CSSS508, Week 11

Working with Model Results

Chuck Lanfear

Jun 5, 2019

Updated: Jun 5, 2019

UW CS&SS

Topics for Today

Displaying Model Results

- `broom`
  - Turning model output lists into dataframes
  - Summarizing models
- `ggeffects`
  - Creating counterfactual estimates
  - Plotting marginal effects
- Manual counterfactual plots
- Making regression tables
  - Using `pander` for models
  - Using `sjTable()` in `sjPlot`
- Wrapping up the course

`broom`

`broom`

`broom` is a package that "tidies up" the output from models such a `lm()` and `glm()`.

It has a small number of key functions:

- `tidy()` - Creates a dataframe summary of a model.
- `augment()` - Adds columns—such as fitted values—to the data used in the model.
- `glance()` - Provides one row of fit statistics for models.

```r
library(broom)
```

Model Output is a List

`lm()` and `summary()` produce lists as output, which cannot go directly into tidyverse functions, particularly those in `ggplot2`.

```
lm_1 <- lm(yn ~ num1 + fac1, data = ex_dat)
summary(lm_1)
```

```
##
## Call:
## lm(formula = yn ~ num1 + fac1, data = ex_dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.9817 -1.9858 -0.1489  1.9035  6.9642
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.3863     0.3775   1.023 0.307482
## num1          0.4428     0.1005   4.407 1.72e-05 ***
## fac1B         1.7557     0.5027   3.492 0.000592 ***
## fac1C         3.0512     0.4972   6.137 4.57e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.808 on 196 degrees of freedom
## Multiple R-squared:  0.2366,    Adjusted R-squared:  0.2249
## F-statistic: 20.25 on 3 and 196 DF,  p-value: 1.795e-11
....
```

Model Output Varies!

Each type of model also produces somewhat different output, so you can't just reuse the same code to handle output from every model.

```
glm_1 <- glm(yb ~ num1 + fac1, data = ex_dat, family=binomial(link="logit"))
summary(glm_1)
```

```
##
## Call:
## glm(formula = yb ~ num1 + fac1, family = binomial(link = "logit"),
##     data = ex_dat)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8205  -1.1139  -0.4827   0.9698   2.0242
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.44708    0.33578  -4.310 1.64e-05 ***
## num1         0.25434    0.08078   3.148 0.001641 **
## fac1B        1.47671    0.40772   3.622 0.000292 ***
## fac1C        1.70849    0.40872   4.180 2.91e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
....
```

`broom::tidy()`

`tidy()` produces the similar output, but as a dataframe.

```
lm_1 %>% tidy()
```

```
## # A tibble: 4 x 5
##   term        estimate std.error statistic       p.value
##   <chr>          <dbl>     <dbl>     <dbl>         <dbl>
## 1 (Intercept)    0.386     0.378      1.02 0.307
## 2 num1           0.443     0.100      4.41 0.0000172
## 3 fac1B          1.76      0.503      3.49 0.000592
## 4 fac1C          3.05      0.497      6.14 0.00000000457
```

Each type of model (e.g. `glm`, `lmer`) has a different *method* with its own additional arguments. See `?tidy.lm` for an example.

`broom::tidy()`

This output is also completely identical between different models.

This can be very useful and important if running models with different test statistics... or just running a lot of models!

```
glm_1 %>% tidy()
```

```
## # A tibble: 4 x 5
##   term        estimate std.error statistic   p.value
##   <chr>          <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)   -1.45     0.336      -4.31 0.0000164
## 2 num1           0.254    0.0808      3.15 0.00164
## 3 fac1B          1.48     0.408       3.62 0.000292
## 4 fac1C          1.71     0.409       4.18 0.0000291
```

`broom::glance()`

`glance()` produces dataframes of fit statistics for models.

If you run many models, you can compare each model row-by-row in each column… or even plot their different fit statistics to allow holistic comparison.

```
glance(lm_1)
```

```
## # A tibble: 1 x 11
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <int>  <dbl> <dbl> <dbl>
## 1     0.237         0.225  2.81      20.2 1.80e-11     4  -488.  986. 1003.
## # ... with 2 more variables: deviance <dbl>, df.residual <int>
```

`broom augment()`

`augment()` takes values generated by a model and adds them back to the original data. This includes fitted values, residuals, and leverage statistics.

```
augment(lm_1) %>% head()
```

```
## # A tibble: 6 x 10
##       yn    num1 fac1  .fitted .se.fit .resid   .hat .sigma .cooksd
##    <dbl>   <dbl> <fct>   <dbl>   <dbl>  <dbl>  <dbl>  <dbl>   <dbl>
## 1  1.96  -0.0643 A      0.358    0.379  1.60  0.0182   2.81 1.53e-3
## 2 -0.159  3.81   A      2.08     0.489 -2.23  0.0304   2.81 5.12e-3
## 3  3.35   1.34   A      0.980    0.378  2.37  0.0182   2.81 3.35e-3
## 4  0.758 -0.725  A      0.0653   0.397  0.693 0.0200   2.81 3.17e-4
## 5  5.62   0.0759 C      3.47     0.340  2.15  0.0147   2.81 2.22e-3
## 6  0.580 -2.16   C      2.48     0.454 -1.90  0.0261   2.81 3.15e-3
## # ... with 1 more variable: .std.resid <dbl>
```

The Power of `broom`

The real advantage of `broom` becomes apparent when running many models at once. Here we run separate models for each level of `fac1`:

```
ex_dat %>% group_by(fac1) %>% do(tidy(lm(yn ~  num1 + fac2 + num2, data = ex_dat)))
```

```
## # A tibble: 12 x 6
## # Groups:   fac1 [3]
##    fac1  term        estimate std.error statistic  p.value
##    <fct> <chr>          <dbl>     <dbl>     <dbl>    <dbl>
##  1 A     (Intercept)    1.69     0.242       6.95 5.15e-11
##  2 A     num1           0.479    0.0821      5.84 2.17e- 8
##  3 A     fac2No         1.05     0.328       3.19 1.66e- 3
##  4 A     num2           0.655    0.0536     12.2  7.18e-26
##  5 B     (Intercept)    1.69     0.242       6.95 5.15e-11
##  6 B     num1           0.479    0.0821      5.84 2.17e- 8
##  7 B     fac2No         1.05     0.328       3.19 1.66e- 3
##  8 B     num2           0.655    0.0536     12.2  7.18e-26
##  9 C     (Intercept)    1.69     0.242       6.95 5.15e-11
## 10 C     num1           0.479    0.0821      5.84 2.17e- 8
## 11 C     fac2No         1.05     0.328       3.19 1.66e- 3
## 12 C     num2           0.655    0.0536     12.2  7.18e-26
```

`do()` repeats whatever is inside it once for each level of the variable(s) in `group_by()` then puts them together as a data frame.

Plotting Model Results

`geom_smooth()`

I have used `geom_smooth()` in many past examples.

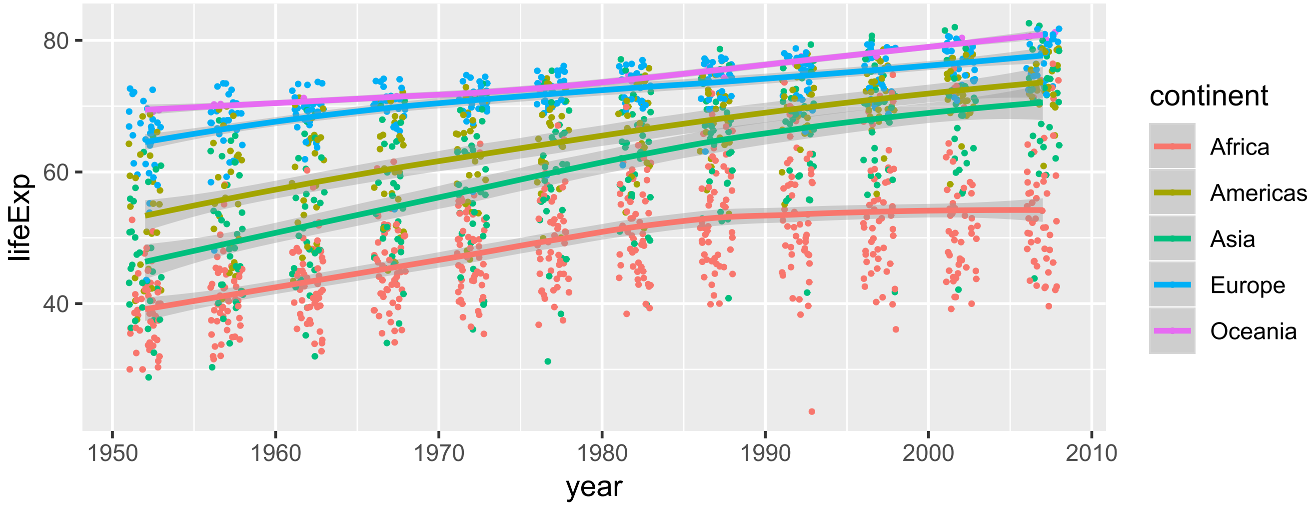
`geom_smooth()` generates "smoothed conditional means" including loess curves and generalized additive models (GAMs).

Note, however, that most regression models are conditional mean models, such as ordinary least squares, generalized linear models.

We can use `geom_smooth()` to add a layer depicting common bivariate models.

Default `geom_smooth()`

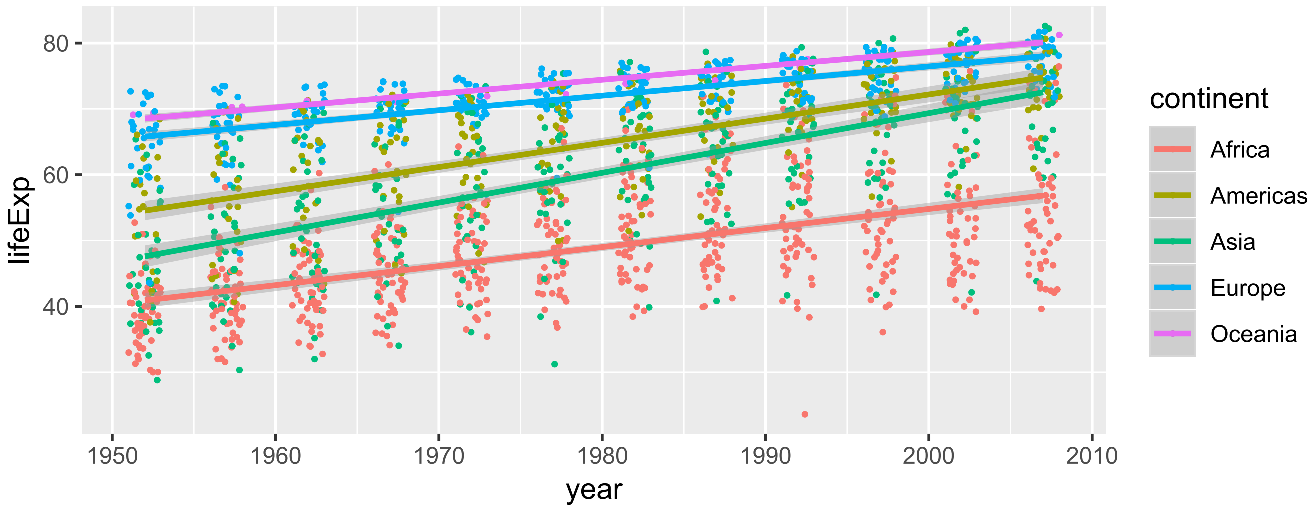
```
library(gapminder)
ggplot(data = gapminder,
       aes(x = year, y = lifeExp, color = continent)) +
  geom_point(position = position_jitter(1,0), size = 0.5) +
  geom_smooth()
```



By default, `geom_smooth()` chooses either a loess smoother (N < 1000) or a GAM depending on the number of observations.

Linear `glm`

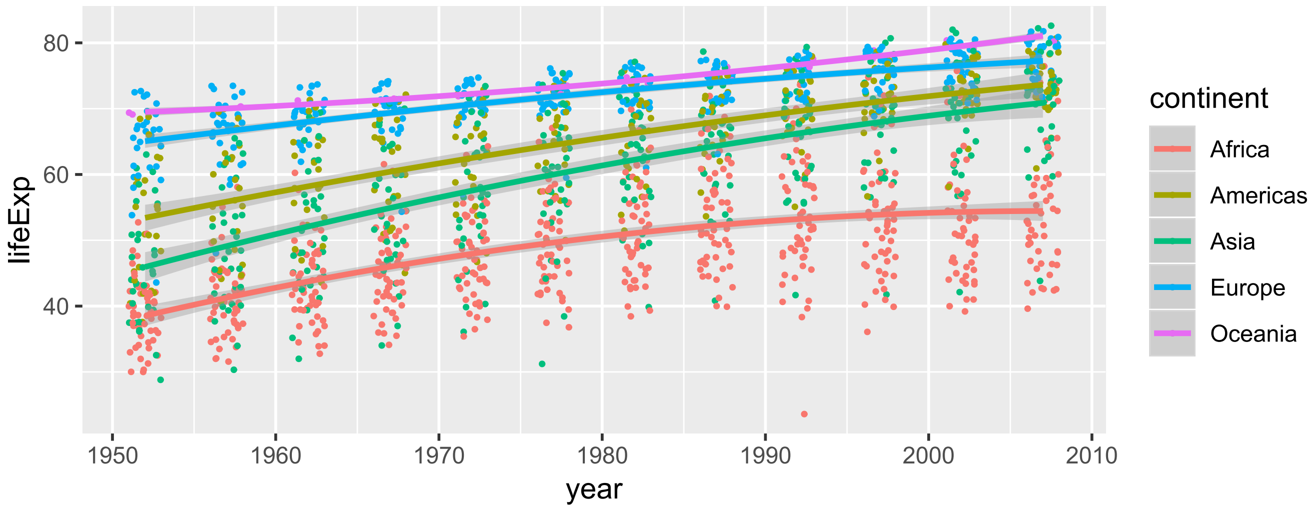
```
ggplot(data = gapminder,
       aes(x = year, y = lifeExp, color = continent)) +
  geom_point(position = position_jitter(1,0), size = 0.5) +
  geom_smooth(method = "glm", formula = y ~ x)
```



We could also fit a standard linear model using either `method = "glm"` or `method = "lm"` and a formula like `y ~ x`.

Polynomial `glm`

```
ggplot(data = gapminder,
       aes(x = year, y = lifeExp, color = continent)) +
  geom_point(position = position_jitter(1,0), size = 0.5) +
  geom_smooth(method = "glm", formula = y ~ poly(x, 2))
```



`poly(x, 2)` produces a quadratic model which contains a linear term (`x`) and a quadratic term (`x^2`).

More Complex Models

What if we want something more complex than a bivariate model?

What if we have a statistically complex model, like nonlinear probability model or multilevel model?

We need to go beyond `geom_smooth()`!

But first, vocab!

We are often interested in what might happen if some variables take particular values, often ones not seen in the actual data.

When we set variables to certain values, we refer to them as **counterfactual values** or just **counterfactuals**.

For example, if we know nothing about a new observation, our prediction for that estimate is often based on assuming every variable is at its mean.

Sometimes, however, we might have very specific questions which require setting (possibly many) combinations of variables to particular values and making an estimate or prediction.

Providing specific estimates, conditional on values of covariates, is a nice way to summarize results, particularly for models with unintuitive parameters (e.g. logit models).

ggeffects

`ggeffects`

If we want to look at more complex models, we can use `ggeffects` to create and plot tidy *marginal effects*.

That is, tidy dataframes of *ranges* of predicted values that can be fed straight into `ggplot2` for plotting model results.

We will focus on two `ggeffects` functions:

- `ggpredict()` - Computes predicted values for the outcome variable at margins of specific variables.
- `plot.ggeffects()` - A plot method for `ggeffects` objects (like `ggredict()` output)

```r
library(ggeffects)
```

Quick Simulated Data

To best show off `ggeffects`, I need a data frame with numeric and categorical variables with strong relationships. It is easiest to just simulate it:

```
ex_dat <- data.frame(num1 = rnorm(200, 1, 2),
                     fac1 = sample(c(1, 2, 3), 200, TRUE),
                     num2 = rnorm(200, 0, 3),
                     fac2 = sample(c(1, 2))) %>%
  mutate(yn = num1 * 0.5 + fac1 * 1.1 + num2 * 0.7 +
              fac2 - 1.5  + rnorm(200, 0, 2)) %>%
  mutate(yb = as.numeric(yn > mean(yn))) %>%
  mutate(fac1 = factor(fac1, labels = c("A", "B", "C")),
         fac2 = factor(fac2, labels = c("Yes", "No")))
```

Now we can get `ggpredict`ing!

`ggpredict()`

When you run `ggpredict()`, it produces a dataframe with a row for every unique value of a supplied predictor ("independent") variable (`term`).

Each row contains an expected (estimated) value for the outcome ("dependent") variable, plus confidence intervals.

```
lm_1 <- lm(yn ~ num1 + fac1, data = ex_dat)
lm_1_est <- ggpredict(lm_1, terms = "num1")
```

If desired, the argument `interval="prediction"` will give predicted intervals instead.

`ggpredict()` output

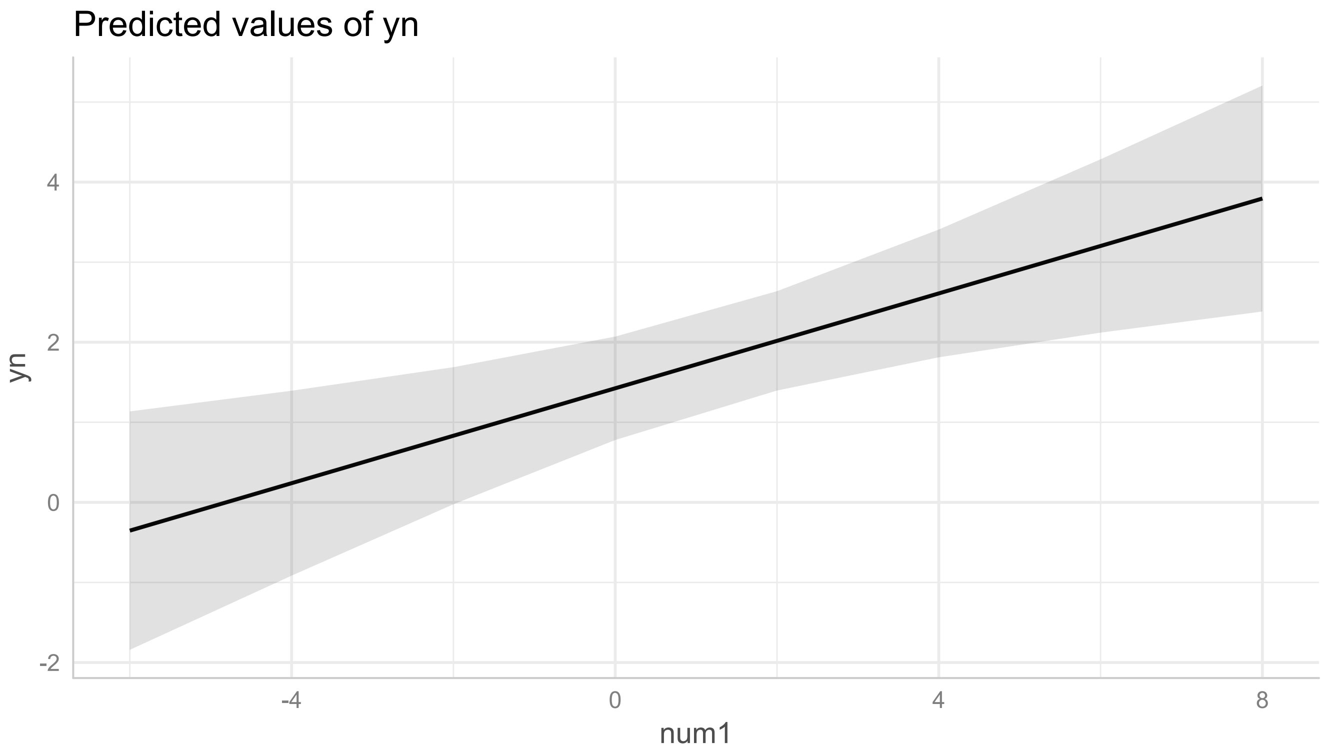
```
lm_1_est
```

```
##
## # Predicted values of yn
## # x = num1
##
##    x predicted std.error conf.low conf.high
##   -6    -0.353     0.760   -1.842     1.136
##   -4     0.240     0.589   -0.915     1.394
##   -2     0.832     0.437   -0.024     1.689
##    0     1.425     0.330    0.779     2.071
##    2     2.017     0.317    1.397     2.638
##    4     2.610     0.407    1.812     3.408
##    6     3.202     0.552    2.120     4.285
##    8     3.795     0.720    2.384     5.206
##
## Adjusted for:
## * fac1 = A
```

`plot()` for `ggpredict()`

`ggeffects` features a `plot()` *method*, `plot.ggeffects()`, which produces a ggplot when you give `plot()` output from `ggpredict()`.

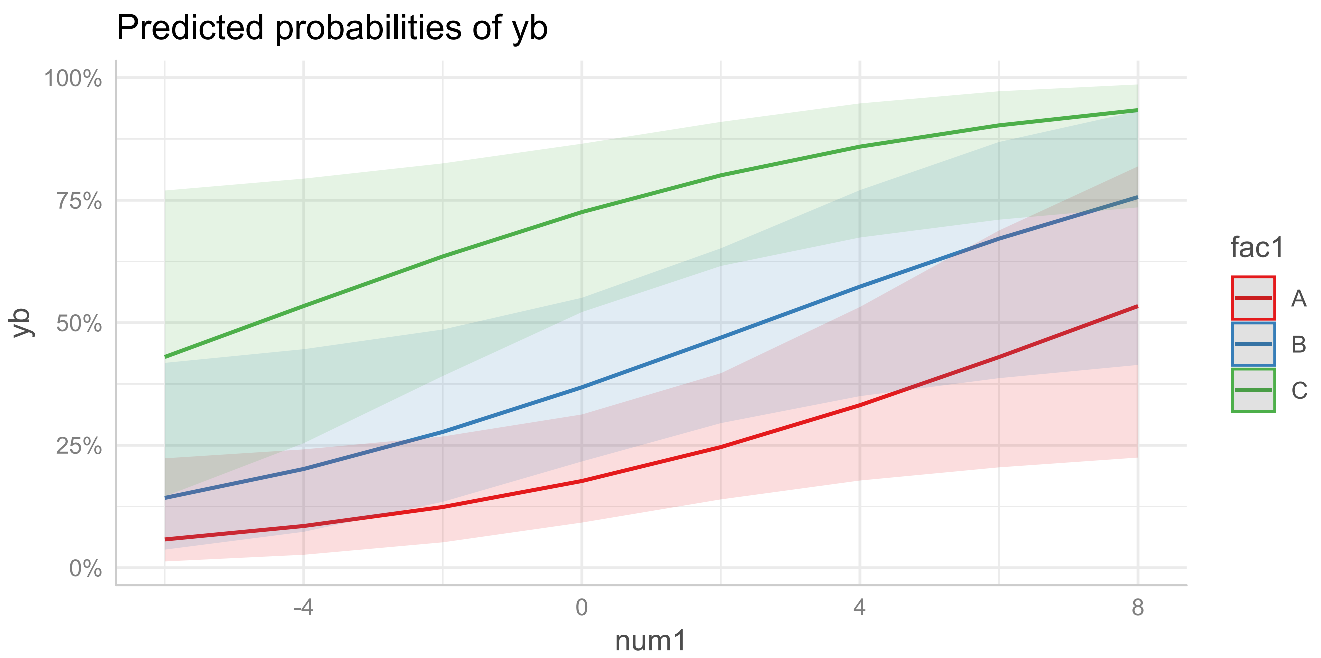
```
plot(lm_1_est)
```



Grouping with `ggpredict()`

When using a vector of `terms`, `ggeffects` will plot the first along the x-axis and use others for *grouping*. Note we can pipe a model into `ggpredict()`!

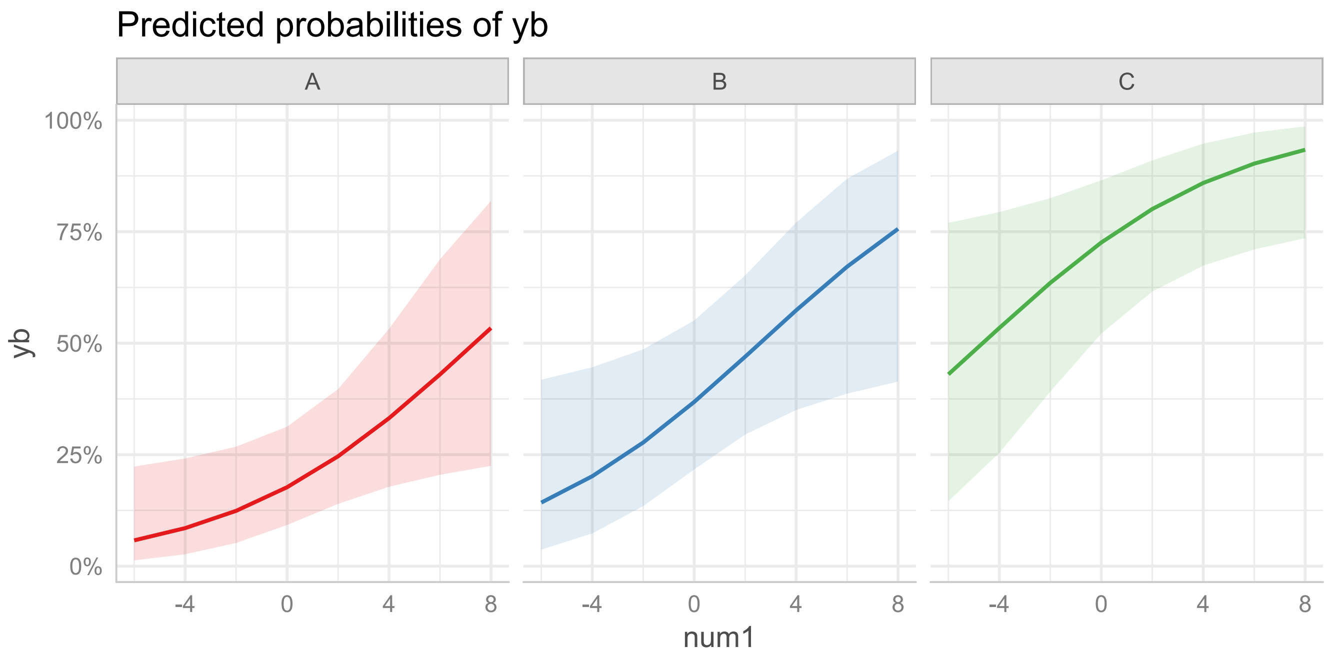
```
glm(yb ~ num1 + fac1 + num2 + fac2, data = ex_dat, family=binomial(link = "logit")) %>%
  ggpredict(terms = c("num1", "fac1")) %>% plot()
```



Faceting with `ggpredict()`

You can add `facet=TRUE` to the `plot()` call to facet over *grouping terms*.

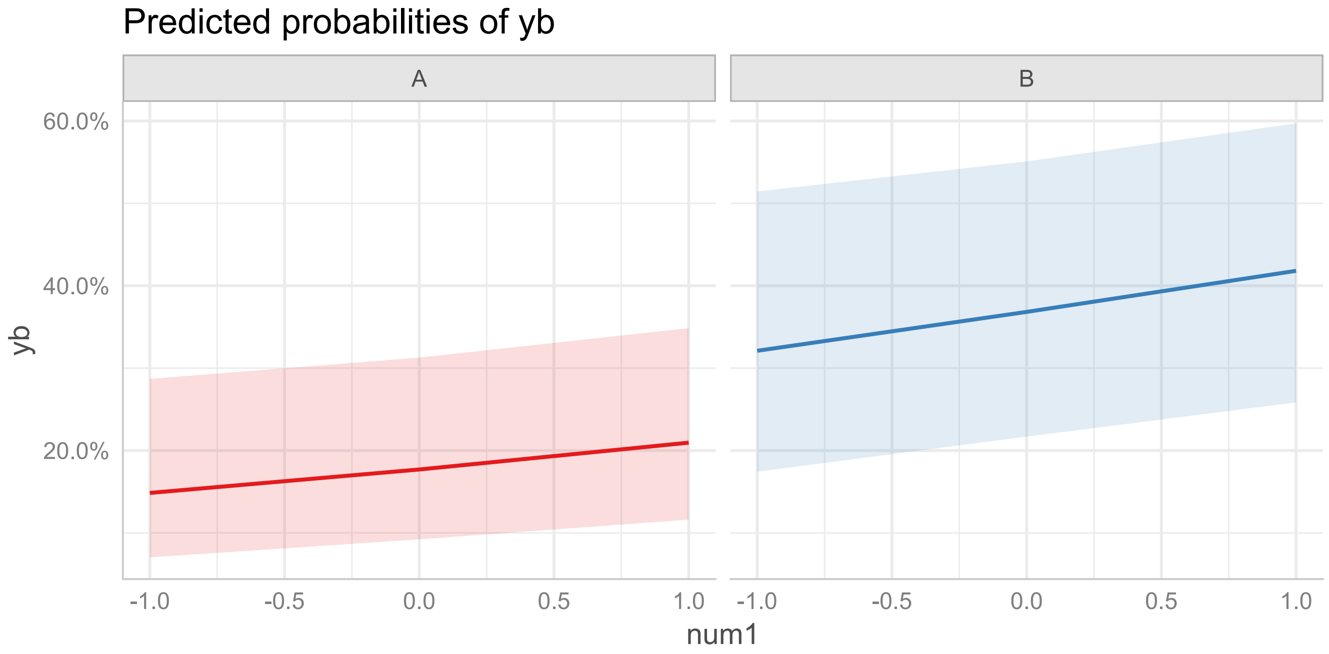
```
glm(yb ~ num1 + fac1 + num2 + fac2, data = ex_dat, family = binomial(link = "logit")) %>%
  ggpredict(terms = c("num1", "fac1")) %>% plot(facet=TRUE)
```



Counterfactual Values

You can add values in square brackets in the `terms=` argument to specify counterfactual values.

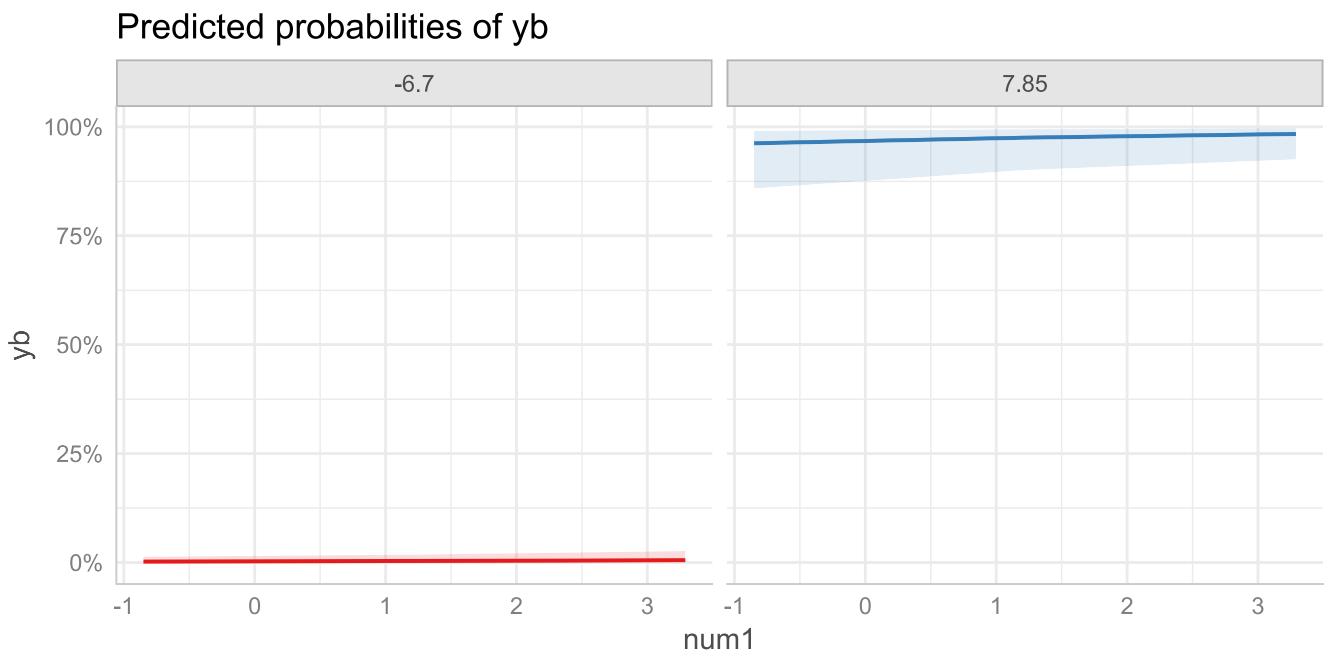
```
glm(yb ~ num1 + fac1 + num2 + fac2, data=ex_dat, family=binomial(link="logit")) %>%
  ggpredict(terms = c("num1 [-1,0,1]", "fac1 [A,B]")) %>% plot(facet=TRUE)
```



Representative Values

You can also use `[meansd]` or `[minmax]` to set representative values.

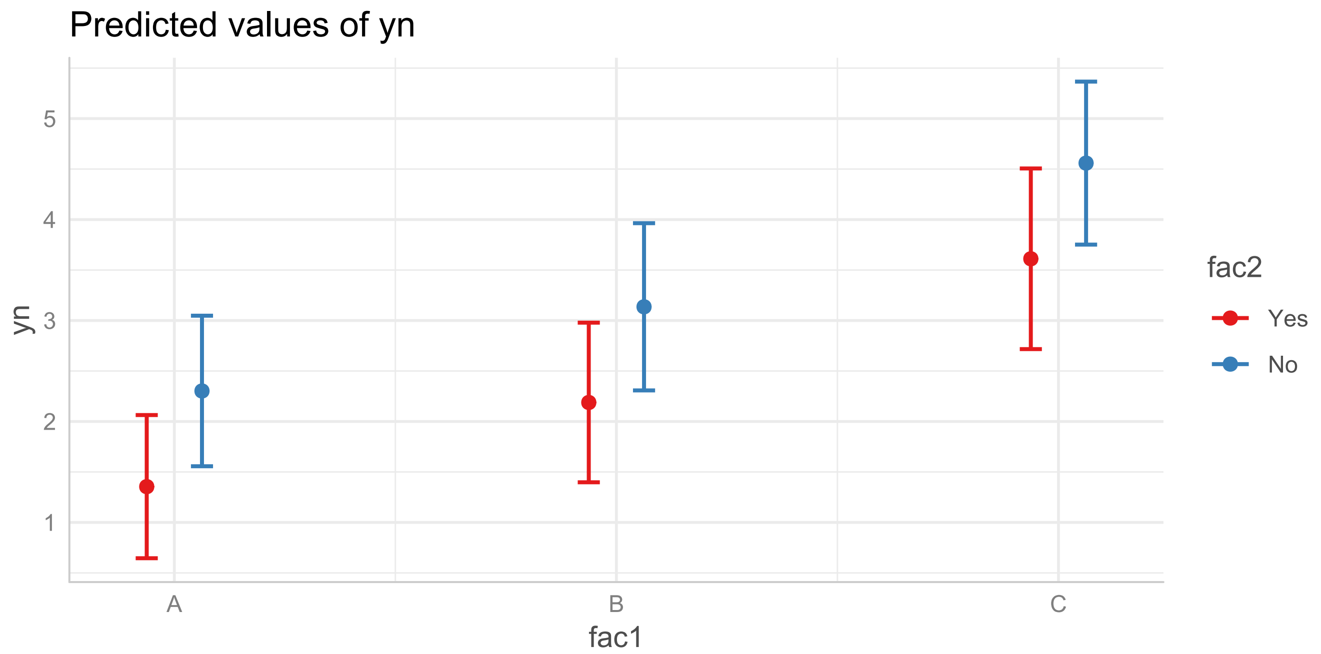
```
glm(yb ~ num1 + fac1 + num2 + fac2, data = ex_dat, family = binomial(link = "logit")) %>%
  ggpredict(terms = c("num1 [meansd]", "num2 [minmax]")) %>% plot(facet=TRUE)
```



Dot plots with `ggpredict()`

`ggpredict` will produce dot plots with error bars for categorical predictors.

```
lm(yn ~ fac1 + fac2, data = ex_dat) %>%
  ggpredict(terms=c("fac1", "fac2")) %>% plot()
```



Notes on `ggeffects`

There is a lot more to the `ggeffects` package that you can see in the package vignette and the github repository. This includes, but is not limited to:

- Predicted values for polynomial and interaction terms
- Getting predictions from models from dozens of other packages
- Sending `ggeffects` objects to `ggplot2` to freely modify plots

An Advanced Example

Here is an example using a model from a recent article I worked on.

This models the likelihood of arrest of a target in a police contact conditional on neighborhood, race of target, and race of who called the police.

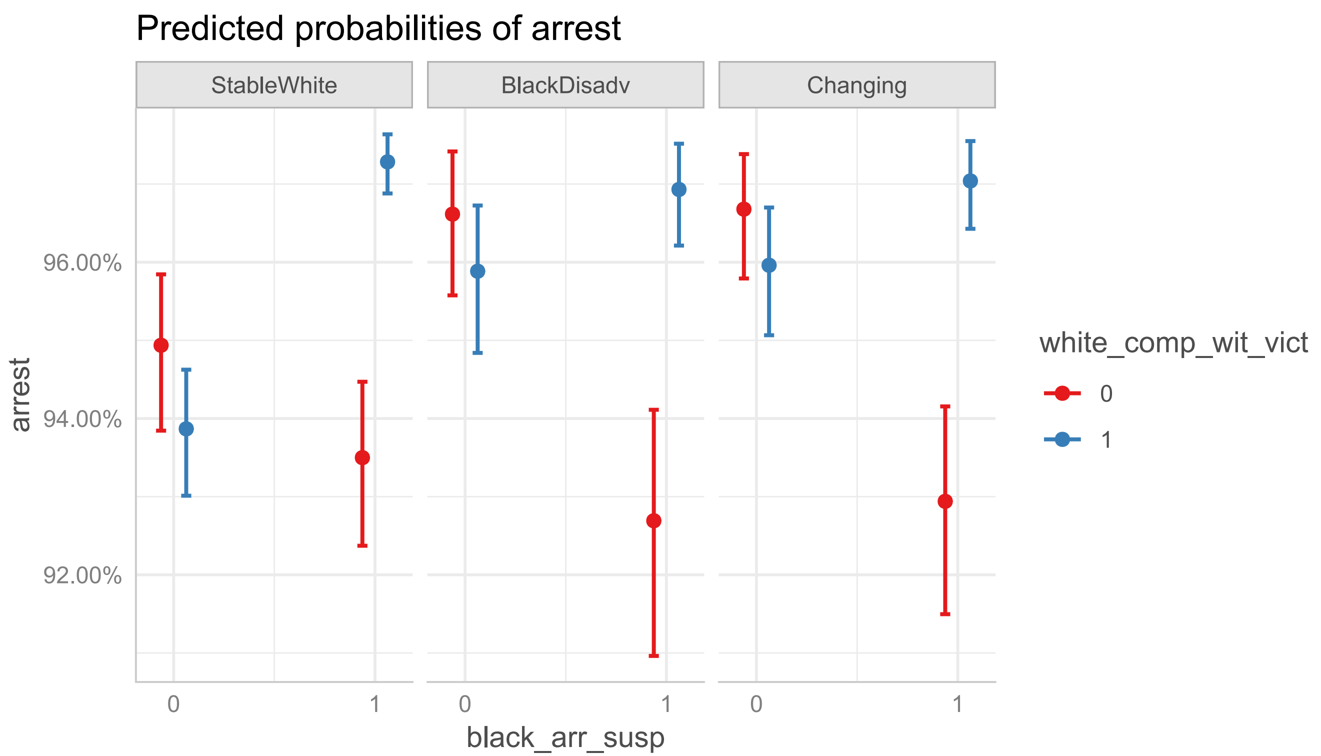
```
load("data/any_arrest_data.RData")
mod_arrest <- glm(arrest ~ white_comp_wit_vict*black_arr_susp +
                  crime_type*white_comp_wit_vict + caller_type +
                  arr_susp_subj_count + comp_wit_vict_count +
                  black_arr_susp*neighb_type + crime_type*neighb_type +
                  serious_rate + pbl + pot + dis + year,
                  family = binomial(link = "logit"),
                  data = any_arrest_data)
```

There are a lot of interactions here:

- Target Race x Caller Race
- Crime Type x Caller Race
- Target Race x Neigbhorhood Type
- Crime Type x Neighborhood Type

`ggeffects` Output

```
mod_arrest %>% ggpredict(terms = c("black_arr_susp",
              "white_comp_wit_vict", "neighb_type")) %>% plot()
```



A Complex Example

`ggpredict()` can only handle three variables in its `terms=` argument.

For my article, I wanted to plot estimates across counterfactual values of all four variables in my interaction terms:

- Caller Race
- Target Race
- Crime Type
- Neighborhood Type

How could I do this?

Stats + Math + Code = ♡

Some Background

Given we've estimate a model, consider the following:

1. $\hat{Y} = X\hat{\beta}$, where X is the model matrix and $\hat{\beta}$ is the coefficients.
2. $\hat{\beta}$ is a vector of *random variables* whose estimated distribution is described by parameter variance-covariance matrix Σ.

Using this, we can do the following:

1. Extract the model matrix X, estimated coefficients ($\hat{\beta}$), and Σ from our fitted model.
2. Make lots of random parameter draws centered on $\hat{\beta}$ and distributed according to Σ.
3. Multiply *each* of these draws by *counterfactual X values* to get $\hat{Y}$ values.
4. Take the 2.5% and 97.% quantiles of these $\hat{Y}$ values.

This produces a *simulated* mean and confidence interval. This is called the **percentile method**, a type of *bootstrapping*.

Simulating Coefficients

We can make random draws from our estimated distribution of parameters using `MASS::mvrnorm()` which takes three main arguments:

1. `n`: The number of draws
2. `mu`: mean—our coefficient estimates—obtained via `coef()`.
3. `Sigma`: a covariance matrix, obtained via `vcov()`.

```
sim_params <- MASS::mvrnorm(n = 10000,
                            mu = coef(mod_arrest),
                            Sigma = vcov(mod_arrest))
sim_params[1:6, 1:4]
```

```
##      (Intercept) white_comp_wit_vict1 black_arr_susp1 crime_typeNuisance
## [1,]    2.840401           -0.2829818      -0.3509852         -0.4236454
## [2,]    3.005853           -0.3728572      -0.4625391         -0.6269654
## [3,]    2.646754           -0.1211213      -0.1352515         -0.6166415
## [4,]    2.784774           -0.2907974      -0.3368231         -0.5356599
## [5,]    2.609253           -0.2215935      -0.2904826         -0.4866047
## [6,]    2.870690           -0.3185690      -0.4651584         -0.6933575
```

Counterfactual Values

Next we need a data frame with our counterfactual values.

We want one row (or *scenario*) per estimate to plot, and all variables at their means *except* the ones we are varying. We also don't want impossible values; `neighb_type` values are mutually exclusive.

```
x_values <- colMeans(model.matrix(mod_arrest)) # vars at mean
n_scen   <- (2*2*2*3) # Number of scenarios
x_frame  <- setNames(data.frame(matrix(x_values, nrow=n_scen,
                                       ncol=length(x_values),
                                       byrow=T)), names(x_values))
cf_vals  <- arrangements::permutations(c(0,1), k=5, replace=T)
cf_vals  <- cf_vals[cf_vals[,4]+cf_vals[,5]!=2 ,] # Remove impossible vals
colnames(cf_vals) <- c("white_comp_wit_vict1", "black_arr_susp1",
                       "crime_typeNuisance", "neighb_typeBlackDisadv",
                       "neighb_typeChanging")
x_frame[colnames(cf_vals)] <- cf_vals # assign to countefactual df
```

`permutations()` is a quick way to get all combinations of some values.

What Do We Have?

```
glimpse(x_frame)
```

```
## Observations: 24
## Variables: 24
## $ `(Intercept)`                              <dbl> 1, 1, 1, 1, 1, 1, ...
## $ white_comp_wit_vict1                       <dbl> 0, 0, 0, 0, 0, 0, ...
## $ black_arr_susp1                            <dbl> 0, 0, 0, 0, 0, 0, ...
## $ crime_typeNuisance                         <dbl> 0, 0, 0, 1, 1, 1, ...
## $ caller_typeVictim                          <dbl> 0.8019442, 0.80194...
## $ caller_typeWitness                         <dbl> 0.08516245, 0.0851...
## $ arr_susp_subj_count                        <dbl> 1.552955, 1.552955...
## $ comp_wit_vict_count                        <dbl> 1.571604, 1.571604...
## $ neighb_typeBlackDisadv                     <dbl> 0, 0, 1, 0, 0, 1, ...
## $ neighb_typeChanging                        <dbl> 0, 1, 0, 0, 1, 0, ...
## $ serious_rate                               <dbl> 2.157661e-17, 2.15...
## $ pbl                                        <dbl> 1.185962e-16, 1.18...
## $ pot                                        <dbl> -2.808814e-17, -2....
## $ dis                                        <dbl> 9.987197e-18, 9.98...
## $ year2009                                   <dbl> 0.2829368, 0.28293...
## $ year2010                                   <dbl> 0.1670504, 0.16705...
## $ year2011                                   <dbl> 0.09201842, 0.0920...
## $ year2012                                   <dbl> 0.1232284, 0.12322...
## $ `white_comp_wit_vict1:black_arr_susp1`     <dbl> 0.4452034, 0.44520...
## $ `white_comp_wit_vict1:crime_typeNuisance`  <dbl> 0.1614479, 0.16144...
## $ `black_arr_susp1:neighb_typeBlackDisadv`   <dbl> 0.04893835, 0.0489...
## $ `black_arr_susp1:neighb_typeChanging`      <dbl> 0.111691, 0.111691...
## $ `crime_typeNuisance:neighb_typeBlackDisadv` <dbl> 0.0154771, 0.01547...
## $ `crime_typeNuisance:neighb_typeChanging`   <dbl> 0.03300077, 0.0330...
```

Fixing Interactions

Our main variables are correct... but we need to make our interaction terms.

The interaction terms in the model matrix have specific form `var1:var2`.

Their counterfactual values are just equal to the products of their components.

```
x_frame <- x_frame %>%
 mutate(
  `white_comp_wit_vict1:black_arr_susp1`       = white_comp_wit_vict1*black_arr_susp1,
  `white_comp_wit_vict1:crime_typeNuisance`    = white_comp_wit_vict1*crime_typeNuisance,
  `black_arr_susp1:neighb_typeBlackDisadv`     = black_arr_susp1*neighb_typeBlackDisadv,
  `black_arr_susp1:neighb_typeChanging`        = black_arr_susp1*neighb_typeChanging,
  `crime_typeNuisance:neighb_typeBlackDisadv`  = crime_typeNuisance*neighb_typeBlackDisadv,
  `crime_typeNuisance:neighb_typeChanging`     = crime_typeNuisance*neighb_typeChanging,
  `black_arr_susp1:neighb_typeBlackDisadv`     = black_arr_susp1*neighb_typeBlackDisadv,
  `black_arr_susp1:neighb_typeChanging`        = black_arr_susp1*neighb_typeChanging)
```

Fixed

```
glimpse(x_frame)
```

```
## Observations: 24
## Variables: 24
## $ `(Intercept)`                                <dbl> 1, 1, 1, 1, 1, 1, ...
## $ white_comp_wit_vict1                         <dbl> 0, 0, 0, 0, 0, 0, ...
## $ black_arr_susp1                              <dbl> 0, 0, 0, 0, 0, 0, ...
## $ crime_typeNuisance                           <dbl> 0, 0, 0, 1, 1, 1, ...
## $ caller_typeVictim                            <dbl> 0.8019442, 0.80194...
## $ caller_typeWitness                           <dbl> 0.08516245, 0.0851...
## $ arr_susp_subj_count                          <dbl> 1.552955, 1.552955...
## $ comp_wit_vict_count                          <dbl> 1.571604, 1.571604...
## $ neighb_typeBlackDisadv                       <dbl> 0, 0, 1, 0, 0, 1, ...
## $ neighb_typeChanging                          <dbl> 0, 1, 0, 0, 1, 0, ...
## $ serious_rate                                 <dbl> 2.157661e-17, 2.15...
## $ pbl                                          <dbl> 1.185962e-16, 1.18...
## $ pot                                          <dbl> -2.808814e-17, -2....
## $ dis                                          <dbl> 9.987197e-18, 9.98...
## $ year2009                                     <dbl> 0.2829368, 0.28293...
## $ year2010                                     <dbl> 0.1670504, 0.16705...
## $ year2011                                     <dbl> 0.09201842, 0.0920...
## $ year2012                                     <dbl> 0.1232284, 0.12322...
## $ `white_comp_wit_vict1:black_arr_susp1`       <dbl> 0, 0, 0, 0, 0, 0, ...
## $ `white_comp_wit_vict1:crime_typeNuisance`    <dbl> 0, 0, 0, 0, 0, 0, ...
## $ `black_arr_susp1:neighb_typeBlackDisadv`     <dbl> 0, 0, 0, 0, 0, 0, ...
## $ `black_arr_susp1:neighb_typeChanging`        <dbl> 0, 0, 0, 0, 0, 0, ...
## $ `crime_typeNuisance:neighb_typeBlackDisadv`  <dbl> 0, 0, 0, 0, 0, 1, ...
## $ `crime_typeNuisance:neighb_typeChanging`     <dbl> 0, 0, 0, 0, 1, 0, ...
```

Estimates!

Then we just multiply our parameters by our counterfactual data:

```
sims_logodds <- sim_params %*% t(as.matrix(x_frame))
sims_logodds[1:6, 1:6]
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## [1,] 2.228627 2.603589 2.632709 1.804981 2.293199 1.982131
## [2,] 2.344299 2.558431 2.608216 1.717334 2.374612 1.882508
## [3,] 2.049694 2.427225 2.520807 1.433053 1.959780 2.014053
## [4,] 2.213538 2.640734 2.552005 1.677878 2.178344 1.946091
## [5,] 2.112706 2.475079 2.569706 1.626101 2.189936 2.013312
## [6,] 2.283927 2.609980 2.697191 1.590569 2.046767 1.788614
```

```
dim(sims_logodds)
```

```
## [1] 10000    24
```

Now we log-odds 10,000 estimates each (rows) of 24 counterfactual scenarios (columns).

Getting Probabilities

The model for this example is a *logistic regression*, which produces estimates in *log-odds* ($ln(Odds(x))$).

We can convert these to probabilities based on two identities:

1. $Odds(x) = e^{ln(Odds(x))}$
2. $Pr(x) = \frac{Odds(x)}{(1+Odds(x))}$

```
sims_prob    <- exp(sims_logodds) / (1 + exp(sims_logodds))
sims_prob[1:6, 1:6]
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.9027909 0.9310922 0.9329372 0.8587542 0.9083122 0.8789081
## [2,] 0.9124800 0.9281379 0.9313885 0.8477851 0.9148708 0.8678989
## [3,] 0.8859167 0.9188800 0.9255877 0.8073765 0.8765092 0.8822647
## [4,] 0.9014587 0.9334376 0.9277081 0.8426234 0.8982878 0.8750198
## [5,] 0.8921320 0.9223762 0.9288863 0.8356348 0.8993421 0.8821877
## [6,] 0.9075371 0.9315011 0.9368607 0.8306962 0.8856205 0.8567573
```

A Quick Function

We are going to want to grab the mean and 95% confidence interval from our simulation estimates.

Here's a quick function to do it and make it pretty.

```
extract_pe_ci <- function(x){
  vals <- c(mean(x), quantile(x, probs=c(.025, .975)))
  names(vals) <- c("PE", "LB", "UB")
  return(vals)
}
```

This returns a length 3 vector with the following names:

- **PE** for *point estimate*
- **LB** for *lower bound* of the confidence interval
- **UB** for *upper bound*

Prep for Plotting

First we extract our point estimates and confidence intervals by *applying* `extract_pe_ci()` to each column of estimated probabilities.

```
estimated_pes <- as.data.frame( t(apply(sims_prob, 2, extract_pe_ci)))
```

Then I add columns describing the scenarios to color, group, and facet over based on the counterfactual values.

```
estimated_pes$`Reporter`     <- ifelse(cf_vals[,1]==1, "Any White", "All Black")
estimated_pes$`Target`       <- ifelse(cf_vals[,2]==1, "Any Black", "All White")
estimated_pes$`Crime Type`   <- ifelse(cf_vals[,3]==1, "Nuisance Crime", "Serious Crime")
estimated_pes$`Neighborhood` <- case_when(
  cf_vals[,4]==1 ~ "Disadvantaged",
  cf_vals[,5]==1 ~ "Changing",
  TRUE ~ "Stable White")
```

Final Tidy Data

```
estimated_pes %>% mutate_if(is.numeric, round, digits=3) # round for display
```

```
##       PE    LB    UB  Reporter    Target     Crime Type  Neighborhood
## 1  0.894 0.877 0.908 All Black All White  Serious Crime  Stable White
## 2  0.929 0.914 0.942 All Black All White  Serious Crime      Changing
## 3  0.927 0.909 0.943 All Black All White  Serious Crime Disadvantaged
## 4  0.831 0.801 0.858 All Black All White Nuisance Crime  Stable White
## 5  0.897 0.873 0.919 All Black All White Nuisance Crime      Changing
## 6  0.879 0.844 0.909 All Black All White Nuisance Crime Disadvantaged
## 7  0.866 0.851 0.879 All Black Any Black  Serious Crime  Stable White
## 8  0.855 0.834 0.874 All Black Any Black  Serious Crime      Changing
## 9  0.850 0.824 0.873 All Black Any Black  Serious Crime Disadvantaged
## 10 0.790 0.758 0.819 All Black Any Black Nuisance Crime  Stable White
## 11 0.799 0.761 0.833 All Black Any Black Nuisance Crime      Changing
## 12 0.765 0.711 0.814 All Black Any Black Nuisance Crime Disadvantaged
## 13 0.873 0.866 0.880 Any White All White  Serious Crime  Stable White
## 14 0.914 0.900 0.927 Any White All White  Serious Crime      Changing
## 15 0.912 0.895 0.928 Any White All White  Serious Crime Disadvantaged
## 16 0.749 0.731 0.766 Any White All White Nuisance Crime  Stable White
## 17 0.842 0.811 0.869 Any White All White Nuisance Crime      Changing
## 18 0.816 0.773 0.852 Any White All White Nuisance Crime Disadvantaged
## 19 0.941 0.937 0.945 Any White Any Black  Serious Crime  Stable White
## 20 0.936 0.927 0.945 Any White Any Black  Serious Crime      Changing
## 21 0.934 0.922 0.944 Any White Any Black  Serious Crime Disadvantaged
## 22 0.875 0.863 0.885 Any White Any Black Nuisance Crime  Stable White
## 23 0.880 0.856 0.900 Any White Any Black Nuisance Crime      Changing
## 24 0.857 0.824 0.886 Any White Any Black Nuisance Crime Disadvantaged
```

Plot Code

Finally we plot estimates (`PE`) as points with error bars (`UB`, `LB`) stratified on `Target` and `Reporter` and faceted by `Crime Type` and `Neighborhood`.

```
ggplot(estimated_pes, aes(x = Target, y = PE, group = Reporter)) +
  facet_grid(`Crime Type` ~ Neighborhood) +
  geom_errorbar(aes(ymin = LB, ymax = UB),
                position = position_dodge(width = .4),
                size = 0.75, width = 0.15) +
  geom_point(shape = 21, aes(fill = Reporter),
             position = position_dodge(width = .4),
             size = 2) +
  scale_fill_manual("Reporter", values=c("Any White" = "white",
                                         "All Black" = "black")) +
  ggtitle("Figure 3. Probability of Arrest",
      subtitle = "by Reporter and Target Race, Neighborhood and Crime Type") +
  xlab("Race of Target") + ylab("Estimated Probability") +
  theme_bw() + theme(legend.position = c(0.86,0.15),
                     legend.background = element_rect(color = 1))
```

Plot

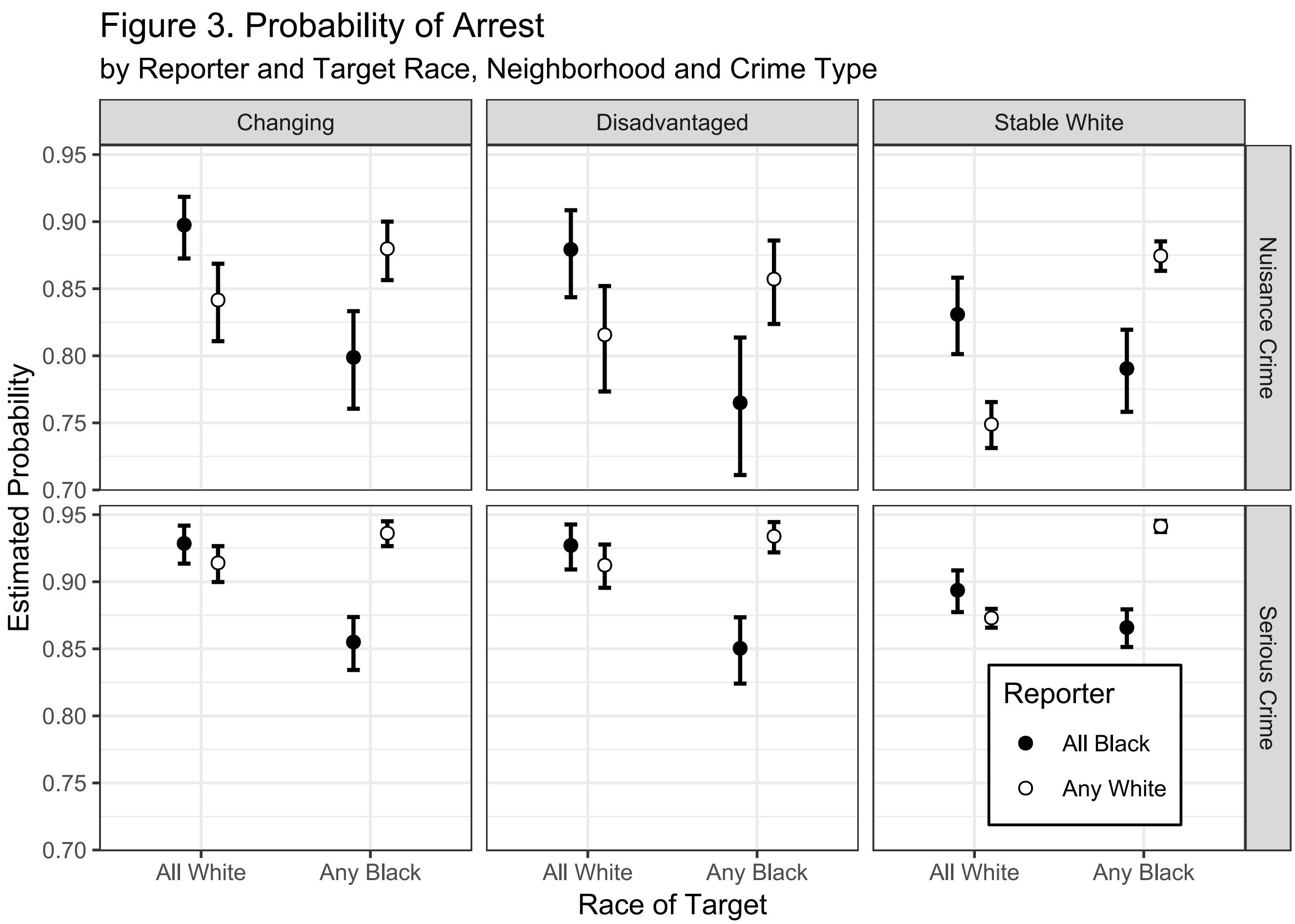


Figure 3. Probability of Arrest
by Reporter and Target Race, Neighborhood and Crime Type
Changing
Disadvantaged
Stable White
Nuisance Crime
Serious Crime
Estimated Probability
0.95
0.90
0.85
0.80
0.75
0.70
All White
Any Black
Race of Target
Reporter
All Black
Any White

Making Tables

`pander` Regression Tables

We've used `pander` to create nice tables for dataframes. But `pander` has *methods* to handle all sort of objects that you might want displayed nicely.

This includes model output, such as from `lm()`, `glm()`, and `summary()`.

```r
library(pander)
```

`pander()` and `lm()`

You can send an `lm()` object straight to `pander`:

```
pander(lm_1)
```

	Estimate	Std. Error	t value	Pr(>t)
(Intercept)	37.23	1.599	23.28	2.565e-20
wt	-3.878	0.6327	-6.129	1.12e-06
hp	-0.03177	0.00903	-3.519	0.001451

Table: Fitting linear model: mpg ~ wt + hp

`pander()` and `summary()`

You can do this with `summary()` as well, for added information:

```
pander(summary(lm_1))
```

	Estimate	Std. Error	t value	Pr(>t)
(Intercept)	37.23	1.599	23.28	2.565e-20
wt	-3.878	0.6327	-6.129	1.12e-06
hp	-0.03177	0.00903	-3.519	0.001451

Observations	Residual Std. Error	R^2	Adjusted R^2
32	2.593	0.8268	0.8148

Table: Fitting linear model: mpg ~ wt + hp

`sjPlot`

`pander` tables are great for basic `rmarkdown` documents, but they're not generally publication ready.

The `sjPlot` package produces `html` tables that look more like those you may find in journal articles.

```
library(sjPlot)
```

`sjPlot` Tables

`tab_model()` will produce tables for most models.

```
model_1 <- lm(mpg ~ wt, data = mtcars)
tab_model(model_1)
```

	mpg		
	B	*CI*	*p*
(Intercept)	37.29	33.45 – 41.12	<.001
wt	-5.34	-6.49 – -4.20	<.001
Observations	32		
R^2 / adj. R^2	.753 / .745		

Multi-Model Tables with `sjTable`

Often in journal articles you will see a single table that compares multiple models.

Typically, authors will start with a simple model on the left, then add variables, until they have their most complex model on the right.

The `sjPlot` package makes this easy to do: just give `tab_model()` more models!

Multiple `tab_model()`

```
model_2 <- lm(mpg ~ hp + wt, data = mtcars)
model_3 <- lm(mpg ~ hp + wt + factor(am), data = mtcars)
tab_model(model_1, model_2, model_3)
```

	mpg			mpg			mpg		
	B	*CI*	*p*	*B*	*CI*	*p*	*B*	*CI*	*p*
(Intercept)	37.29	33.45 – 41.12	<.001	37.23	33.96 – 40.50	<.001	34.00	28.59 – 39.42	<.001
wt	-5.34	-6.49 – -4.20	<.001	-3.88	-5.17 – -2.58	<.001	-2.88	-4.73 – -1.02	.004
hp				-0.03	-0.05 – -0.01	.001	-0.04	-0.06 – -0.02	<.001
factor(am) (1)							2.08	-0.74 – 4.90	.141
Observations	32			32			32		
R^2 / adj. R^2	.753 / .745			.827 / .815			.840 / .823		

`sjPlot` does a lot more

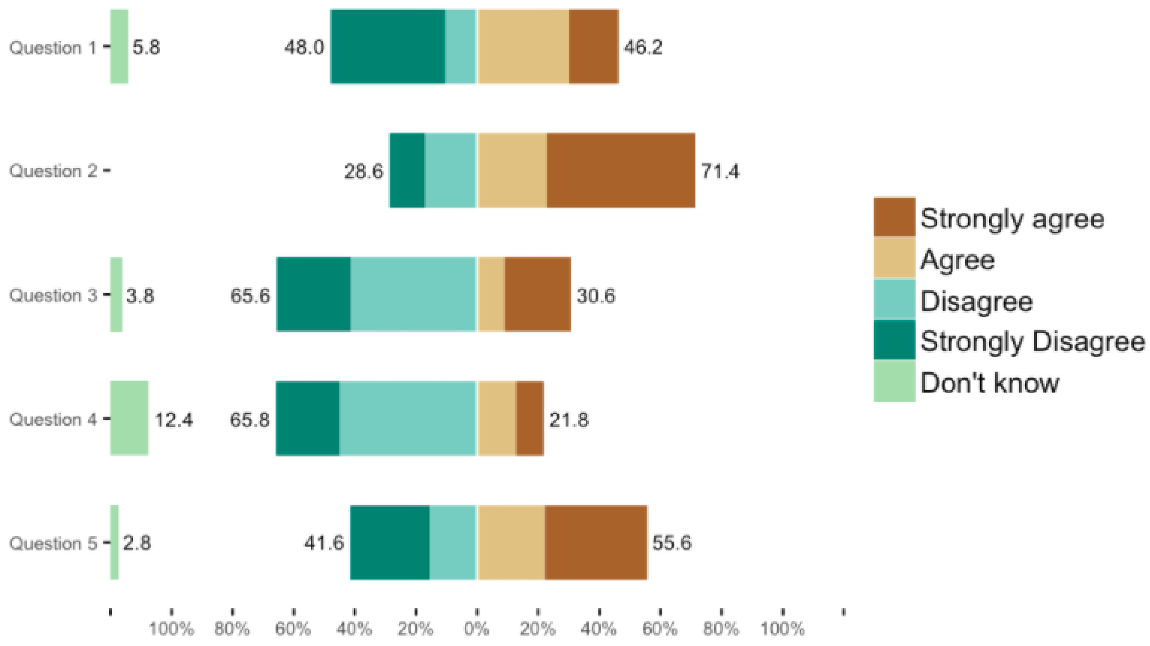
The `sjPlot` package does *a lot* more than just make pretty tables. It is a rabbit hole of *incredibly* powerful and useful functions for displaying descriptive and inferential results.

View the package website for extensive documentation.

`sjPlot` is a bit more complicated than `ggeffects` but can do just about everything it can do as well; they were written by the same author!

`sjPlot` is fairly new but offers a fairly comprehensive solution for `ggplot` based publication-ready social science data visualization. All graphical functions in `sjPlot` are based on `ggplot2`, so it should not take terribly long to figure out.

sjPlot Example: Likert plots



sjPlot Example: Crosstabs

elder's dependency	*carer's level of education* low level of education	 intermediate level of education	 high level of education	*Total*
independent	21 19.6 % 1.4 %	76 71 % 5.1 %	10 9.3 % 0.7 %	107 100 % 7.2 %
slightly dependent	72 19 % 4.9 %	238 63 % 16.1 %	68 18 % 4.6 %	378 100 % 25.6 %
moderately dependent	106 21.3 % 7.2 %	289 58 % 19.5 %	103 20.7 % 7 %	498 100 % 33.7 %
severely dependent	118 23.7 % 8 %	296 59.4 % 20 %	84 16.9 % 5.7 %	498 100 % 33.7 %
Total	317 21.5 % 21.5 %	899 60.7 % 60.7 %	265 18 % 18 %	1481 100 % 100 %

X^2=8.658 · df=6 · Φ_c=.072 · p=.194

LaTeX Tables

For tables in $\LaTeX$—as is needed for `.pdf` files—I recommend looking into the `gt`, `stargazer`, or `kableExtra` packages.

`gt` and `kableExtra` allow the construction of complex tables in either HTML or $\LaTeX$ using additive syntax similar to `ggplot2` and `dplyr`.

`stargazer` produces nicely formatted $\LaTeX$ tables but is idiosyncratic.

If you want to edit $\LaTeX$ documents, you can do it in R using Sweave documents (.Rnw). Alternatively, you may want to work in a dedicated $\LaTeX$ editor. I recommend Overleaf for this purpose.

RMarkdown has support for a fair amount of basic $\LaTeX$ syntax if you aren't trying to get too fancy!

Another approach I have used is to manually format $\LaTeX$ tables but use in-line R calls to fill in the values dynamically. This gets you the *exact* format you want but without forcing you to update values any time something changes.

Bonus: `corrplot`

The `corrplot` package has functions for displaying correlograms.

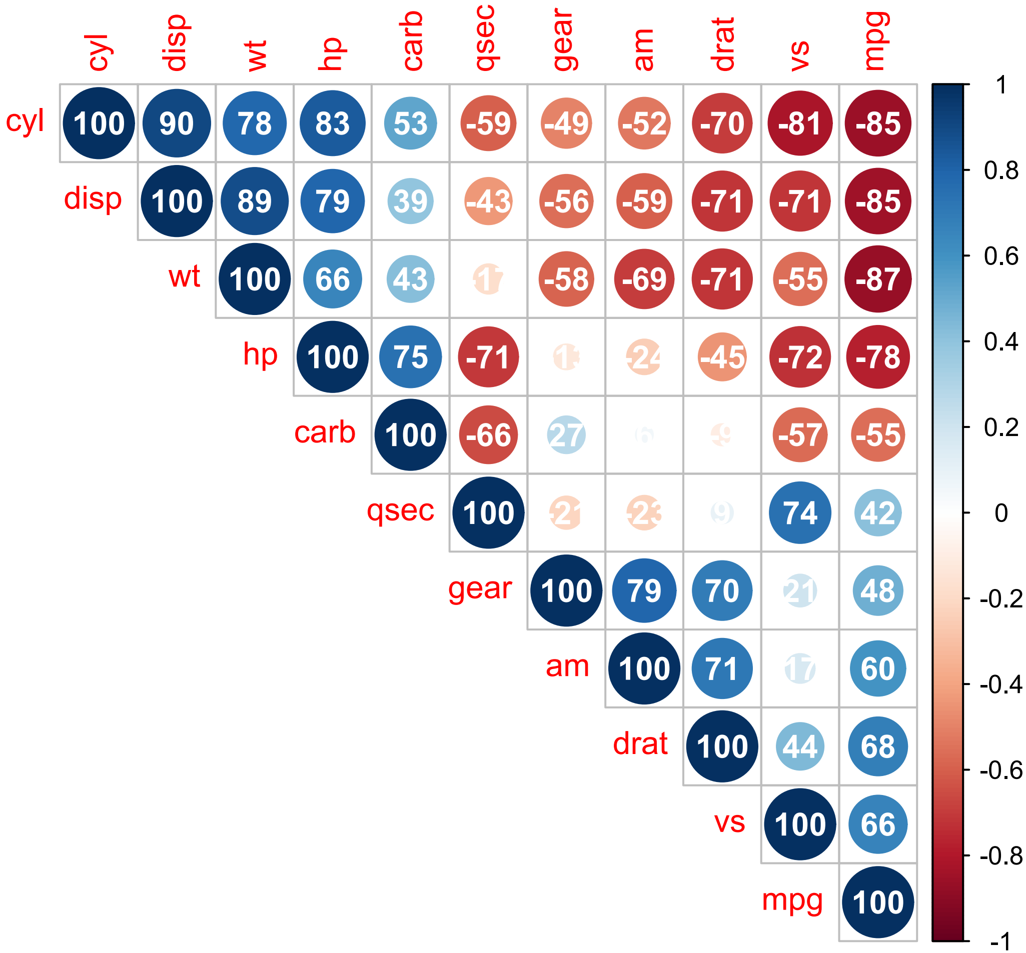
These make visualizing the correlations between variables in a data set easier.

The first argument is a call to `cor()`, the base R function for generating a correlation matrix.

See the vignette for customization options.

```
library(corrplot)
corrplot(
  cor(mtcars),
  addCoef.col = "white",
  addCoefasPercent=T,
  type="upper",
  order="AOE")
```

Correlogram



Wrapping up the Course

What You've Learned

A lot!

- How to get data into R from a variety of formats
- How to do "data custodian" work to manipulate and clean data
- How to make pretty visualizations
- How to automate with loops and functions
- How to combine text, calculations, plots, and tables into dynamic R Markdown reports
- How to acquire and work with spatial data

What Comes Next?

- Statistical inference (e.g. more CSSS courses)
  - Functions for hypothesis testing, hierarchical/mixed effect models, machine learning, survey design, etc. are straightforward to use… once data are clean
  - Access output by working with list structures (like from regression models) or using `broom` and `ggeffects`
- Practice, practice, practice!
  - Replicate analyses you've done in Excel, SPSS, or Stata
  - Think about data using `dplyr` verbs, tidy data principles
  - R Markdown for reproducibility
- More advanced projects
  - Using version control (git) in RStudio
  - Interactive Shiny web apps
  - Write your own functions and put them in a package

Course Plugs

If you...

- have no stats background yet - **SOC504: Applied Social Statistics**
- want to learn more social science computing - **SOC590: Big Data and Population Processes** [1]
- have (only) finished SOC506 - **CSSS510: Maximum Likelihood**
- want to master visualization - **CSSS569: Visualizing Data**
- study events or durations - **CSSS544: Event History Analysis** [2]
- want to use network data - **CSSS567: Social Network Analysis**
- want to work with spatial data - **CSSS554: Spatial Statistics**
- want to work with time series - **CSSS512: Time Series and Panel Data**

[1] We're hoping to offer that again soon!
[2] Also a great maximum likelihood introduction.

Thank you!