

Lab 8

Karen Lopez

11:59PM April 7, 2019

Load the `ggplot2` library and its dataset called `mpg`. Print out a summary of the dataset using `summary` and `str`.

```
pacman::p_load(ggplot2)
```

```
data(mpg)
```

```
mpg$drv = factor(mpg$drv)
```

```
mpg$manufacturer = factor(mpg$manufacturer)
```

```
mpg$model = factor(mpg$model)
```

```
mpg$fl = factor(mpg$fl)
```

```
mpg$class = factor(mpg$class)
```

```
mpg$cyl = factor(mpg$cyl)
```

```
summary(mpg)
```

```
##      manufacturer      model      displ      year
## dodge      :37  caravan 2wd      : 11  Min.   :1.600  Min.   :1999
## toyota      :34  ram 1500 pickup 4wd: 10  1st Qu.:2.400  1st Qu.:1999
## volkswagen:27  civic      : 9  Median :3.300  Median :2004
## ford        :25  dakota pickup 4wd : 9  Mean    :3.472  Mean    :2004
## chevrolet   :19  jetta      : 9  3rd Qu.:4.600  3rd Qu.:2008
## audi        :18  mustang     : 9  Max.    :7.000  Max.    :2008
## (Other)     :74  (Other)      :177
## cyl      trans      drv      cty      hwy      fl
## 4:81  Length:234      4:103  Min.   : 9.00  Min.   :12.00  c: 1
## 5: 4   Class :character f:106  1st Qu.:14.00  1st Qu.:18.00  d: 5
## 6:79  Mode  :character  r: 25  Median :17.00  Median :24.00  e: 8
## 8:70                                     Mean    :16.86  Mean    :23.44  p: 52
##                                     3rd Qu.:19.00  3rd Qu.:27.00  r:168
##                                     Max.    :35.00  Max.    :44.00
##
##      class
## 2seater   : 5
## compact  :47
## midsize   :41
## minivan   :11
## pickup    :33
## subcompact:35
## suv       :62
```

```
?mpg
```

```
str(mpg)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame': 234 obs. of 11 variables:
## $ manufacturer: Factor w/ 15 levels "audi","chevrolet",...: 1 1 1 1 1 1 1 1 1 ...
## $ model       : Factor w/ 38 levels "4runner 4wd",...: 2 2 2 2 2 2 3 3 3 ...
## $ displ      : num 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
## $ year       : int 1999 1999 2008 2008 2008 1999 1999 2008 1999 2008 ...
## $ cyl        : Factor w/ 4 levels "4","5","6","8": 1 1 1 1 3 3 3 1 1 1 ...
## $ trans      : chr "auto(15)" "manual(m5)" "manual(m6)" "auto(av)" ...
```

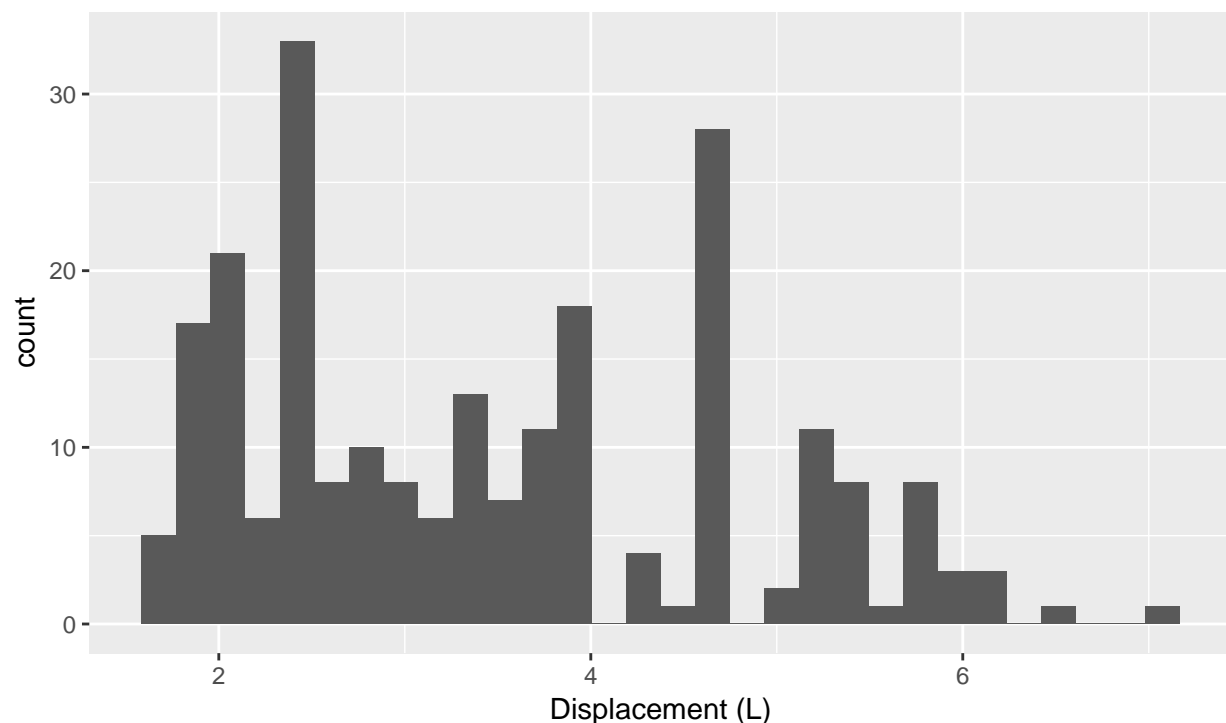
```
## $ drv      : Factor w/ 3 levels "4","f","r": 2 2 2 2 2 2 2 1 1 1 ...
## $ cty      : int   18 21 20 21 16 18 18 18 16 20 ...
## $ hwy      : int   29 29 31 30 26 26 27 26 25 28 ...
## $ fl       : Factor w/ 5 levels "c","d","e","p",...: 4 4 4 4 4 4 4 4 4 4 ...
## $ class    : Factor w/ 7 levels "2seater","compact",...: 2 2 2 2 2 2 2 2 2 2 ...
```

Visualize a histogram then a density estimate of the `displ` variable, the engine displacement. Use `labs` to create a title, subtitle, caption and x-label via `x` and y-label via `y`. Do this for every single illustration in this lab.

```
ggplot(mpg) +
  aes(displ) +
  geom_histogram() +
  labs(title = "Engine Displacement Histogram" , subtitle = "", x = "Displacement (L)", caption = "Source: EPA 2008 Fuel Economy Dataset")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

Engine Displacement Histogram



Source: EPA 2008 Fuel Economy Dataset

```
ggplot(mpg) +
  aes(displ) +
  geom_density(fill = "blue", alpha = 0.2, col = "white") +
  labs(title = "Engine Displacement Density" , subtitle = "", x = "Displacement (L)", caption = "Source: EPA 2008 Fuel Economy Dataset")
```

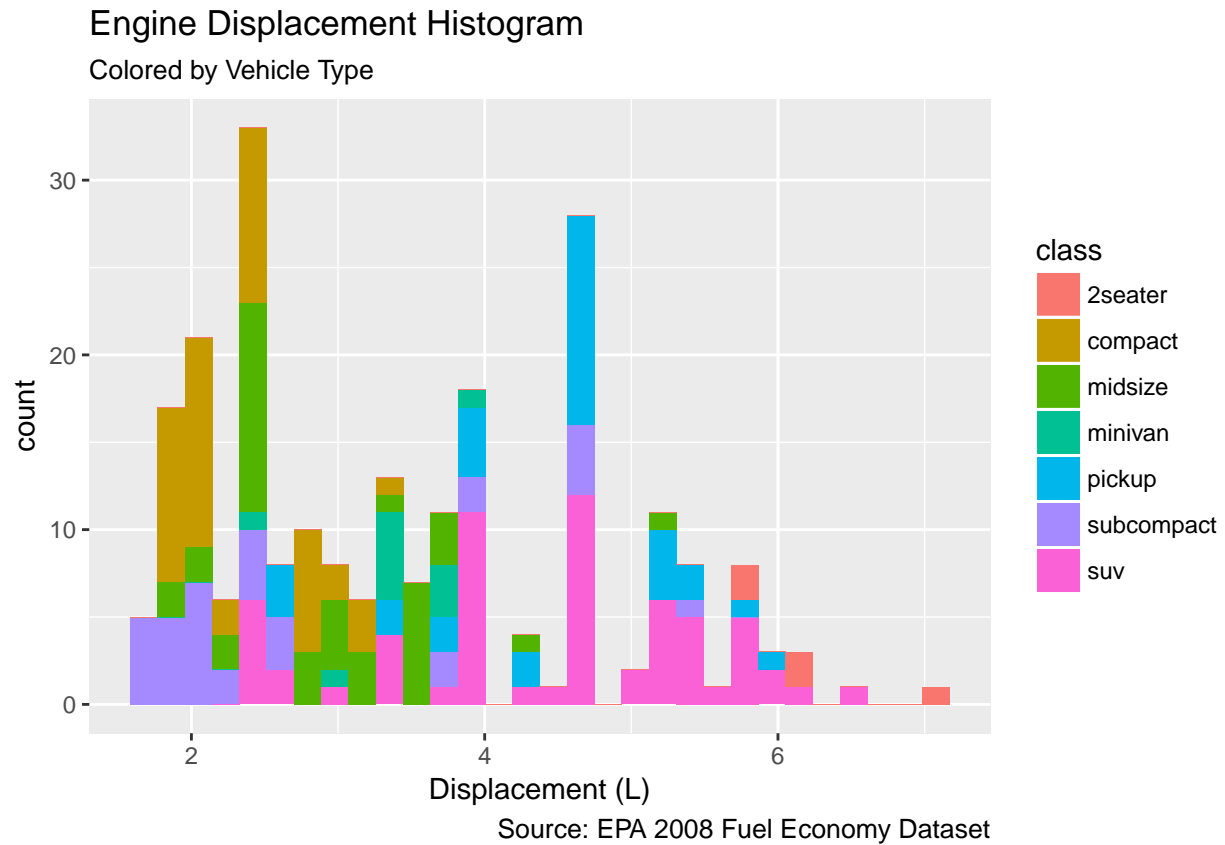
Engine Displacement Density



Source: EPA 2008 Fuel Economy Dataset

Visualize a histogram the `displ` variable, but then fill the color of the bar by the `class` of the car. You will have to pass `class` in as the `fill` in the aesthetic of the histogram.

```
ggplot(mpg) +  
  aes(displ) +  
  geom_histogram(aes(fill = class)) +  
  labs(title = "Engine Displacement Histogram" , subtitle = "Colored by Vehicle Type", x = "Displacement (L)")  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

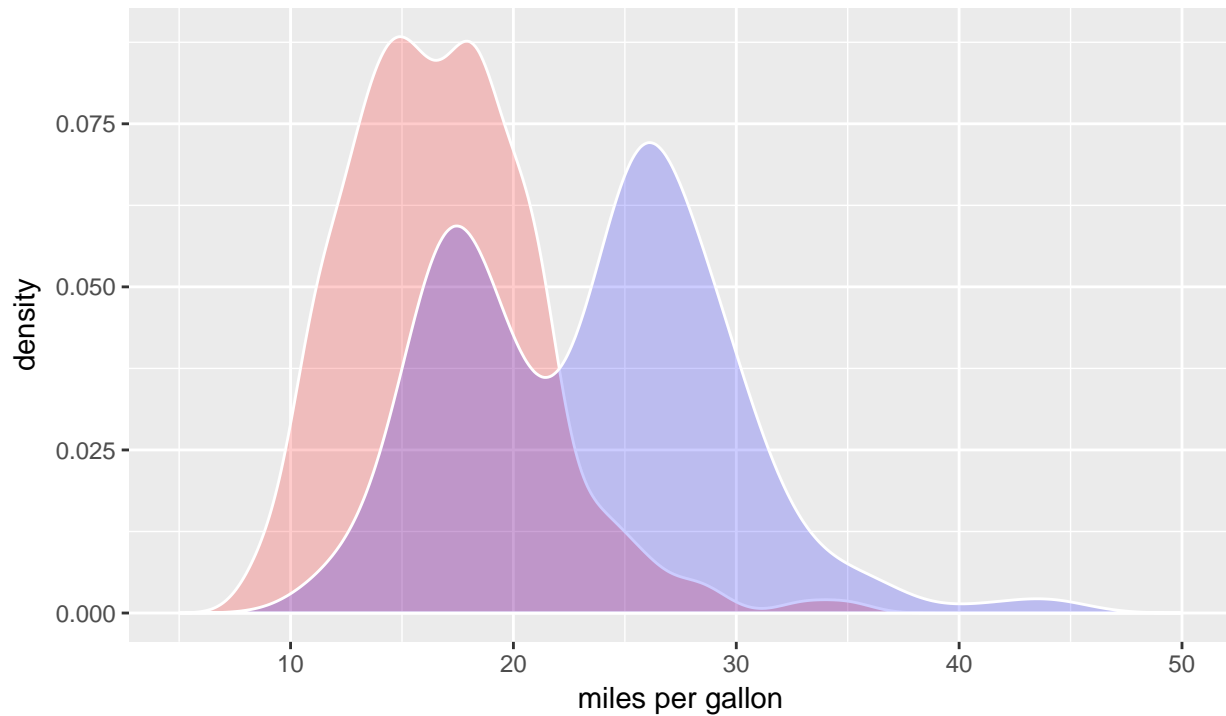


Visualize overlapping densities of `cty` (city miles per gallon) and `hwy` (highway miles per gallon) using two colors with an alpha blend.

```
ggplot(mpg) +
  geom_density(aes(cty), fill = "red", col = "white", alpha = 0.2) +
  geom_density(aes(hwy), fill = "blue", col = "white", alpha = 0.2) +
  xlim(5, 50) +
  labs(title = "Fuel Efficiency Density", subtitle = "City in Red and Highway in Blue", x = "miles per gallon")
```

Fuel Efficiency Density

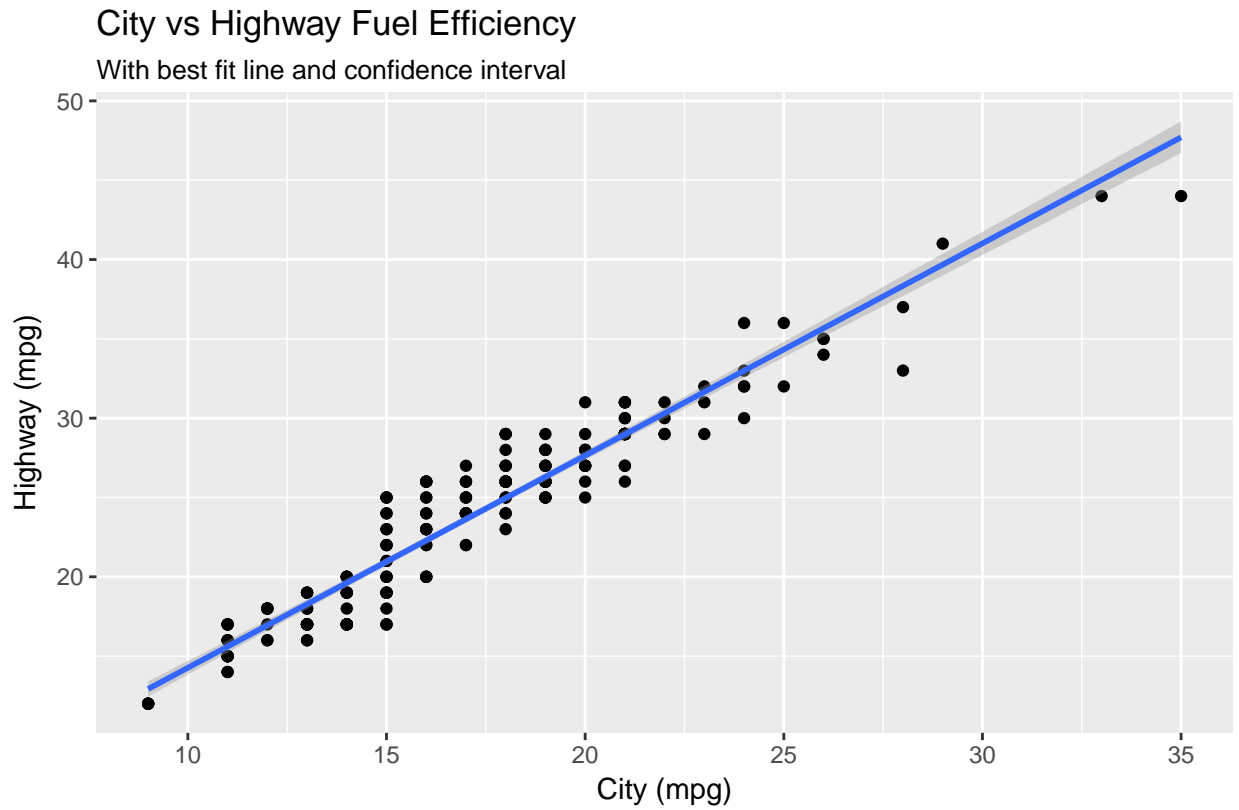
City in Red and Highway in Blue



Source: EPA 2008 Fuel Economy Dataset

Plot `cty` (city miles per gallon) vs `hwy` (highway miles per gallon) and draw a best fit line with a confidence region of that line.

```
ggplot(mpg, aes(x = cty, y = hwy)) +  
  geom_point() +  
  geom_smooth(method = "lm") +  
  labs(title = "City vs Highway Fuel Efficiency", subtitle = "With best fit line and confidence interval")
```



Source: EPA 2008 Fuel Economy Dataset

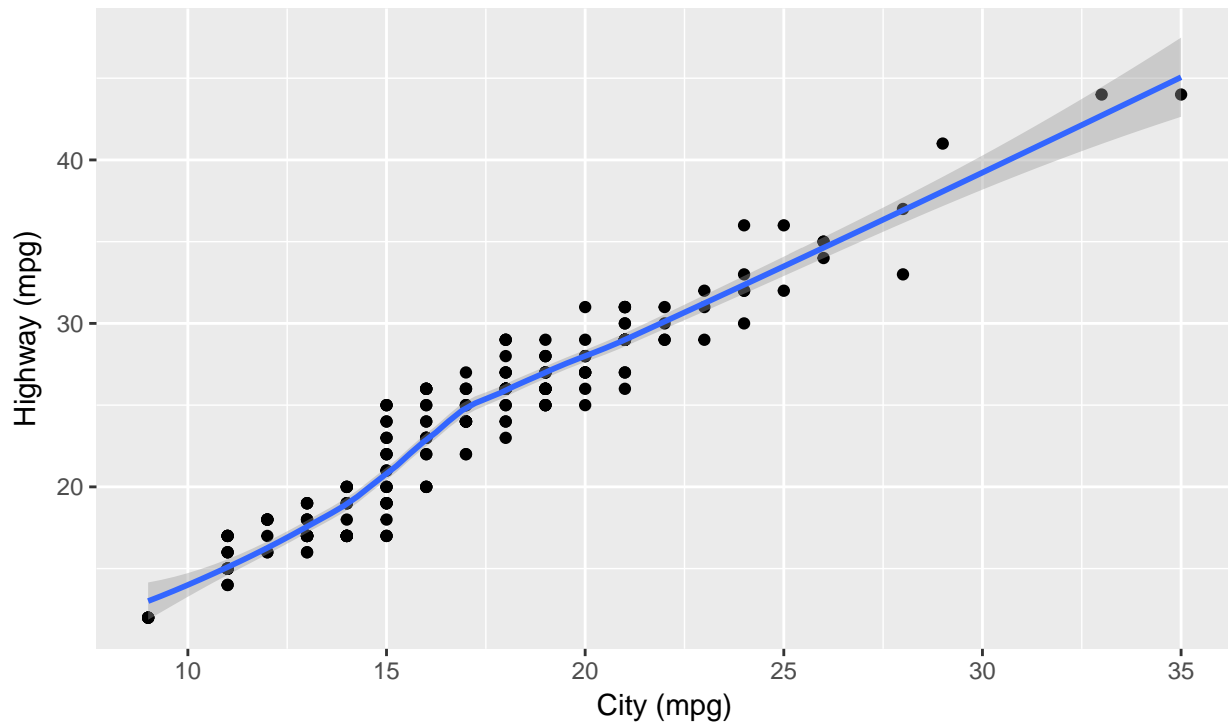
Plot `cty` (city miles per gallon) vs `hwy` (highway miles per gallon) and draw a best fit non-parametric functional relationship with a confidence region of that relationship.

#why is it such a nice correlation? if you have a full efficient engine youre gonna be good in the city

```
ggplot(mpg, aes(x = cty, y = hwy)) +  
  geom_point() +  
  geom_smooth() +  
  labs(title = "City vs Highway Fuel Efficiency", subtitle = "With best fit line and confidence interval")  
  
## `geom_smooth()` using method = 'loess'
```

City vs Highway Fuel Efficiency

With best fit line and confidence interval

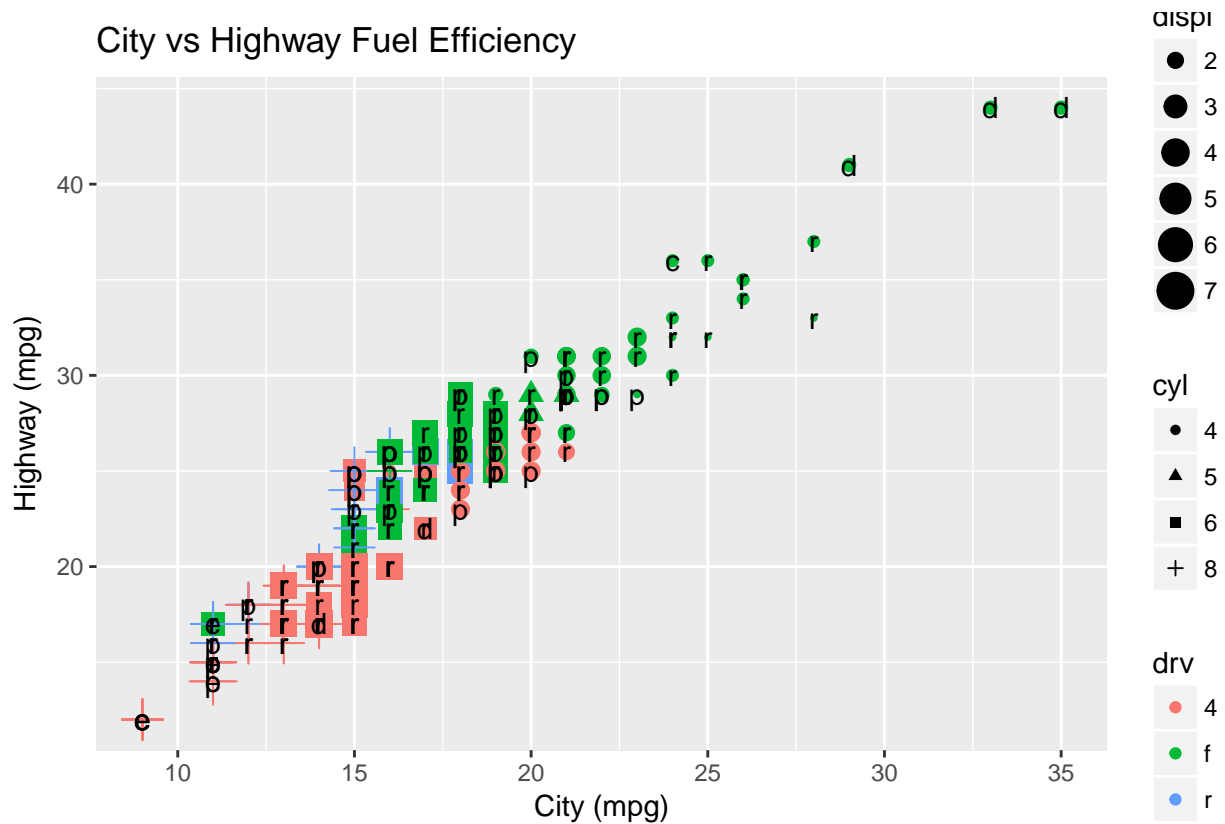


Source: EPA 2008 Fuel Economy Dataset

Plot `cty` (city miles per gallon) vs `hwy` (highway miles per gallon) and then try to visualize *as many other variables* as you can visualize effectively on the same plot. Try text, color, size, shape, etc.

#hypothesis set is richer than linear its using the spline model... this could be overfitting or the class

```
ggplot(mpg, aes(x = cty, y = hwy)) +  
  geom_point(aes(col = drv, shape = cyl, size = displ)) +  
  geom_text(aes(label = fl)) +  
  labs(title = "City vs Highway Fuel Efficiency", x = "City (mpg)", y = "Highway (mpg)", caption = "Sou
```



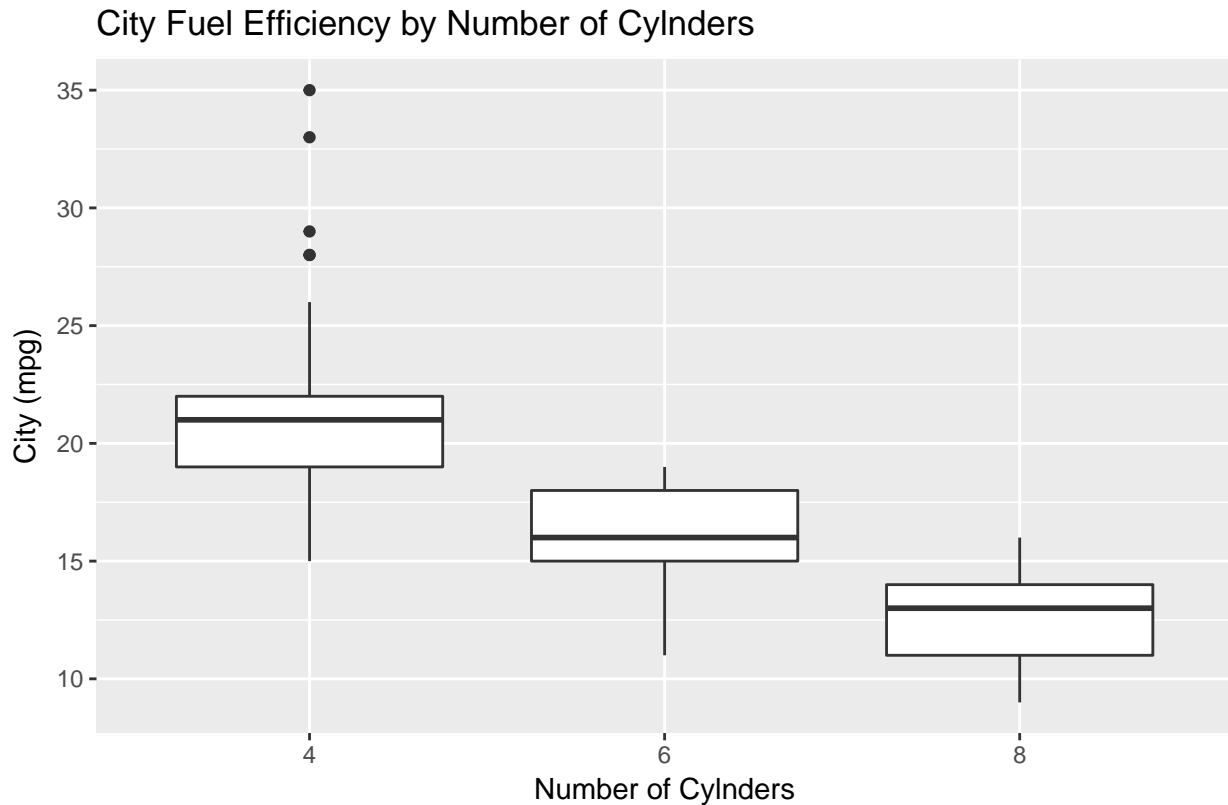
Source: EPA 2008 Fuel Economy Dataset

Convert `cyl` to an ordinal factor. Then use the package `dplyr` to retain only cars with 4, 6, 8 cylinders in the dataset. Then make a canonical illustration of `cty` by `cyl`.

```
mpg$cyl = factor(mpg$cyl, ordered = TRUE)
#mpg$cyl
pacman::p_load(dplyr)

mpg = mpg %>%
  filter(cyl %in% c(4, 6, 8))

ggplot(mpg, aes(x = cyl, y = cty)) +
  geom_boxplot() +
  labs(title = "City Fuel Efficiency by Number of Cylinders", x = "Number of Cylinders", y = "City (mpg)")
```

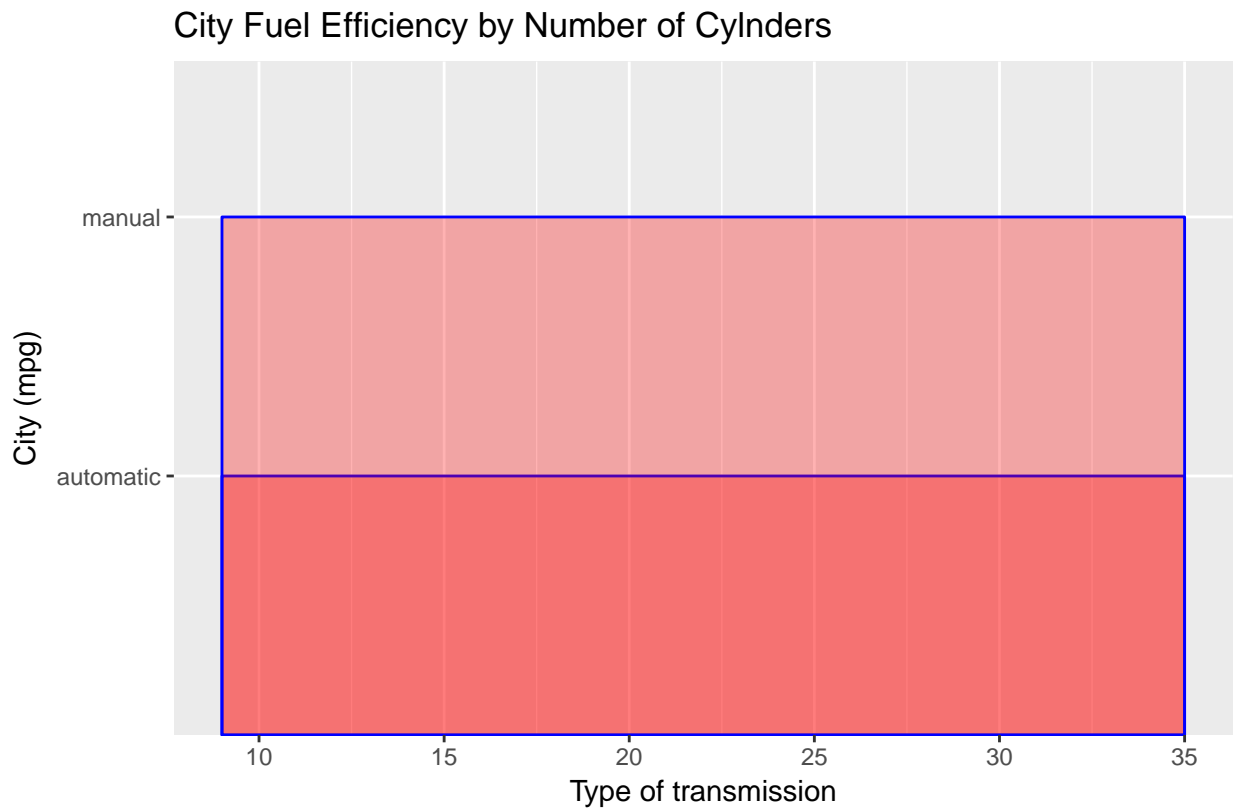



Load the `stringr` library. Use the `str_detect` function in this library to rewrite the `trans` variable in the data frame to be just “manual” or “automatic”.

```
pacman::p_load(stringr)
mpg$trans = ifelse(str_detect(mpg$trans, "^a"), 'automatic', 'manual')
```

Now visualize `cty` by `trans` via two overlapping alpha-blended densities.

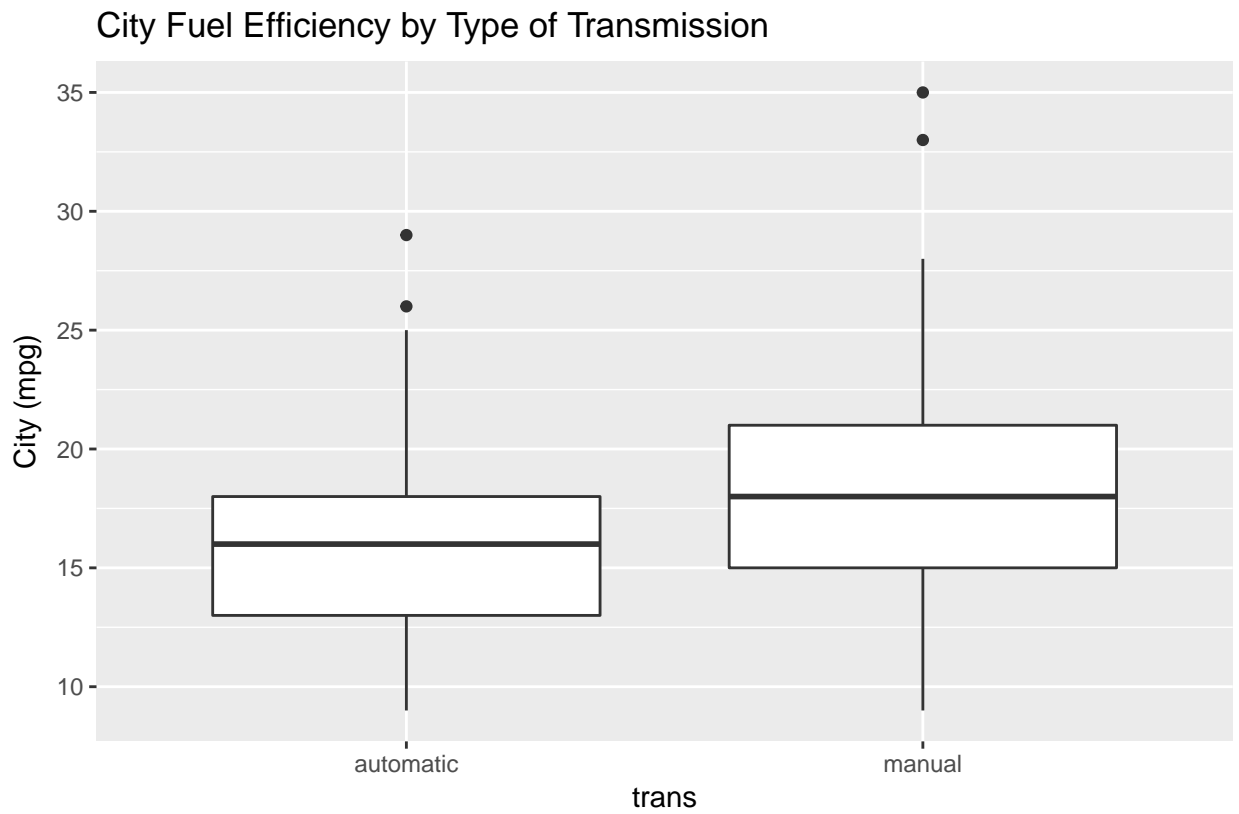
```
ggplot(mpg, aes(cty, trans)) +
  geom_density(aes(cty), fill = "red", col = "blue", alpha = 0.3) +
  labs(title = "City Fuel Efficiency by Number of Cylinders", x = "Type of transmission",
        y = "City (mpg)", caption = "Source: EPA 2008 Fuel Economy Dataset")
```



Source: EPA 2008 Fuel Economy Dataset

Now visualize `cty` by `trans` via a box and whisker plot.

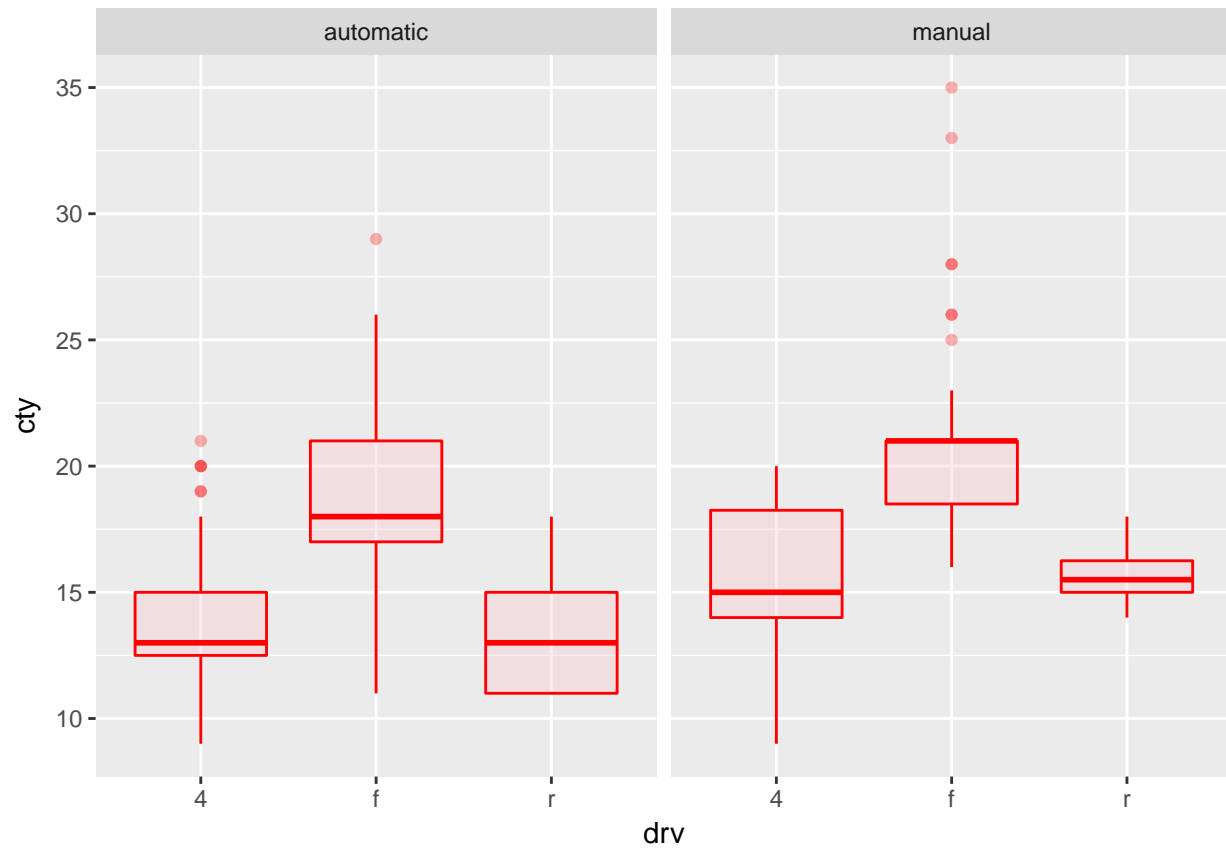
```
ggplot(mpg, aes(x = trans, y = cty)) +  
  geom_boxplot() +  
  labs(title = "City Fuel Efficiency by Type of Transmission", x = "trans", y = "City (mpg)", caption =
```



Source: EPA 2008 Fuel Economy Dataset

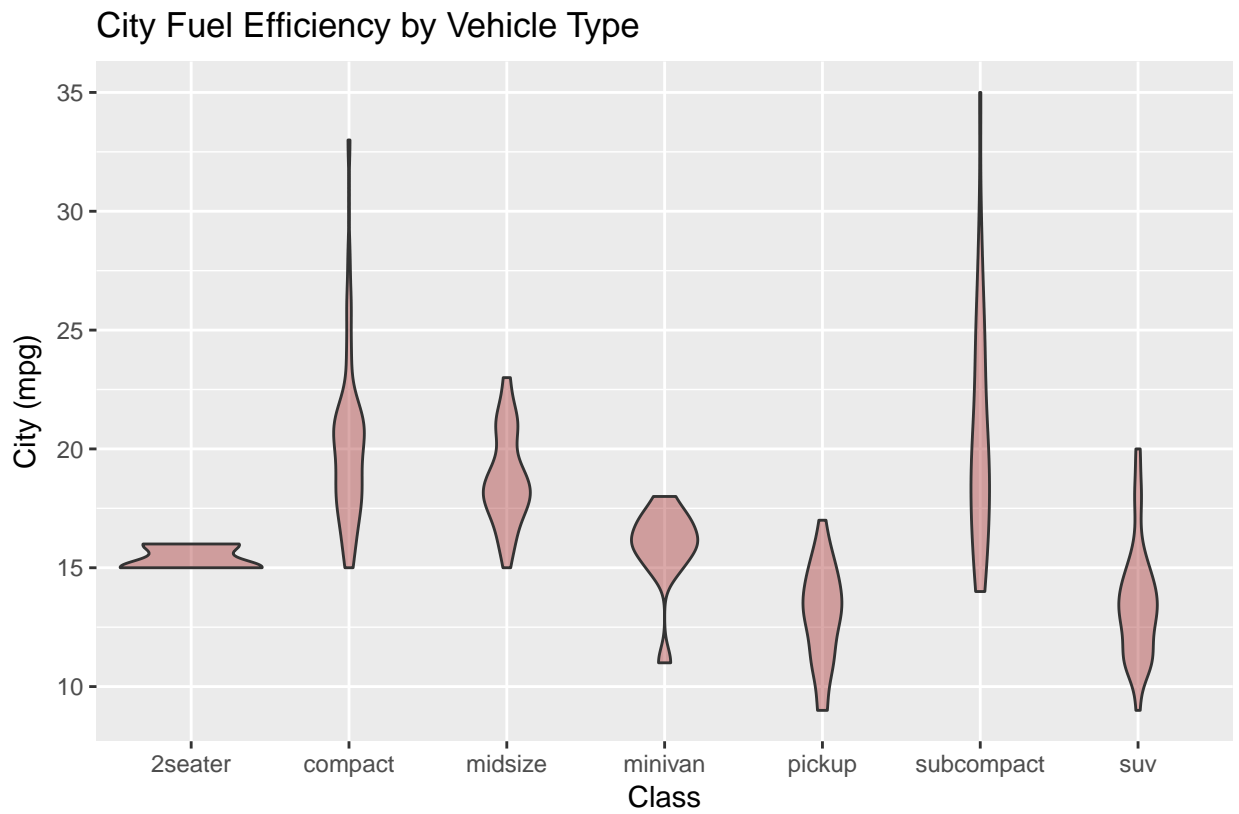
Now visualize cty by drv by trans via two box and whisker plots horizontally laid out.

```
ggplot(mpg, aes(x = drv, y = cty)) +  
  geom_boxplot(col = "red", fill = "pink", alpha = 0.3) +  
  facet_grid(. ~ trans)
```



Now visualize cty by class via a violin plot. Look at the ggplot cheatsheet!

```
ggplot(mpg, aes(x = class, y = cty)) +
  geom_violin(fill = "brown", alpha = 0.4) +
  labs(title = "City Fuel Efficiency by Vehicle Type", x = "Class", y = "City (mpg)", caption = "Source")
```



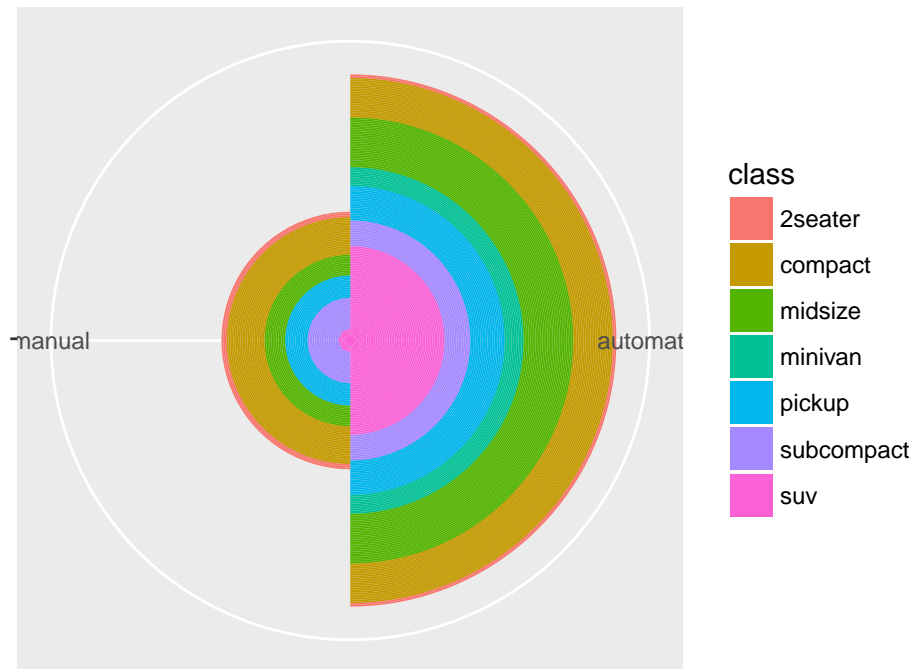
Source: EPA 2008 Fuel Economy Dataset

Make a pie chart of `class`.

Visualize `trans` vs `class`. Look at the ggplot cheatsheet!

```
ggplot(mpg, aes(x = trans , y = "", fill = class)) +  
  geom_bar(width = 1, stat = "identity") +  
  coord_polar("x", start = 0) +  
  labs(title = "Pie Chart of Transmission by Class" , subtitle = "", x = " ", y = " ", caption = "Source: EPA 2008 Fuel Economy Dataset")
```

Pie Chart of Transmission by Class



Source: EPA 2008 Fuel Economy Dataset

Using the package ggExtra's ggMarginal function, look at the hwy by year and plot the marginal density on both the x and y axes.

```
pacman::p_load(ggExtra)
```

```
## also installing the dependency 'miniUI'
```

```
##
```

```
## There are binary versions available but the source versions are
```

```
## later:
```

```
##      binary  source needs_compilation
```

```
## miniUI  0.1.1 0.1.1.1             FALSE
```

```
## ggExtra  0.7   0.8               FALSE
```

```
## installing the source packages 'miniUI', 'ggExtra'
```

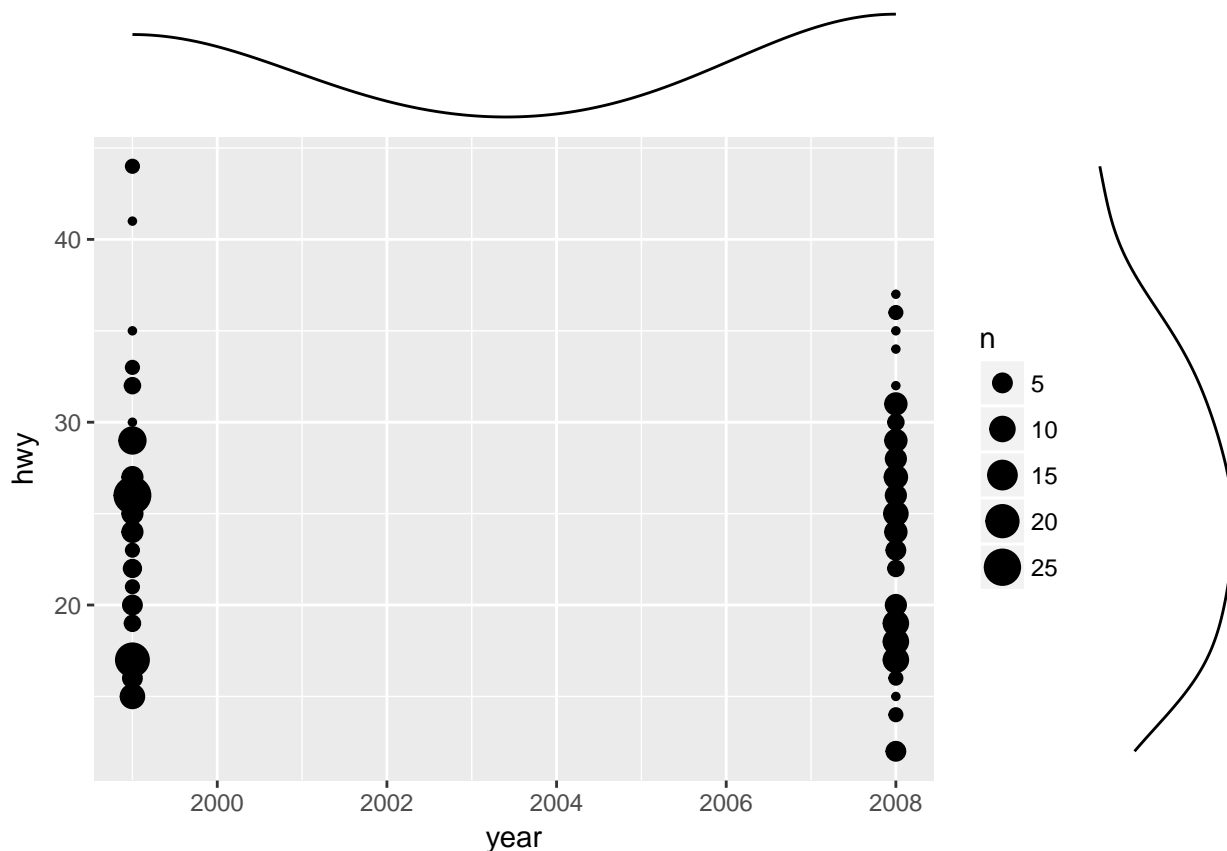
```
##
```

```
## ggExtra installed
```

```
g = ggplot(mpg, aes(year, hwy)) +
```

```
  geom_count()
```

```
ggMarginal(g, type = "density", fill = "transparent")
```



Using the package `ggcorrplot`'s `ggcorrplot` function, look at the correlations for all variables in this dataset that are legal in a correlogram. Use `dplyr` to `select_if` the variable is appropriate.

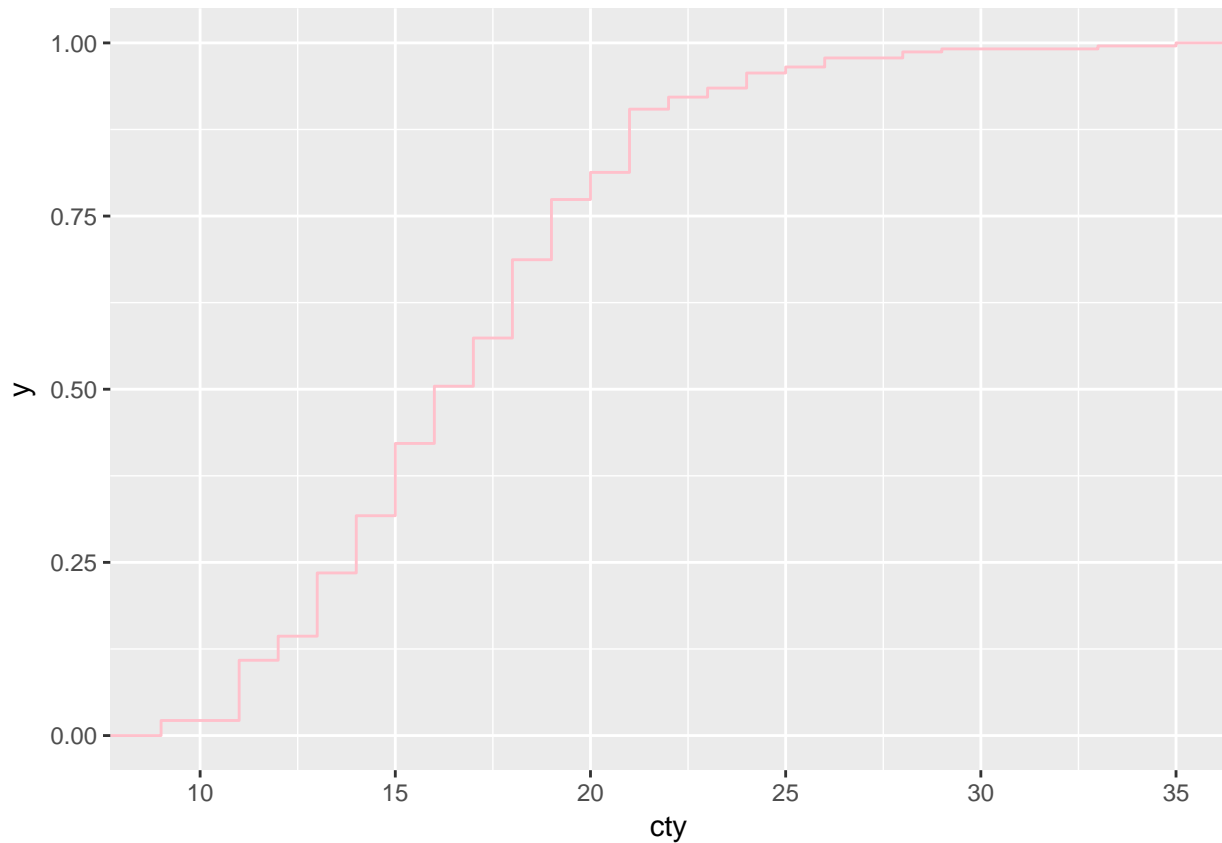
```
pacman::p_load(ggcorrplot)
```

```
##
##   There is a binary version available but the source version is
##   later:
##           binary source needs_compilation
## ggcorrplot 0.1.1 0.1.2                FALSE
## installing the source package 'ggcorrplot'
##
## ggcorrplot installed
```

```
library(ggcorrplot)
```

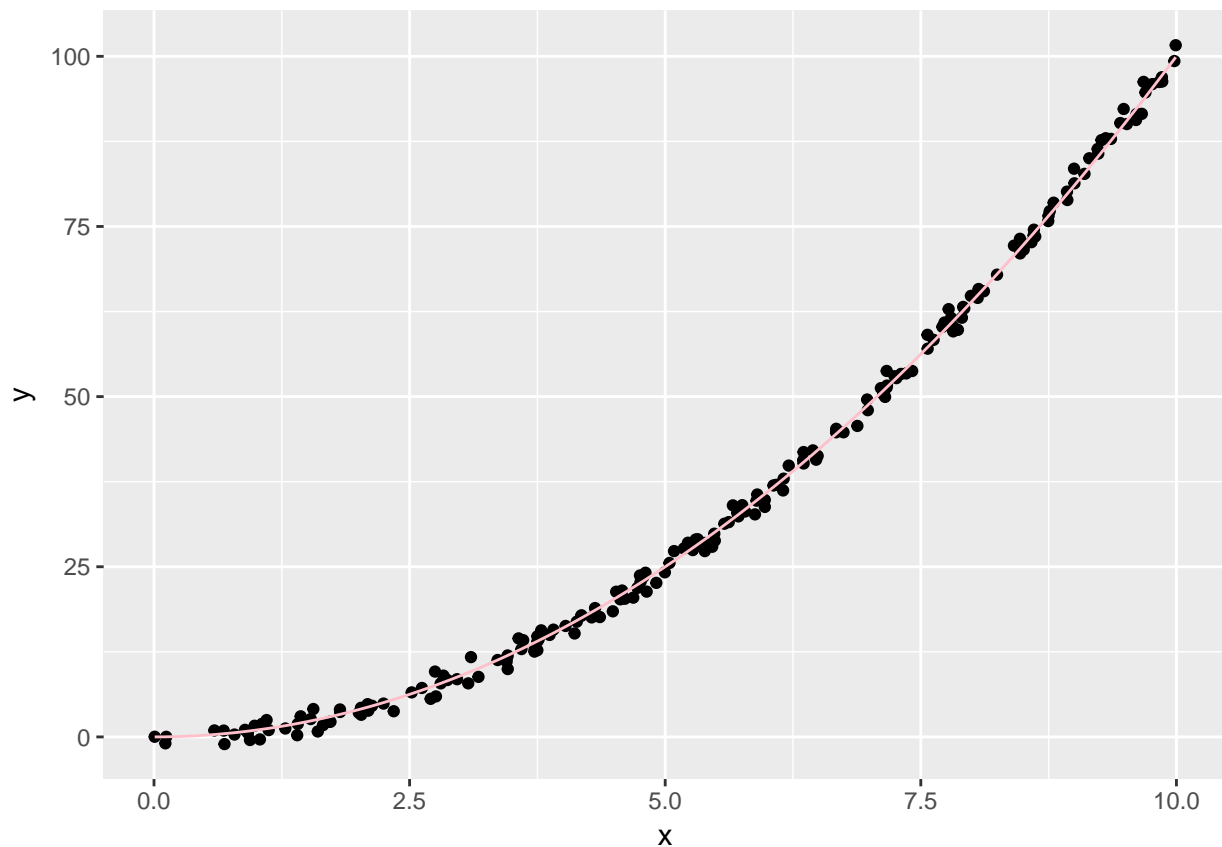
Use the `stat_ecdf` function to plot the estimated cumulative distribution of 'cty'.

```
ggplot(mpg, aes(cty)) +
  stat_ecdf(geom = "step", col = "pink")
```



Create a data generating process where x is uniform between 0 and 10 and y is x^2 plus $N(0, 1)$ noise. Plot $n = 200$ points and then plot the quadratic relationship $y = x^2$ using the function `stat_function`.

```
n = 200
x = runif(n, min = 0, max = 10)
noise = rnorm(n, 0, 1)
y = x^2 + noise
df = data.frame(x = x, y = y)
ggplot(df, aes(x, y)) +
  geom_point() +
  stat_function(fun = function(x){y=x^2}, col= "pink")
```

We now move to Rcpp. Load the library.

```
library(Rcpp)
```

Write an R function `is_odd` and a C++ function `is_odd_cpp` that evaluates if a number is odd and returns true if so.

```
is_odd = function(n){
  n %% 2 == 1
}
cppFunction('
  bool is_odd_cpp (int n){
    return n % 2 == 1;
  }
')
```

Using 'system.time', run both functions 1,000,000 times on the numbers 1, 2, ..., 1000000. Who is faster and by how much?

```
system.time(
  for(i in 1 : 1e6){
    is_odd(i)
  }
)
```

```
##   user system elapsed
##  1.053   0.009   1.077
```

```
system.time(
  for(i in 1 : 1e6){
```

```

    is_odd_cpp(i)
  }
)

```

```

##      user  system elapsed
##    3.015   0.269   3.320

```

Write an R function `fun` and a C++ function `fun_cpp` that takes a natural number n and returns n if n is 0 or 1 otherwise the result of the function on $n - 1$ and $n - 2$. This is the function that returns the n th Fibonacci number.

```

fun = function(n){
  if (n == 0 | n == 1) n
  fun (n - 2) + fun (n - 1)
}
cppFunction('
  int fun_cpp(int n){
    if(n == 0 || n == 1) return n;
    return fun_cpp( n - 2) + fun_cpp( n - 1 );
  }
')

```

Using ‘system.time’, run both functions on the numbers 1, 2, ..., 100. Who is faster and by how much?

Write an R function `logs` and a C++ function `logs_cpp` that takes a natural number n and returns an array of $\ln(1), \ln(2), \dots, \ln(n)$.

#TO-DO

Using ‘system.time’, run both functions on the numbers 1, 2, ..., 1000000. Who is faster and by how much?

#TO-DO

Write an R function `max_distances` and a C++ function `max_distances_cpp` that takes an $n \times p$ matrix X and returns an $n \times n$ matrix called D of NA’s where the upper triangular portion above the diagonal is the max distances between the elements of the i, j th rows of X .

```

max_distances = function(X){
  n = nrow(X)
  p = ncol(X)
  D = matrix(NA, nrow = n, ncol = n)
  for(i_1 in 1:(n - 1)){
    for(i_2 in (i_1 + 1):n){
      sqd_diff = 0
      for(j in 1:p){
        sqd_diff = sqd_diff + (X[i_1, j] - X[i_2, j])^2
      }
      D[i_1, i_2] = sqrt(sqd_diff)
    }
  }
  D
}
cppFunction('
NumericMatrix max_distances_cpp(NumericMatrix X) {
  int n = X.nrow();
  int p = X.ncol();
  NumericMatrix D(n, n);
  std::fill(D.begin(), D.end(), NA_REAL);
}
')

```

```

for (int i_1 = 0; i_1 < (n - 1); i_1++){
  for (int i_2 = i_1 + 1; i_2 < n; i_2++){
    int sqd_diff = 0;
    for (int j = 0; j < p; j++){
      sqd_diff += pow(X(i_1, j) - X(i_2, j), 2); //by default the cmath library in std is loaded
    }
    D(i_1, i_2) = sqrt(sqd_diff); //by default the cmath library in std is loaded
  }
}
return D;
}'
)

```

Create a matrix X of $n = 1000$ and $p = 20$ filled with iid $N(0, 1)$ realizations. Using 'system.time', calculate D using both functions. Who is faster and by how much? cpp is faster and by 2.5 seconds

```

n = 1000
p = 20
X = matrix(rnorm(n * p), nrow = n, ncol = p)
system.time(
  max_distances(X)
)

```

```

##      user  system elapsed
## 13.326   0.127  13.938

```

```

system.time(
  max_distances_cpp(X)
)

```

```

##      user  system elapsed
##   0.043   0.001   0.045

```