Exploratory Data Analysis: Template

Math 437 Spring 2023

2/15/2023

Let's walk through some steps of exploratory data analysis. We'll use the `Hitters` dataset from the `ISLR2` package.

Step 1: Make Sure You Can Get the Data Into R

Since this dataset is already part of a package, we just have to load the package to have the data available to us. This is rarely the case; we almost always have to import the data from an external file.

Some "conventional" file types (.csv, .xls/.xlsx, and database files from SPSS/SAS/Stata) can be imported using the "Import Dataset" button in R Studio, but if you're not in R Studio, or you're in an R Markdown document, you'll need to write code to import it. It's usually easiest to just copy the code out of the "Code Preview" section of the R Studio dialog. Note that R Studio can sometimes crash trying to preview a very large dataset, so you may have to write code even if the file extension is one supported by the R Studio GUI. For large delimited text files (e.g., .csv or tab-delimited .txt files), `fread` from the `data.table` package usually works best.

For other data formats, you'll usually have to look online to figure out the best package to use to import the file. Worst-case scenario, any file that can be opened in a text editor can probably be read using `scan`, `read.table`, or `readLines`.

```
library(ISLR2) # already contains Hitters dataset
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.1.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3
```

I like to do a few checks to make sure everything imported okay. `View()` is the recommended function for viewing the full dataset in a separate window in R Studio, but you may also want to use some of:

```
head(Hitters) # first 6 rows
```

```
##                   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Andy Allanson      293   66     1   30  29    14     1    293    66      1
```

```
## -Alan Ashby          315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis         479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson        496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga    321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin     594  169     4   74  51    35    11   4408  1133     19
##                    CRuns CRBI CWalks League Division PutOuts Assists Errors
## -Andy Allanson        30   29     14      A        E     446      33     20
## -Alan Ashby          321  414    375      N        W     632      43     10
## -Alvin Davis         224  266    263      A        W     880      82     14
## -Andre Dawson        828  838    354      N        E     200      11      3
## -Andres Galarraga     48   46     33      N        E     805      40      4
## -Alfredo Griffin     501  336    194      A        W     282     421     25
##                    Salary NewLeague
## -Andy Allanson         NA         A
## -Alan Ashby         475.0         N
## -Alvin Davis        480.0         A
## -Andre Dawson       500.0         N
## -Andres Galarraga    91.5         N
## -Alfredo Griffin    750.0         A
```

```
dim(Hitters) # number of rows and columns
```

```
## [1] 322  20
```

```
colnames(Hitters) # variable names
```

```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"
##  [7] "Years"     "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"
## [13] "CWalks"    "League"    "Division"  "PutOuts"   "Assists"   "Errors"
## [19] "Salary"    "NewLeague"
```

Step 2: Figure Out What Is In Your Data

By looking at the dataset and investigating the data dictionary (`?Hitters`), you should be able to answer the following questions:

- What does one row in this dataset represent? How many rows are there?
- What does each variable in this dataset represent? Which variables are categorical and which are numerical?
- Which variable would be best to use as the response variable?

For a different dataset, you may also want to investigate:

- Are there any variables or sets of variables in the dataset that represent the same thing?
- Are there any variables whose type in R does not match their type in the study/data dictionary?

Step 3: Split Your Data

Before doing any actual analysis in R, you should split your data into a training and holdout set. The holdout set is for model validation and/or testing; you should never even peek at the holdout set when doing exploratory analysis.

Be careful when doing the splitting to ensure that information does not "leak" into the holdout set. For example, with time-stamped data, everything in the training set should have been observed earlier than anything in the holdout set. For another example, if two observational units are clearly related (e.g., two

divisions of the same company), the rows corresponding to both observational units should go in one or the other set.

Since there is no obvious dependence in this data, we can use just a random 80%/20% split.

```
set.seed(195)
n <- nrow(Hitters)
train_indices <- sample(n, size = floor(0.8*n))
Hitters_train <- Hitters[train_indices,]
Hitters_holdout <- Hitters[-train_indices,]
```

Step 4: Ask Questions

Questions generally come in three categories:

- Distribution: What is the distribution of a variable?
- Relationship: How does that distribution change in relationship to another variable?
- Identification: Which observational units have a particular property?

Step 4a: Distribution Questions

I typically find it best to start by investigating the distribution of the response variable.

In this dataset, the response variable in this dataset is numerical, which means that we should be able to describe its distribution in terms of center, variability, shape, and number/location of outliers.

The chunk below to uses the `summarize` function in the `dplyr` package to obtain the number of total observations, number of missing observations, five-number summary, mean, and standard deviation for the response variable in the training set `Hitters_train`. The argument `na.rm = TRUE` ensures that I don't return `NA` if there is missing data.

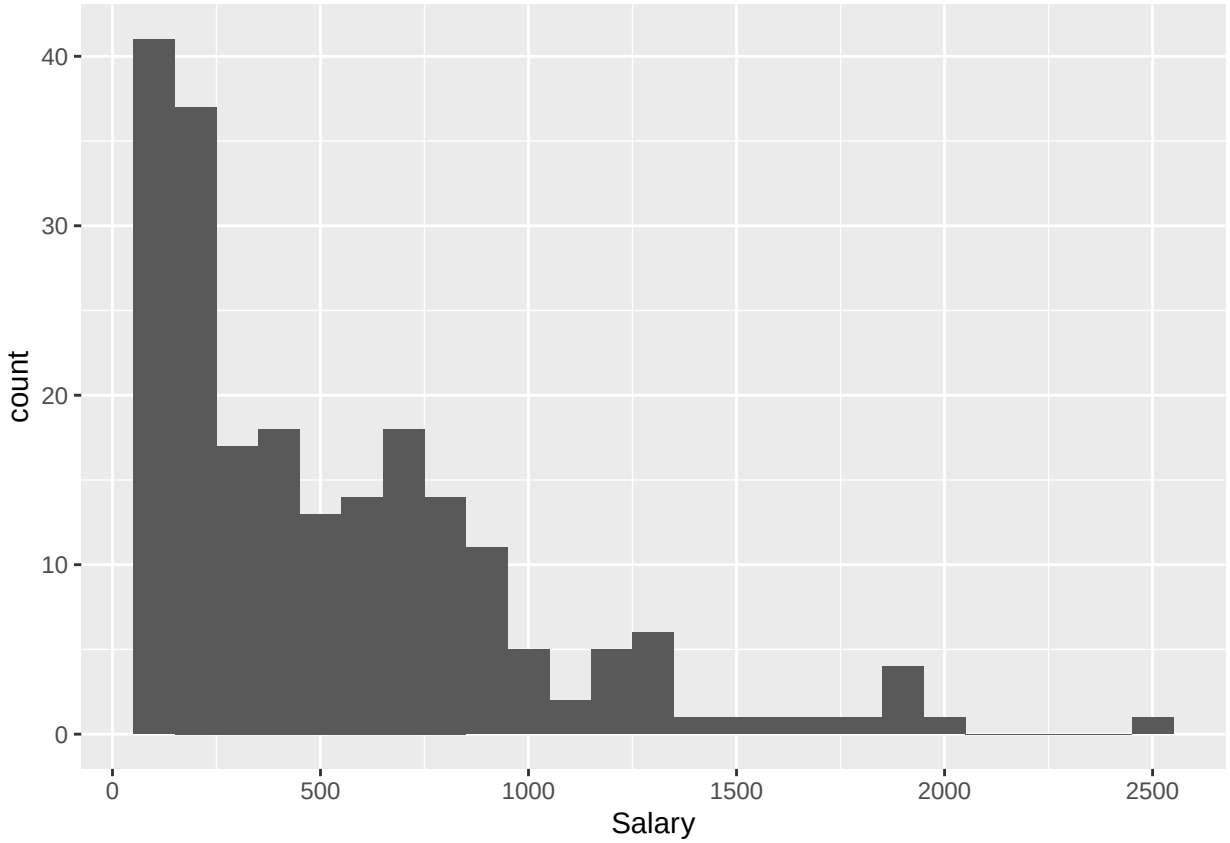
```
Hitters_train %>% summarize(
  num_total = n(),
  num_missing = sum(is.na(Salary)),
  min = min(Salary, na.rm = TRUE),
  Q1 = quantile(Salary, 0.25, na.rm = TRUE),
  median = median(Salary, na.rm = TRUE),
  Q3 = quantile(Salary, 0.75, na.rm = TRUE),
  max = max(Salary, na.rm = TRUE),
  mean = mean(Salary, na.rm = TRUE),
  sd = sd(Salary, na.rm = TRUE)
  )
```

```
##   num_total num_missing min     Q1 median       Q3  max     mean       sd
## 1       257          45  68 183.75  422.5 767.0833 2460 542.3609 451.9114
```

Shape and outliers, especially, are more easily identified graphically using a histogram:

```
# Note: I don't initially include the center and binwidth arguments
# I make the initial histogram using the (bad) defaults and then add them in based on what seems sensib
ggplot(data = Hitters_train,
       mapping = aes(
  x = Salary
)) + geom_histogram(center = 500, binwidth = 100)
```

```
## Warning: Removed 45 rows containing non-finite values (stat_bin).
```

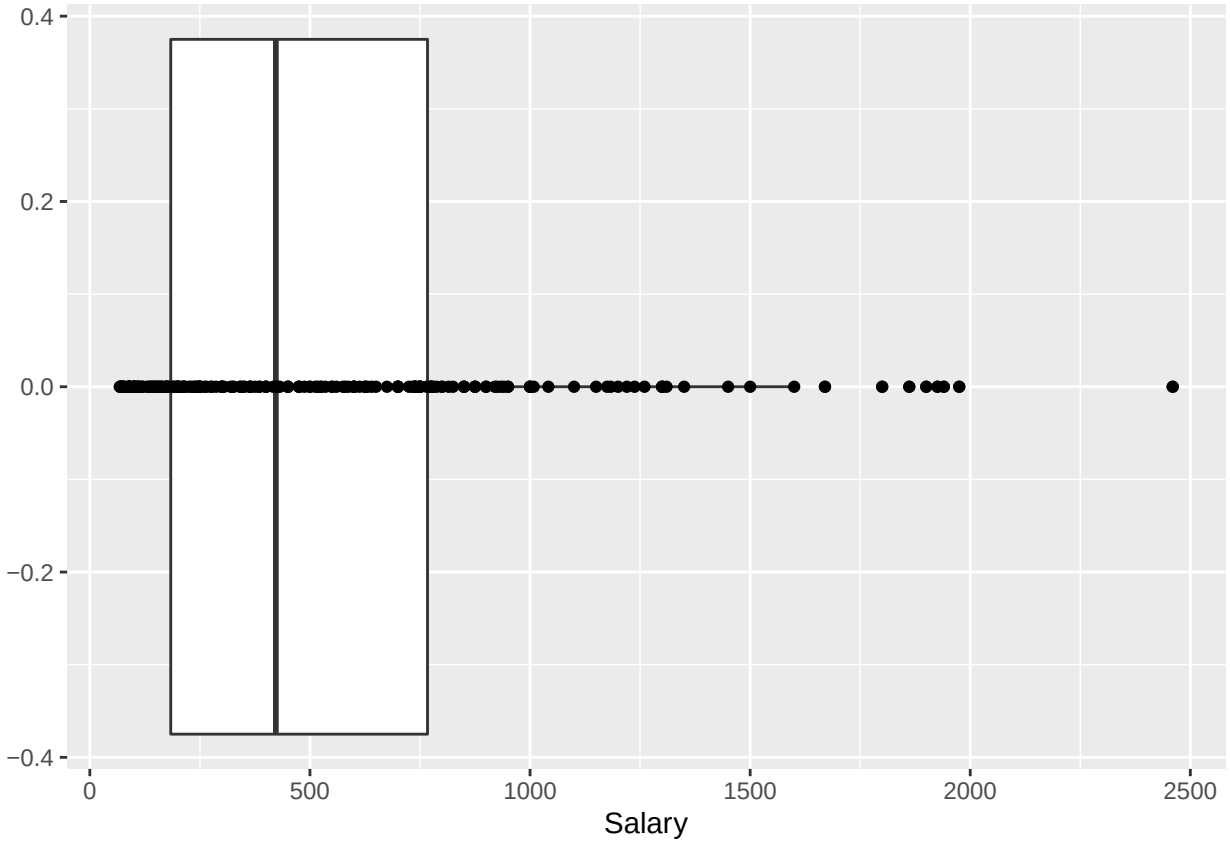


and/or a boxplot. For small datasets, I like to overlay the points on top of the box:

```
ggplot(data = Hitters_train,
       mapping = aes(
  y = Salary
)) + geom_boxplot() +
  geom_point(x = 0) + coord_flip()
```

```
## Warning: Removed 45 rows containing non-finite values (stat_boxplot).
```

```
## Warning: Removed 45 rows containing missing values (geom_point).
```




```
  # + coord_flip() for horizontal boxplot
```

Notice that R gives me a warning message that the missing data was removed from the dataset before plotting.

When we create our graphical and numerical summaries, we don't care about making anything look good - our graphs and summaries just need to be organized enough that *we* know how to interpret them. Generally I don't bother adding things like labels or color unless I have to show someone else the graph. I do try to create and name new variables within the `summarize()` function, but that's mostly because I find it easier to read the shorthand column names than something like `quantile(Salary, 0.75, na.rm = TRUE)`.

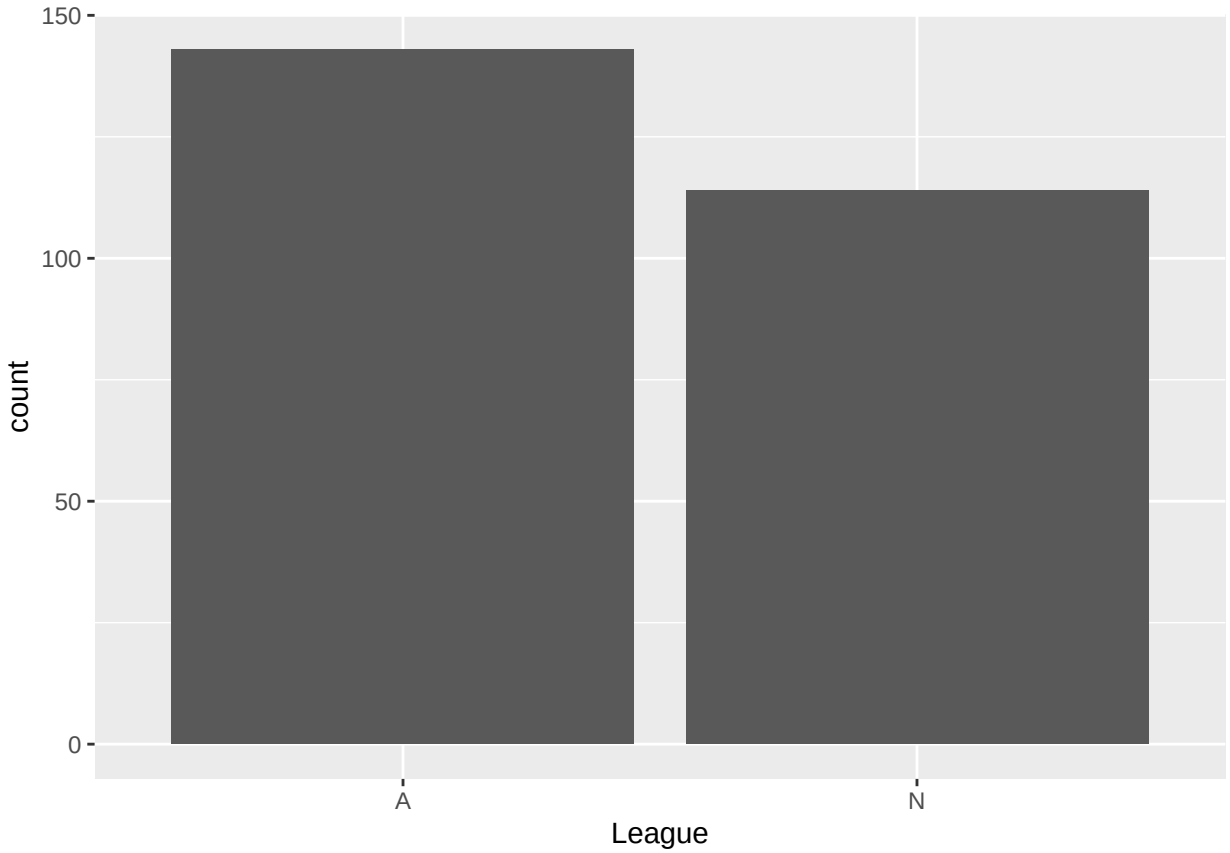
For categorical variables, it doesn't make sense to get numerical summaries like mean and standard deviation. The chunk below gets the count of observations in each category. Note that an `NA` category will get its own row in the table.

```
Hitters_train %>% group_by(League) %>% summarize(num = n())
```

```
## # A tibble: 2 x 2
##   League   num
##   <fct>  <int>
## 1 A        143
## 2 N        114
```

Sometimes it is easier to show this in a bar graph than a table:

```
ggplot(data = Hitters_train,
       mapping = aes(
         x = League
       )) + geom_bar()
```



Step 4b: Describe Relationships

Let's look at some ways to describe the relationship between a categorical and numerical variable.

First, we can combine the `group_by()` and `summarize()` functions to get a grouped summary of the response variable within each group:

```
Hitters_train %>% group_by(League) %>% summarize(
  num_total = n(),
  num_missing = sum(is.na(Salary)),
  min = min(Salary, na.rm = TRUE),
  Q1 = quantile(Salary, 0.25, na.rm = TRUE),
  median = median(Salary, na.rm = TRUE),
  Q3 = quantile(Salary, 0.75, na.rm = TRUE),
  max = max(Salary, na.rm = TRUE),
  mean = mean(Salary, na.rm = TRUE),
  sd = sd(Salary, na.rm = TRUE)
)
```

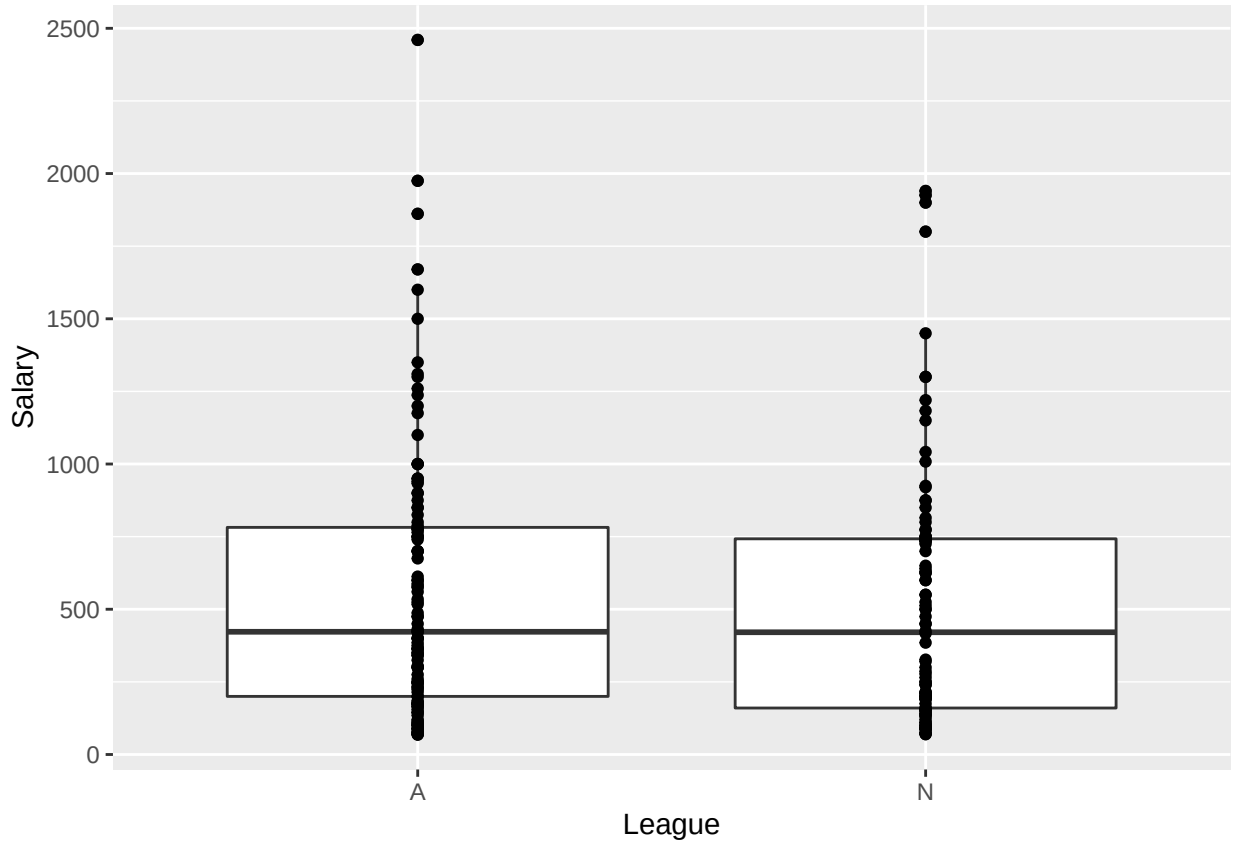
```
## # A tibble: 2 x 10
##   League num_total num_missing   min    Q1 median    Q3   max  mean    sd
##   <fct>      <int>       <int> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 A            143          27    68   200   422.  782.  2460  560.  457.
## 2 N            114          18    70   160   421.  742.  1940  521.  447.
```

It is easiest to visualize the distribution within each group using `ggplot2`. I like to make box plots and overlay a dot plot. We always start by mapping the x-axis to the categorical variable and the y-axis to the numerical variable, then tell R to flip the axes if the categorical variable is the response.

```
ggplot(data = Hitters_train,
       mapping = aes(x = League,
                     y = Salary)) +
  geom_boxplot() +
  geom_point() #+ coord_flip() if response is categorical
```

```
## Warning: Removed 45 rows containing non-finite values (stat_boxplot).
```

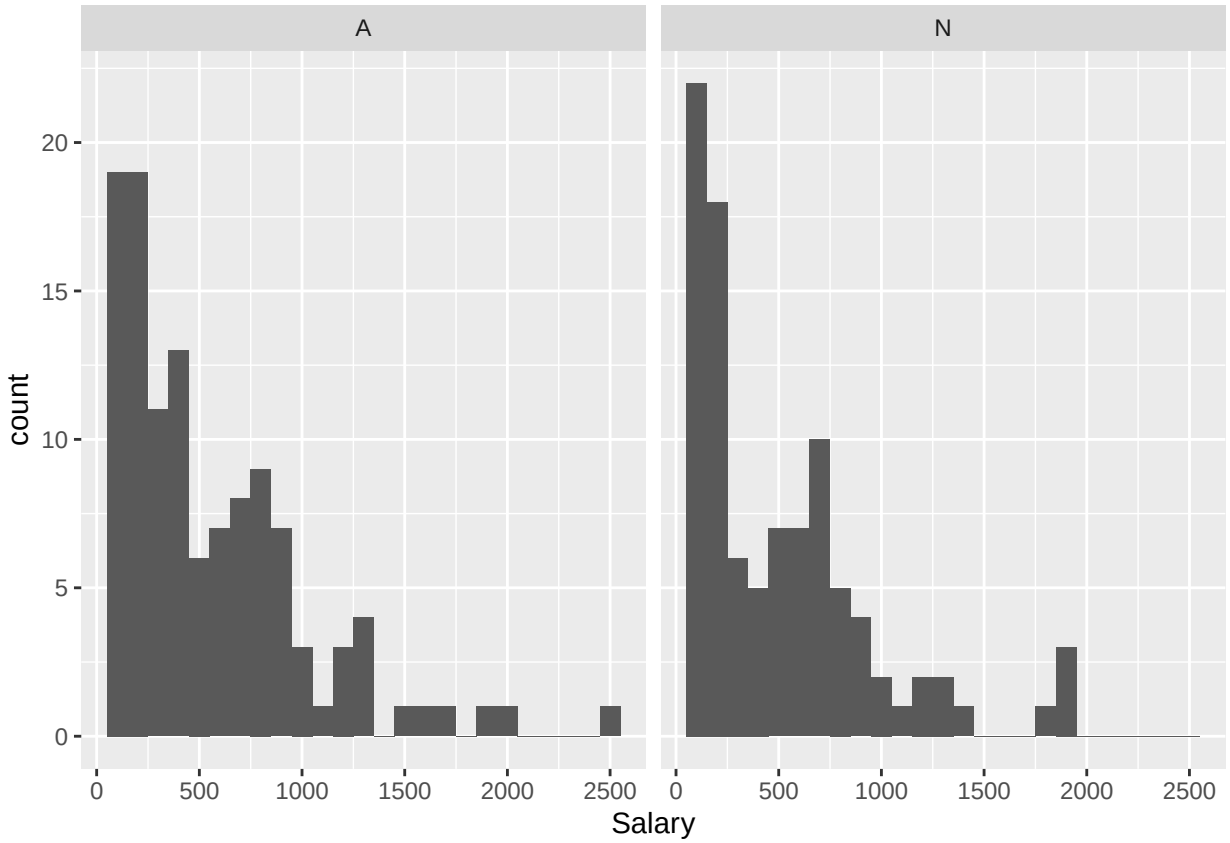
```
## Warning: Removed 45 rows containing missing values (geom_point).
```



If you insist on making histograms (for example, to check distribution shape), you want to use `facet_wrap`:

```
ggplot(data = Hitters_train,
       mapping = aes(x = Salary)) +
  geom_histogram(center = 500, binwidth = 100) +
  facet_wrap(~League)
```

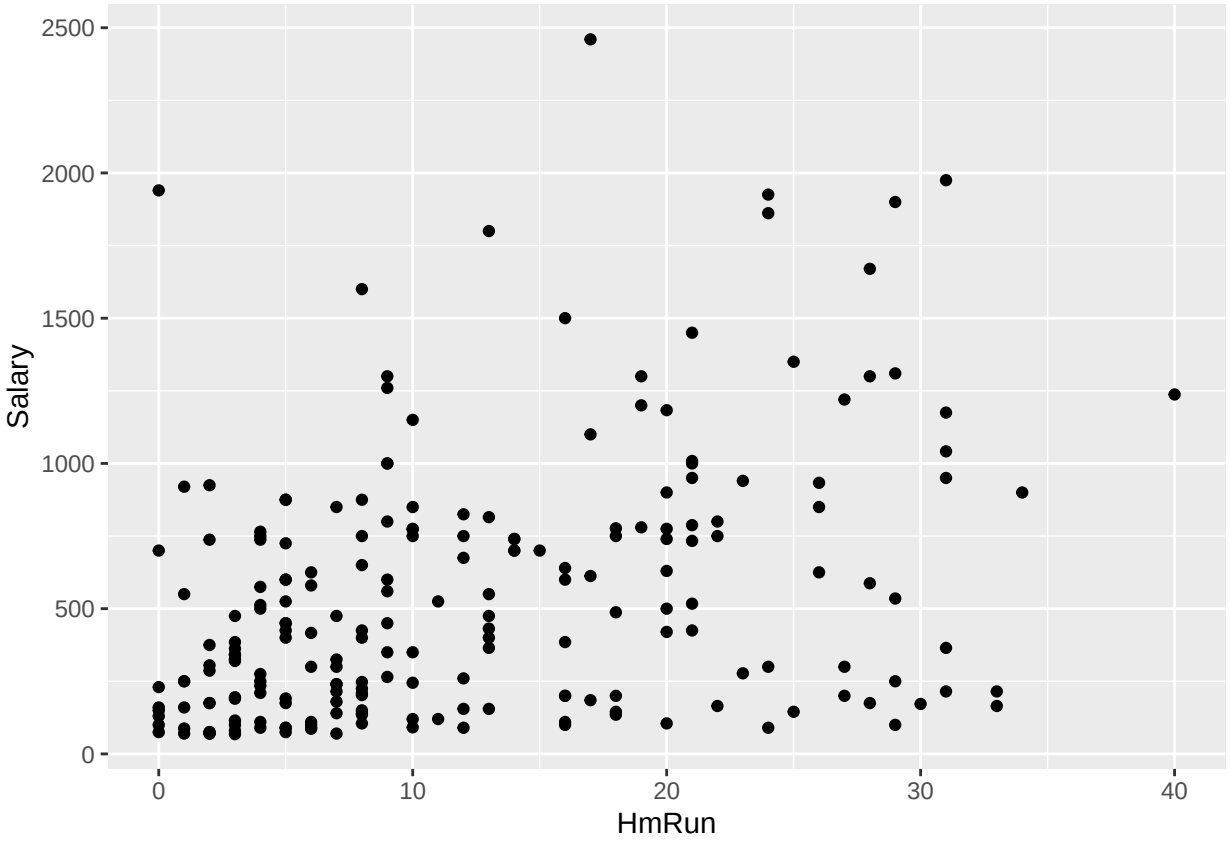
```
## Warning: Removed 45 rows containing non-finite values (stat_bin).
```



For relationships between numerical variables, we have the familiar scatterplot. In the chunk below, I create a scatterplot to show the relationship between the explanatory variable `HmRun` and the response variable.

```
ggplot(data = Hitters_train,
       mapping = aes(x = HmRun,
                     y = Salary)) +
  geom_point()
```

```
## Warning: Removed 45 rows containing missing values (geom_point).
```



We can also get the correlation between the two variables, but note that we should always start with the scatterplot to verify that the relationship is approximately linear. If the relationship is nonlinear or there are influential outliers, the correlation we get may be misleading.

```
Hitters_train %>% summarize(r = cor(HmRun, Salary,
                                    use = "pairwise.complete.obs"))
```

```
##           r
## 1 0.3706672
```

The `use` argument tells us to use all observations for which both variables have a value.

For categorical-categorical relationships, we should be used to a two-way table:

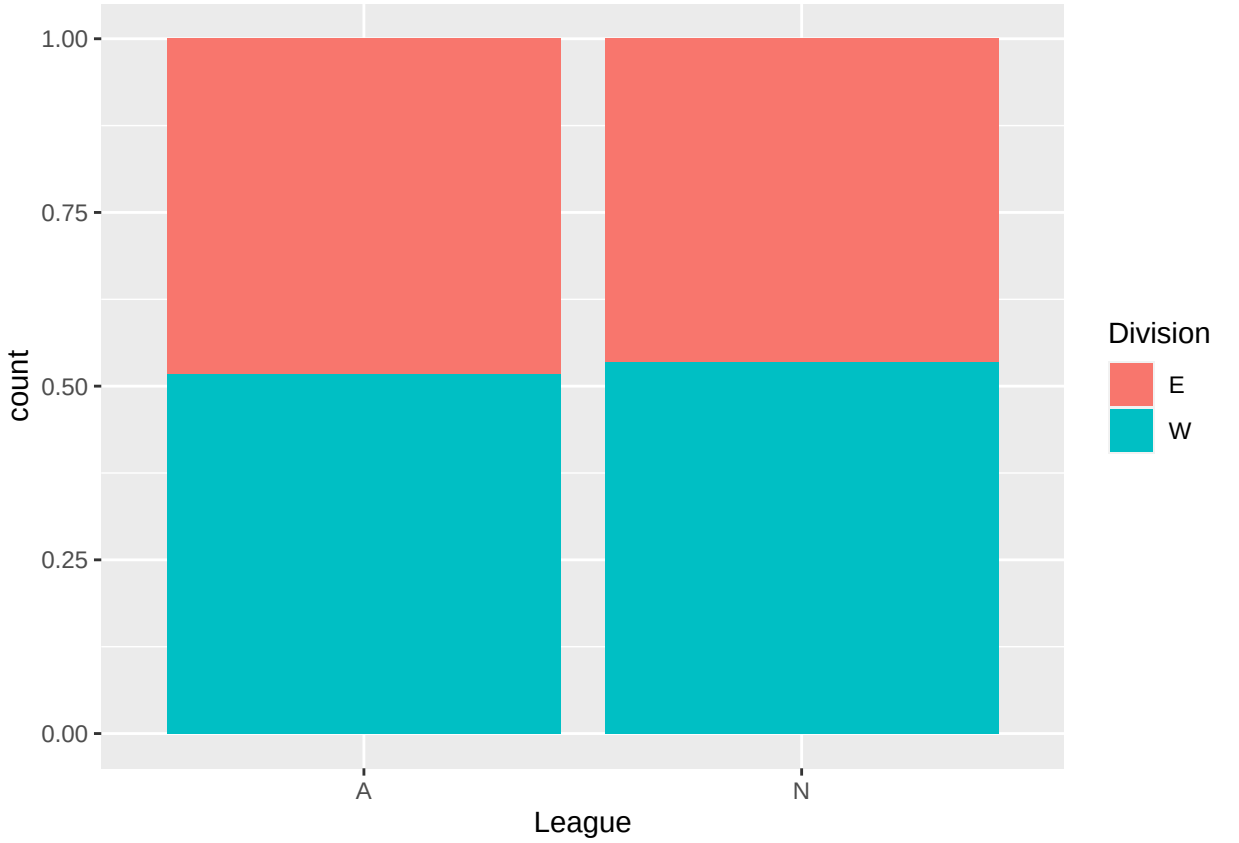
```
xtabs(~ League + Division, data = Hitters_train) %>%
  addmargins() # adds marginal totals
```

```
##       Division
## League   E   W Sum
##    A    69  74 143
##    N    53  61 114
##    Sum 122 135 257
```

To display this table graphically, I like to use a standardized stacked bar plot, which is basically a set of pie charts that are "unrolled" to be placed on the same set of axes:

```
ggplot(Hitters_train, mapping = aes(
  x = League,
  fill = Division,
```

```
)) +
  geom_bar(position = "fill")
```



This type of graph is great for seeing relationships as long as the `fill` variable is ordinal or doesn't have too many categories. Otherwise it gets a bit busy.

A variant on this type of plot, the *mosaic plot*, sets the width of each bar proportion to the number of observations in each category (i.e., the marginal totals for the variable on the x-axis). Thus, the area of each rectangle is proportional to the number of observations in each cell of the table. If you are interested in mosaic plots, check out the `ggmosaic` package.

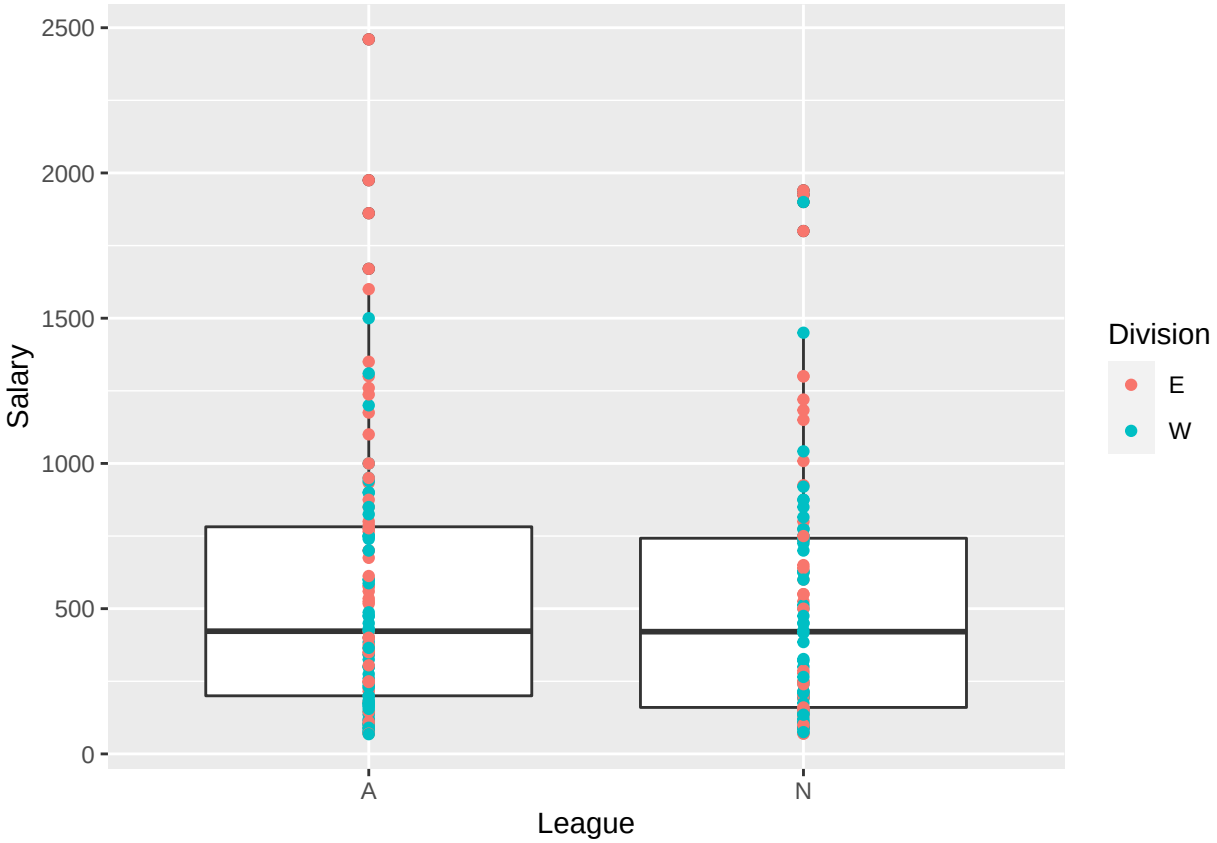
Step 4c: Describe More Complex Relationships

Plotting three or more variables on the same graph requires a bit of forethought, because each additional variable needs to be mapped to a different property of the plot (other than location along the x or y axes). The most obvious one to use is color:

```
ggplot(data = Hitters_train,
       mapping = aes(x = League,
                     y = Salary)) +
  geom_boxplot() +
  geom_point(aes(color = Division)) #+ coord_flip() if response is categorical
```

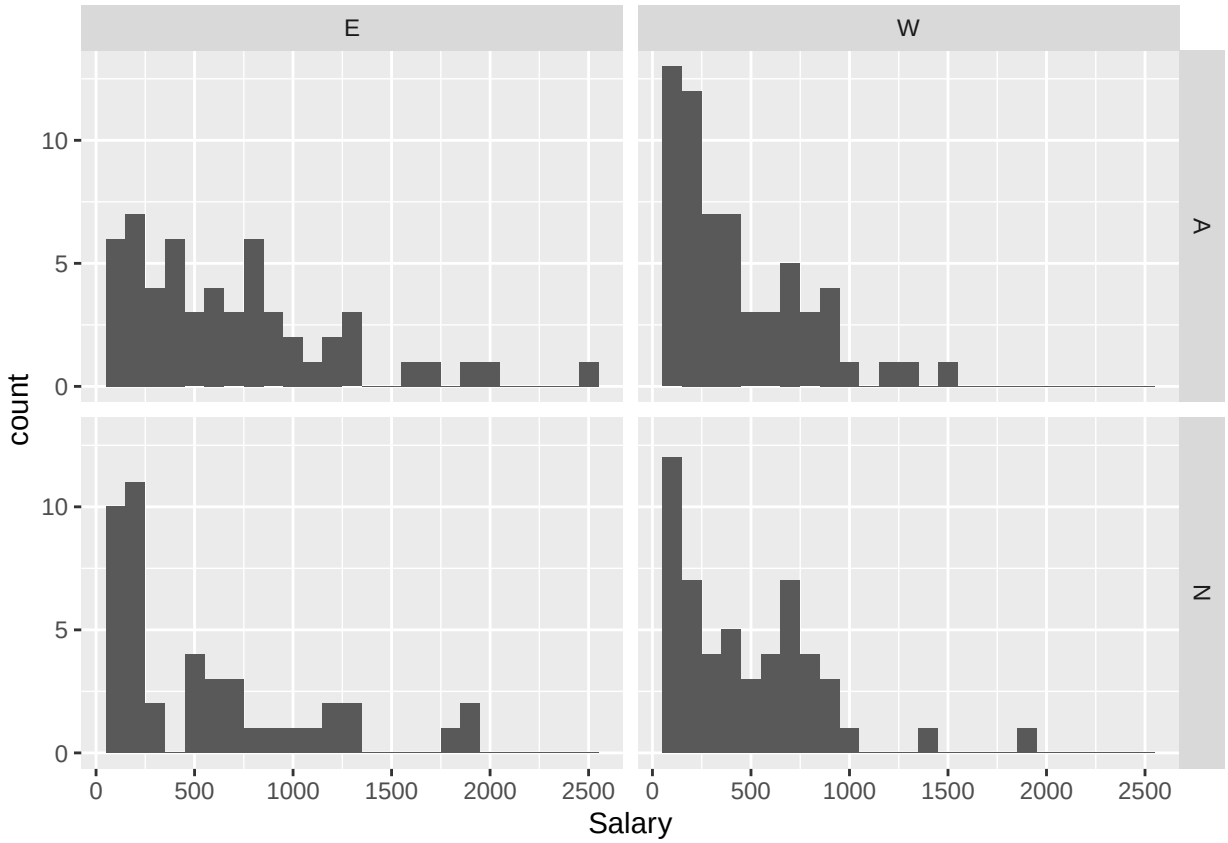
```
## Warning: Removed 45 rows containing non-finite values (stat_boxplot).
```

```
## Warning: Removed 45 rows containing missing values (geom_point).
```



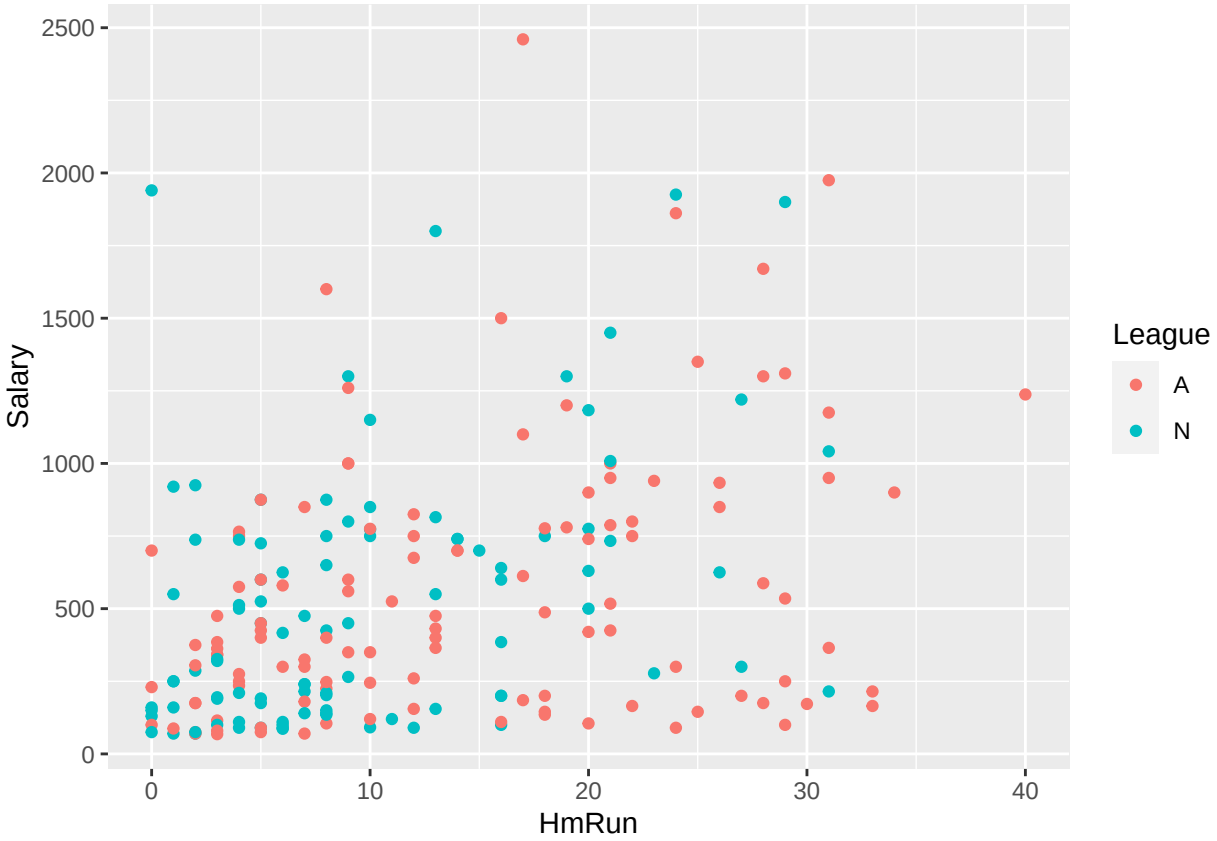

```
ggplot(data = Hitters_train,
       mapping = aes(x = Salary)) +
  geom_histogram(center = 500, binwidth = 100) +
  facet_grid(League~Division) # League groups on rows, Division groups on columns
```

```
## Warning: Removed 45 rows containing non-finite values (stat_bin).
```




```
ggplot(data = Hitters_train,
       mapping = aes(x = HmRun,
                     y = Salary,
                     color = League)) +
  geom_point()
```

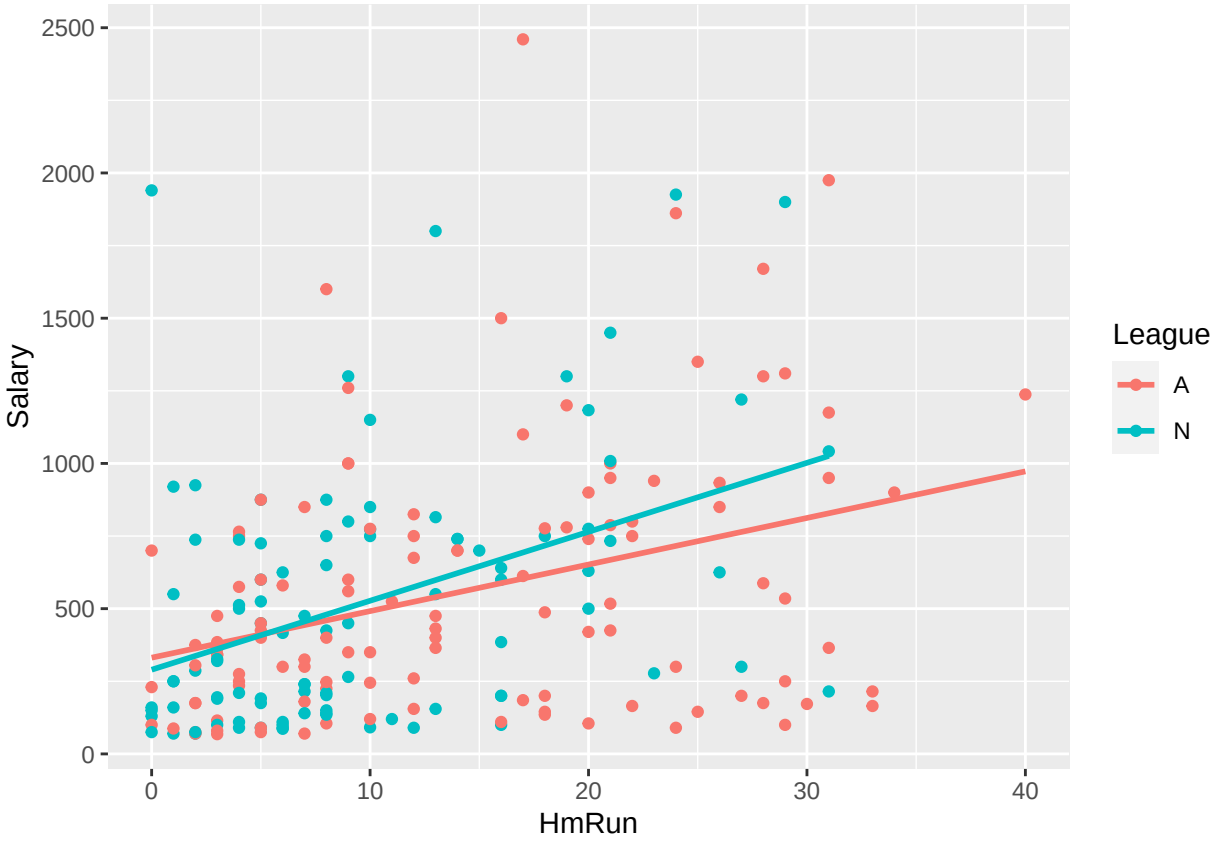
```
## Warning: Removed 45 rows containing missing values (geom_point).
```



One fun trick with ggplot2 is to automatically add the least-squares regression line:

```
ggplot(data = Hitters_train,
       mapping = aes(x = HmRun,
                     y = Salary,
                     color = League)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'
## Warning: Removed 45 rows containing non-finite values (stat_smooth).
## Warning: Removed 45 rows containing missing values (geom_point).
```



By default `ggplot2` will include a 95% confidence interval for the mean of the response along with the smoothing; `se = FALSE` says to just indicate the trend.

There isn't much we can do in the way of tabular three-variable summaries, but we can put extra stuff in the `group_by` argument:

```
Hitters_train %>% group_by(League, Division) %>% summarize(
  num_total = n(),
  num_missing = sum(is.na(Salary)),
  min = min(Salary, na.rm = TRUE),
  Q1 = quantile(Salary, 0.25, na.rm = TRUE),
  median = median(Salary, na.rm = TRUE),
  Q3 = quantile(Salary, 0.75, na.rm = TRUE),
  max = max(Salary, na.rm = TRUE),
  mean = mean(Salary, na.rm = TRUE),
  sd = sd(Salary, na.rm = TRUE)
)
```

```
## `summarise()` has grouped output by 'League'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 4 x 11
## # Groups:   League [2]
##   League Division num_total num_missing   min    Q1 median    Q3   max  mean
##   <fct>  <fct>        <int>       <int> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1 A      E               69          14    70  323.   575   942.  2460  696.
## 2 A      W               74          13    68  165    325   700   1500  437.
## 3 N      E               53           9    70  159.   282.  762.  1940  559.
```

```
## 4 N      W               61           9    75  190    438.  738.  1900  489.
## # ... with 1 more variable: sd <dbl>
```

Step 5: Summarize and Follow Up On Your Findings

If you notice anything interesting or unusual when looking at your summaries, make a note of it and investigate further if possible. This is also the time to ask and answer identification-type questions, for example, to identify which observations corresponded to the outliers in your graphs.

Typically, identification questions get answered using `filter`. For example, we might notice that there is a player who has a very high salary despite not hitting any home runs. We could figure out which player that is:

```
Hitters_train %>% select(HmRun, Salary) %>% filter(HmRun == 0, Salary > 1500)
```

```
##               HmRun Salary
## -Ozzie Smith      0   1940
```

We could also figure out which players were within $\$5,000$ of the median salary in each league, using a grouped filter:

```
Hitters_train %>% tibble::rownames_to_column(var = "Name") %>%
  group_by(League) %>%
  filter(abs(Salary - median(Salary, na.rm = TRUE)) <= 5) %>%
  select(Name, League, HmRun, Salary)
```

```
## # A tibble: 5 x 4
## # Groups:   League [2]
##   Name            League HmRun Salary
##   <chr>           <fct>  <int>  <dbl>
## 1 -Ron Kittle     A         21   425
## 2 -Lou Whitaker   A         20   420
## 3 -Terry Harper   N          8   425
## 4 -Craig Reynolds N          6   417.
## 5 -Tony Phillips  A          5   425
```

If any additional data wrangling is suggested (for example, combining groups or transforming a numerical variable), modify the training data to do the appropriate transformation and then simply re-run your analysis with the overwritten dataset.

Based on the set of summaries and graphs produced earlier in this example, you should already know and be able to describe:

- The center, variability, and shape of the distribution of `Salary`
- How the center, variability, and shape of the distribution of `Salary` change depending on the `League` and `Division`
- Whether there are any outliers or otherwise "weird" values (e.g., values that suggest a clear data collection/entry error, such as a negative salary) for `Salary`
- The groups in which those unusual values occurred
- The strength, direction, and form of the relationship between `HmRun` and `Salary`
- How that relationship changes depending on the `League`

If you have a pretty good handle on everything up to this point, pick a new explanatory/predictor variable and go through the same process of asking distribution, relationship, and identification questions.

Now It's Your Turn

Ask and answer a few of your own questions about the `Hitters_train` dataset. Note that actually writing the code to find your answer may be just outside your current capabilities; if you are having trouble getting an answer, first write pseudocode for how you would achieve it, then call me over to help you figure out the correct R syntax to do it.

If you're having trouble coming up with your own, here are a few that I came up with as I prepared this worked example:

- We expect players to hit a "prime" in the early-to-middle part of their career and then drop off later. Is this actually reflected in the 1986 stats? What about the 1987 salaries?
- Who had the highest salary in each league? What were their stats in 1986?
- How many players switched leagues prior to the 1987 season, and did they tend to make more money than players who stayed in the same league?

All three of these questions lead to really interesting follow-up questions, so you're welcome to start there.