_kde$fhat, col="skyblue",
+        xlab="Sepal.Length", ylab="Petal.Length",
+        zlab="Density", theta=180, phi=40)
```



`phi`: colatitude.

persp(): Making a Perpective Plot – 5

```
> persp(x=iris_kde$x1,y=iris_kde$x2,
+           z=iris_kde$fhat, col="skyblue",
+         xlab="Sepal.Length", ylab="Petal.Length",
+         zlab="Density", theta=180, phi=70)
```



persp(): Making a Perpective Plot – 6

Try the following command:

```
> for(w in (1:36)*10){
+   persp(x=iris_kde$x1,y=iris_kde$x2,
+         z=iris_kde$fhat, col="skyblue",
+         xlab="Sepal.Length", ylab="Petal.Length",
+         zlab="Density", theta=w, phi=40)
+   Sys.sleep(1)
+ }
```

You should see a rotating perspective plot; this shows the effect of `theta`.

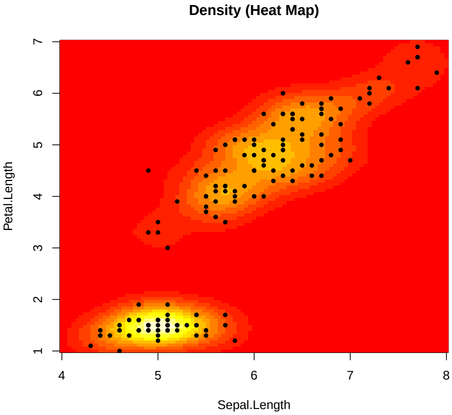
`persp()`: Making a Perpective Plot – 7

Try the following command:

```
> for(w in c((0:9)*10, (9:0)*10)){
+   persp(x=iris_kde$x1,y=iris_kde$x2,
+         z=iris_kde$fhat, col="skyblue",
+         xlab="Sepal.Length", ylab="Petal.Length",
+         zlab="Density", theta=10, phi=w)
+   Sys.sleep(1)
+ }
```

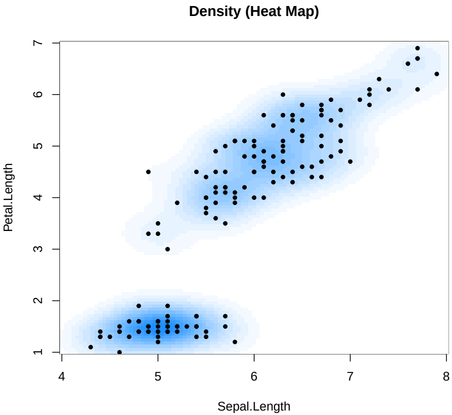
You should see a perspective plot from different colatitute.

```
> image(x=iris_kde$x1,y=iris_kde$x2,
+        z=iris_kde$fhat, xlab="Sepal.Length",
+        ylab="Petal.Length", main="Density (Heat Map)")
> points(data1, pch=20)
```



image(): Change Color

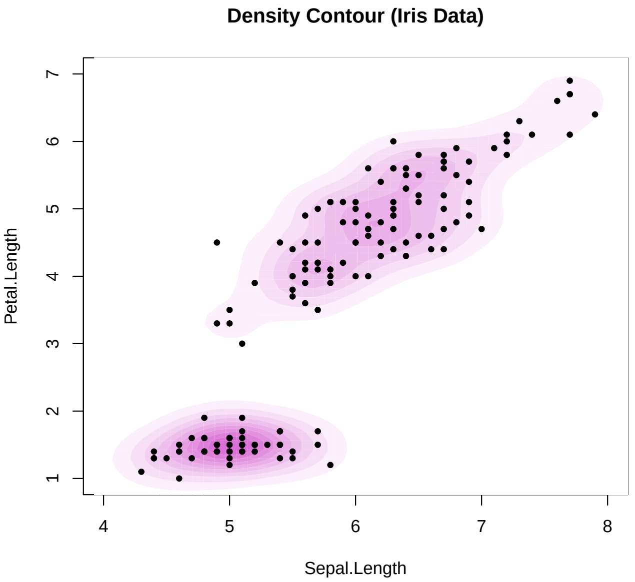
```
> colP = colorRampPalette(c("white","dodgerblue"))
> image(x=iris_kde$x1,y=iris_kde$x2,
+         z=iris_kde$fhat, xlab="Sepal.Length",
+         ylab="Petal.Length", main="Density (Heat Map)",
+         col=colP(20))
> points(data1, pch=20)
```



To add a color contour to a scatter plot, we will use the function `.filled.contour()`.

```
> col_tmp <- colorRampPalette(c("white","orchid"))(10)
> level_tmp <- (0:10)/10*max(c(iris_kde$fhat))
>   # defining the levels and the corresponding colors
> plot(NULL, xlim=c(4,8), ylim=c(1,7),
+      main="Density Contour (Iris Data)",
+      xlab="Sepal.Length", ylab="Petal.Length")
>   # this makes an empty plot
> .filled.contour(x=iris_kde$x1,y=iris_kde$x2,
+                 z=iris_kde$fhat,
+                 levels=level_tmp,
+                 col=col_tmp)
>   # this filled in the colored contours
> points(data1, col="black",pch=20)
>   # finally we add data points
```

Adding Colored Contours to a Scatter Plot – 2



In-class Exercises

- Try to plot the density curve of `faithful$waiting` using function `density()`.
- Now plot the histogram of `faithful$waiting` first and the add the density curve to it.
- Change the smoothing bandwidth `bw` to 1, 5, and 10 and check how the density curve changes.