STAT 301

A/B Testing (2-sample Hypothesis Tests)

- test to compare the param of two pop: control (A) & variation (B)
  - basically 2-sample hypothesis test (t–test if population sd σ is unknown, z–test if known)
  - need to randomly assign ppl to make it an experiment
- hypothesis test: always hypothesizing about the population param
  - H_0 is the null hypothesis, it refers to the status quo (i.e there's no change in engagement) - **it's always an equality**
  - H_1 is the alternative hypothesis, it's the researcher's hypothesis of interest (i.e there's in increase in engagement)

Means

- code to test $H_a : \mu_1 > \mu_2$ (mean of sample 1 vs sample 2)

```
# this is testing is x > y (so x - y > 0)
result <- tidy(t.test(x = sample1, y = sample2
    alternative = "greater", # or  "two.sided" or "lesser"
    var.equal = FALSE)) # TRUE if assumed equal var
```

- from `result` – we can get the p–val, if it's less than significance level, we reject (p is low, null must go – it's evidence against the null)
- another way to get the p-value is **bootstrapping**, the code is

```
obs_test_stat <- tiktok_sample %>% # test stat from og sample
    specify(formula = response_var ~ explantory_var) %>%
    calculate(stat = "diff in means", # it's new - current
        order = c("new", "current")) %>% pull()
pvalue = tiktok_sample %>% # get p-value from test stats
    specify(formula = response_var ~ explantory_var) %>%
    hypothesize(null = "independence") %>%
    generate(reps = 1000, type = "permute") %>%
    calculate(stat = "diff in means",
        order = c("new", "current")) %>%  # compare resample
    get_p_value(obs_stat = obs_test_stat, #  test stat against
        direction = "greater") %>% pull() #  og test stat
```

- errors in hypothesis testing
  - **Type I error**: reject H_0 when H_0 is true, denoted α
  - **Type II error**: fail to reject H_0 when H_0 is false, denoted β
  - α = significance level is set by experiment designer (i.e is **fixed**)
- **power**: prob of rejecting H_0 when H_0 is False $\to$ Power $= 1 - \beta$
  - larger (sample size, effect size, α) = bigger power
  - effect size is the difference between hypothesized and true mean
  - smaller variance = bigger power

Proportions

- when doing A/B testing for proportions - it's two-sample z-test
  - ex. $H_0 : p_A = p_B$ vs. $H_1 : p_A > p_B$
- if the samples are dependent - do pairwise test

```
pairwise_comparisons <- pairwise.prop.test(
    x = successes, # x is vector of counts of successes
    n = trials,    # n is vector of counts of trials
    p.adjust.method = "bonferroni", alternative = "two.sided")
```

- if just doing two-sample proportion test

```
result <- prop.test(x = c(success_group1, success_group2),
    n = c(total_group1, total_group2),
    alternative = "two.sided")
```

- Bonferroni adjustment: needed for pairwise to prevent increased Type I error (see later)
  - how: do $\alpha_{\text{new}} = \alpha/\#$ of comparisons (ex. 0.05/10)

Early Stopping & Principled Peeking

- some platforms allow users to continuously monitor the p-values and CI as you get more people (sample size changes/increases)
- big point: stopping an experiment & rejecting H_0 as soon as the p-value falls below specified α can drastically inflate Type I error
- principled peeking: methods to solve the early-stopping problem
  - a basic way is Bonferroni method – see above
  - other is Pocock & O'Brien-Fleming, but all of them can be thought of as reducing α or increasing critical val to reject less
- common experiment to demonstrate FWER error is A/A testing
  - you have 2 samples and you **know** their parameters are identical, so every time you reject, you know you're incorrectly rejecting
- code for principled peeking (A/A testing)

```
crit_unadj <- qt(1 - alpha, n) # getting critical value
crit_bonferroni <- qt(1 - (alpha/num_experiments), n)
exp_design <- gsDesign(
    k = num_experiments,  # Number of interim analyses
    test.type = 1,        # One-sided test
    alpha = 0.05,         # Significance level
    beta = 0.2,           # 1 - power
    sfu = "Pocock"/"OF",  # Pocock or O'Brien Fleming
    n.fix = 1             # This adjusts the sample size
    ↪  calculation; for A/A testing, effect size = 0
)
crit_adj_pocock_or_of <- exp_design$upper$bound
```

  - only O'Brien changes `crit_val` as n inc (starts very high)
  - Pocock is less conservative than Bonferonni
  - α : unadj > Pocock > Bonferroni (opposite for critical-value)
  - all of these affect (decreases) power of the test ($\alpha \propto 1/\text{power}$)
  - Bonferroni bad for seq test bc too conservative & assume indep

Simple Linear Regression

Intro

- simple here means only 1 input variable
  - Y: response variable, dependent variable, output
  - X: explanatory variables, independent variables, features
- the model:

$$Y = \beta_0 + \beta_1 + \varepsilon \qquad E[\varepsilon \mid X] = 0$$

  - ε is the error term - contains all other factors affecting Y
- β_0 is the (true) intercept variable, β_1 is the (true) slope variable
  - we will estimate these using `lm()` and denote it $\hat{\beta}_0, \hat{\beta}_1$

Estimation of LR Line

- methods we use to get the linear regression line is **Least Squares** (LSE) method – minimize sum of squares of the residuals (e_i^2)
- code: do `lm(response_var ~ explanatory_var, data)`
  - we get back a model that has the estimated values $\hat{\beta}_0$ and $\hat{\beta}_1$
- plotting the line: add `geom_smooth(method="lm", se=FALSE)`
- say "for every 1 unit increase of x, y is expected to inc/dec by β_1"

SLR Inference (Hypothesis Test + CI)

- variation: estimates of β_0 and β_1 depends on our sample (they may vary between our samples)
  - variation of these estimates from sample to sample is called standard error (note: we can't take multiple samples IRL)
  - in reality: take 1 sample & compute SE via sampling dist
- hypothesis testing: question is "is the input variable linearly associated with the response"

$$H_0 : \beta_i = 0 \qquad H_A : \beta_i \neq 0$$

  - ex. null is saying the slope β_1 is 0, corresponds to the assumption that there is no linear relationship between X and Y
  - we can check this using `tidy(lm_model, conf.int=TRUE)`
    - give 95% CI as well (change `conf.level` as needed)
    - this is t-test: if p–value is low, reject the null
    - if we reject, we say the variables are statistically associated
  - note: high p–value means we don't have strong evidence to reject $\beta_i = 0$, doesn't mean that $\beta_i = 0$ or that they're not associated
- sampling distribution: there are 2 ways to get it
  - theoretical: this is what `lm()` does (**and what we focus on**)

$$T = \frac{\hat{\beta}_1 - 0}{\text{SE}(\hat{\beta}_1)} \sim t_{n-k} \qquad k = \# \text{ of regression param}$$

    - use `qt()` like above to get test stat and p–value
  - bootstrapping: resample w/ rep from OG sample, calc estimate, get list of estimates, get our estimated sampling dist

```
bootstrap_sample <- replicate(num_replicate, {
    sample_n(og_sample, sample_size, replace=TRUE) %>%
    lm(y_var ~ x_var, data = .) %>% .$coef
})
bootstrap_sample <- data.frame(
    boot_intercept = bootstrap_sample[1, ],
    boot_slope = bootstrap_sample[2, ]
)
# or instead you can use infer
bootstrap_slope_infer <- original_sample %>%
  specify(formula = y_var ~ x_var) %>%
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "slope")
head(bootstrap_slope_infer)
visualize(bootstrap_slope_infer, bin=30) # histogram
```

  - larger `num_replicate` means better approximation of dist
  - larger og sample size means sampling dist tighter & more smooth
  - note: the sampling dist is about the statistics (i.e stuff with hats) and SLR sampling dist is related to $\hat{\beta}_i$
- confidence interval
  - theoretical approach: $CI = \hat{b} \pm SE(\hat{b}) \times t_{\alpha/2, n-k}$ (use `tidy()`)
  - bootstrap (percentile) approach: get the sampling distribution, run `quantile()` or `infer` on it to get the 95% CI

```
# using quantile
CI <- boostramp sample %>%
    summarize(lower = quantile(boot_slope,0.025),
              upper = quantile(boot_slope,0.975))
# using infer
percentile_ci <- bootstrap_slope_infer %>%
get_confidence_interval(type = "percentile", level=0.95)
```

  - interpretation: given a CI, we say "we are 95% **confident** that the true coefficient is in the given range"
  - note: CI not including 0 also show statistical significance

Multiple Linear Regression

Additive models (Continuous Variable)

- we use the + in `lm()` function to indicate additive models

```
# povertyPercent and PctPrivateCoverage are continuous
MLR_poverty_coverage <- lm(formula=TARGET_deathRate ~
↪  povertyPercent + PctPrivateCoverage, US_cancer_data)
```

- after calling `tidy()` on these we'll get p–value and estimates of each and we can interpret them **separately**
  - i.e we can reject the null btw y and x_1 or reject the null btw y and x_2 but nothing about their combination
  - ex. if estimate for $x_1 = 0.5$, means that for one-unit inc in x_1, predict a 0.5 unit inc in y, holding all other variables constant

Additive Models (Categorical + Continuous Variables)

- here, we'll look at adding 1 cont. variable and 1 categorical variable
- bc model is additive (no interaction), **they share a common slope**
- to rep categorical var - we'll use dummy variables (turn on or off)
  - if a categorical var has k levels, you need $k - 1$ dummy var
- case study: continue with the example of the 2 states

$$Y_i = \beta_0 + \beta_1 \text{isWashington} + \beta_2 \text{povertyPercent} + \varepsilon_i$$

here, Indiana is called the **baseline** (it's alphabetical)

$$\begin{aligned} \text{in Indiana} : Y_i &= \beta_0 + \beta_1(0) + \beta_2\text{povertyPercent} + \varepsilon_i \\ &= \beta_0 + \beta_2\text{povertyPercent} + \varepsilon_i \\ \text{in Washington} : Y_i &= \beta_0 + \beta_1(1) + \beta_2\text{povertyPercent} + \varepsilon_i \\ &= (\beta_0 + \beta_1) + \beta_2\text{povertyPercent} + \varepsilon_i \end{aligned}$$

  - code: `lm(y ~ state + povertyPercent, data = cancer_data)`
- interpretation
  - β_0 is intercept of the **reference** line.
  - β_1 (coefficient of the dummy variable) is the **difference** between intercepts of both lines (inc/dec when going from Indy to Wash)
  - β_2 is the **common** slope of both lines

Interactive Model (Categorical + Continuous Variable)

- say we believe slope changes btw states too (add interaction terms)
- can interpret following as 2 separate LR in 1 eq, representing 2 lines

$$Y_i = \beta_0 + \beta_1(\text{isWashington}) + \beta_2(\text{povertyPercent}) + \beta_3(\text{isWashington} \times \text{povertyPercent}) + \varepsilon_i$$

  - in Indiana: `isWashington = 0`

$$Y_i = \beta_0 + \beta_1(0) + \beta_2(\text{povertyPercent}) + \beta_3(0) + \varepsilon_i = \beta_0 + \beta_2(\text{povertyPercent}) + \varepsilon_i$$

  - in Washington

$$Y_i = (\beta_0 + \beta_1) + (\beta_2 + \beta_3)\text{povertyPercent} + \varepsilon_i$$

  - code: `lm(y ~ state * povertyPercent, data = cancer_data)`
- interpretation
  - β_0/β_2 is the intercept/slope of the **reference** line
  - β_1/β_3 is the difference in intercept/slopes between both lines (ex. for Wash the intercept is $(\beta_0+\beta_1)$ and the slope is $(\beta_2+\beta_3)$)
- note: choice of baseline doesn't affect $\hat{y}$, it changes interpretation (ex. can only speak about stat significance of slope for the baseline)
- largest model: number of scenarios × number of continuous vars

LR: Model Assumption & Diagnostic

1. Linear relation: is it LR
   - "linear"= linear in terms of linear combination of the variables
   - consequence: regression line might not fit (represent) data well
   - diagnostic: look at residuals vs. fitted values plot - should look random without a pattern for a good fit
2. Errors are independent
   - can be assessed from the design of the study → randomization and repeated experiments ensure the independence of errors
   - consequence: estimates may be underestimated, which can lead to falsely significant p-values
   - ex. time series data is not independent
3. Is the conditional distribution of the errors Normal?
   - note: errors don't have to be Normal for valid inference result → approximate sampling dist via CLT (large n) or bootstrapping
   - diagnose: use Q-Q plot - straight line on QQ plot is good
   - consequence: the estimates of params are not heavily affected, but the SE are large (which we rely on to create distributions)
   - fix: variables transformations can be used as possible "remedies"
4. Equal variance of the error terms (aka homoscedasticity)
   - diagnose: use residuals-fitted plot - funnel shape = bad
   - consequence: the estimates will be similar to the true population parameters but their estimated standard errors will be inflated
   - fix: try transforming the variable
5. Multicollinearity: occurs when (some of) input var are correlated
   - when that happens, the information of one variable can be masked by another variable carrying correlated data
   - diagnose: check correlation (`cor(x,y)`) bt var via pariwise plots
   - diagnose: detect multi-col using variance inflation factor (VIF)
     - if $VIF_j \gg 1$, there's multicollinearity w/ x_j in the data
     - code: `vif(lr_model)` – remove the ones w the highest score
   - consequences:
     - inflates SE of the regression estimator
     - cause instability in the regression coefficients (small changes to data result in large changes in the coefficients)
     - mask true importance of predictors by inflating coefficients/significance of correlated var (unreliable results)
   - fix: select which variable (among the correlated ones) to keep

Statistical Design and Causality

- confounding factors: **confounder** is a variable that causes changes in both the response **and** at least one input variable
  - consequences: distort observed relationship btw vars, lead to false conclusions (inaccurate estimates, doesn't generalize well)
  - fix: different approach bt observational & experiment (see later)
- confounder example: explanatory variable is exercise amount, response variable is risk of heart disease, confounding variable is diet
  - healthier diets correlate with more exercise and lower heart disease risk; but we may incorrectly attribute heart disease risk reduction to exercise alone; accounting for diet in analysis clarifies exercise's true effect on heart disease
- casual inference: establishing causal effects (challenging)
  - depends on how data is collected & stat method used for analysis
- Completely Randomized Design (CRD): experimental units are randomized throughout the data layout → can establish causal rela
  - observed and unobserved confounders are balanced, on average
  - considered the gold standard design for causal inference
- Randomized Block Design (RBD): splits experimental units into homogeneous blocks to remove variation from nuisance factors, then randomly assigns treatments to each block
  - ex. subjects of similar age groups are blocks, grouping by sex
  - in RBD, only observed confounders are balanced so only average treatment effects can be estimated (via appropriate methods)
- observational data: we collect data by measuring variables or surveying members without applying any treatment to them
  - treatments not controlled by design - **causal effects can not be naively established** (need to include confounder in model)

Goodness of Fit & Nested Models

- the residual: defined as $\text{res}_i = y_i - \hat{y}_i$
  - where $\hat{y}_i$ is the predicted value $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$
  - in R we can do `lm.fitted` and `lm.res`
- quantites used
  - ESS (explained SS): if model better than nothing, this would be large, measures how much var in data explained by LR
  - RSS (residuals SS): our LR algo try to minimize this
  - TSS (total SS): res SS from the null model, $TSS = ESS + RSS$
- goodness of fit is asking "is our model better than the null model"
  - null model is the intercept only model (where the intercept is estimated by the sample mean $\bar{y}$ bc $E[Y] = \bar{y}$)
  - null model in R is `lm(response ~ 1)`
- **coefficient of determination** R^2: interpreted as the proportion of variance of the response explained by the model
  - math: $R^2 = 1 - RSS/TSS = ESS/TSS$
  - ranges from 0 to 1, closer to 1 is better
  - in R, can get all summary statistics by doing `glance(model)`
  - limitation: can't use to say how good model is at extrapolating
  - limitation: inc as new var added to the model (regardless of relevance) - can't be used to compare models of diff size
- **adjusted** R^2: accounts for model complexity (penalize added var)
  - math: adjusted $R^2 = 1 - \frac{RSS/(n-p)}{TSS/(n-1)}$
  - p is number of coefficients (variables) and n is sample size
  - like R^2, closer to 1 is better
- **Residual Standard Error** RSE
  - math: $RSE = \text{sqrt}\left(\frac{1}{n-p} \times RSS\right)$
  - called `sigma` in `glance(model)`
  - estimates the standard deviation of the error term ε
  - smaller is better
- **Mean Squared Error** MSE
  - math: $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$
  - often used in cases where the focus is on prediction accuracy
  - training MSE: `.resid` column in the `augment(model)` output
  - testing MSE: compute on new data and predicted value, use to evaluate out-of-sample prediction performance
  - $RMSE$ = root mean square error = $\sqrt{MSE}$
- about `glance()`: the H_0 it's testing is if **all** predictors are 0 (ANOVA - i.e our model is not better than the null)

```
my_glance <- data %>% summarize(
    RSS = sum(lm_model$resid^2),
    TSS = sd(data$response)^2 * (n - 1),
    R2 = 1 - RSS/TSS,
    adjR2 = 1- (RSS/(n-2)) / (TSS/(n-1)),
    sigma = sqrt(RSS/(n-2)))
```

The F-Test (ANOVA)

- used to compare model of different size (i.e full vs reduced model)
- hypothesis test: testing if any of the additional variables are zero

$$H_0 : \beta_{q+1} = \beta_{q+2} = \ldots = \beta_s = 0$$
$$H_1 : \text{at least one } \beta_j \neq 0 \quad (\text{for } j = q+1, q+2, \ldots, s)$$

- in R (NB: `glance()` is also doing F-test but against the null)

```
lm_red <- lm(protein ~ 1, dat_3genes) # null
lm_full <- lm(protein ~ gene + mrna, dat_3genes) # full
anova(lm_red,lm_full)
```

  - note: ANOVA can compare b/t any models, not just the null
  - small p–value = stat evidence that complex model provides better fit than the simpler model (explains more variability)
- t vs F test: t to assess the sig of individual coeff; F to assess the overall sig of model or compare nested models (full vs reduced)
  - note: equivalent when only 1 coeff is added btw full and reduced
  - ex. if comparing $y = \beta_0 + \beta_1$ vs $y = \beta_0$, then `tidy(full_model)`, `glance(full_model)`, `anova(full_model)` are all the same
- from middy: if asked to choose b/t `anova()`/`tidy()`, choose `anova()` if categorical var has > 2 levels, `tidy()` otherwise

Variable Model Selection

Generative Model

- may want to identify the most relevant var to build model
  - need to choose an evaluation metric, this depends on the goal
- forward selection: goal is to select the best model from among all possible models of varying sizes (`num_var`= $2^{\text{number of var}}$)
  - iteratively add var to the model starting from an intercept-only
  - proceeds to evaluate models of increasing size (start w models w 1 variable, then 2, and so on, until reaching the full model)
  - at each step/size, best model picked based on a metric like RSS
  - end: desired size reached or no improvement after add new var
- popular metrics for model selection: adjusted R^2, test mean squared error (MSE), C_p (AIC), or Bayesian information criterion (BIC)
  - C_p, AIC, BIC: smaller = better
- note: these selection algorithms are greedy and might not give globally best model (also might vary between runs)
- R code: for forward selection you can just do `method="forward"`

```
backward_sel <- regsubsets( # NOTE: code for both gen/pred
    x = response_var ~ explanatory_var,
    nvmax = max_size_of_model,
    data = test_set, # or training if generative
    method = "backwards" # or "forward", or "exhaustive"
)
reg_summary <- summary(backwards_sel) # lowest Cp = best model
cp_min = which.min(reg_summary$cp) # get model with min CP
selected_var <- names(coef(backwards_sel, cp_min))[-1] # names
```

Predictive Model

- models meant for inference is called generative models, models to predict is called predictive models – use different metrics
  - generative models: use R^2, adjusted R^2, MSE, F-test, etc
  - predictive model: use MSE, RMSE, R^2
    - for all these, can get metrics on the test set if you have one
    - can also do C_p, AIC, BIC on test set
    - note: can still use these metrics for predictive w/o test set
- test MSE: predict on test set, then calculate residuals/MSE
- test R^2: do $R^2 = corr(y_{\text{test}}, \hat{y}_{\text{test}})$ on the test set
  - interpretation: it's measure of $corr$ between true outcomes & model's predictions, no longer how much var model explains
- code: to get metrics on test set

```
test_metrics <- metrics(data, truth = target_col_name,
    estimate = prediction_col_name) # attach preds onto og DS
```

Uncertainty of Predictions

- predictions are random variables and thus have uncertainty
  - note: both CIP and PI are centered around the **same value**
- confidence intervals for prediction (CIP): range for the expected (average) val of Y for a given value of X (centered at fitted val $\hat{Y}_i$)
  - uses the regression line which introduces uncertainty (i.e $\hat{\beta}_i$)
  - code: `predict(lm_model, newdata, interval="confidence", level = 0.95, se.fit=TRUE)`
  - interpretation: with 95% confidence, the **expected** (average) value of data point with X fall between this range
- prediction interval (PI): range within which we can expect to find the actual observed value (like from og data) of Y for a given X
  - same uncertainty as the CIP plus uncertainty of ε_i so while it's also centered at $\hat{Y}_i$ but **is wider than CIP**
  - code: `predict(lm_model, newdata, interval="prediction")`
  - interpretation: with 95% conf, the Y val for data point with X val will be in this range (note: don't say expected anymore)

Predictive Modelling with LR

- split data into training & testing set (imagine data had ID col)

```
training_data <- sample_n(data,size=nrow(data)*0.7,replace=F)
training_data2 <- slice_sample(data, prop = 0.7) # another way
testing_data <- anti_join(data, training_data, by = "ID")
```

- build model using training data then run model on testing data

```
model <- lm(..., data = training_data)
predictions <- predict(model, newdata = testing_data)
RMSE <- rmse(preds = predictions,  # or can use metrics()
    actuals = testing_data$target) # get test RMSE of model
```

Inference After Model Selection

- if you use the same dataset to model select **and** make inference, you risk overfitting $\rightarrow$ <u>biased estimates and inflated Type I err</u>
  - Type I mean saying model not better than null (when it is)
  - you need to split - training for model selection (additionally using CV), then testing set to see performance
- usually there's trade-off between performance and interpretability
- overfitting: means model won't generalize well to unseen data

Regularized Regression Methods

- by using regularization: while training, they'll assign certain coeff weights of 0 (effectively removing - does both training & selecting)
- regularized models like Ridge and Lasso adds a penalty to loss function which shrink the coefficients towards 0
  - Ridge (L2 penalty) <u>doesn't set coeffs to 0</u>, but removes highly correlated variables (Lasso also works on high corr ds)
  - Lasso (L1 penalty) will set coeffs to 0, can use for model select
- regularization biases the coeffs by imposing penalty on their magnitude (trade-off: increase bias to decrease variance)
  - this means sampling dist doesn't center on true parameter val
- standardizing input var is necessary b/c methods sensitive to scale
- <u>bias correction</u> (post–Lasso): use Lasso to get the "chosen" coefficients, then re-estimate coeffs by running them through OLS
  - note: post-inf model may show non-sig terms - ignore and keep
  - **<u>cannot fully trust inference of re-fitted model</u>** (overfit)
- code: use `glmnet` to fit Ridge/Lasso and `cv.glmnet` for CV to find best λ (regularization strength)

```
cv_lambda <- cv.glmnet(
  x = X_train, y = y_train,
  alpha = 0/1, # 0 for Ridge, 1 for Lasso
  lambda = exp(seq(-5, 10, 0.1))) # range of lambda to try
lambda_min <- cv_lambda$lambda.min # lambda w/ min MSE
ridge_coef <- coef(cv_lambda, s = lambda_min) # get coeffs
ridge_min_predict <- predict(cv_lambda, newx = X_test,
    s = lambda_min) # predict on new data
ridge_rmse <- rmse(preds = ridge_min_redict, actuals = Y_test)
```

- `cv_lambda$lambda.1se` give λ bt 1 SE of min (sometimes better)
- note: use `newx` for reg-regression bc it requires matrix

- step-wise: is greedy (once var removed, not considered again) – only partial contribution considered, bad when num predictors $p > n$
- regularized methods: evaluate all variables jointly and removal not permenant, effective even when $n << p$

Cross-Validation

- hold out method: split into training and testing, create model on training, evaluate on testing set
  - limitation: estimate of test error rate depends on split (variable), tend to overestimate test error rate (fix with CV)
- k-fold CV: split data into k folds, leave 1 as validation set, train on $(k-1)$ folds, estimate test MSE on validation test, let every fold be validation set once, report RMSE/MSE as average of each run
- Leave-one-Out CV (LOOCV): N folds with 1 obs each, train on $N-1$ fold and test on left-out fold, do this N times, average (R)MSE at the end
  - resource intensive bc lots of training (good for small ds)
  - nearly unbiased param estimate but high variance

Logistic Regression

- used to: 1) estimate & test relations bt exp variables and **binary response**, 2) predict probability of a **binary** response (classifier)
- when response is binary (can only be 0 or 1), can't use SLR because it'll give predictions between 0 and 1, also outside of 0 and 1
  - binary response Y follows a Bernoulli distribution so $E(Y_i) = p_i$
- odds: prob positive over prob negative i.e odds $= p/(1-p)$
- the model: after R, you'll get output for $\beta_0, \beta_1, \ldots$, model is

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \ldots \beta_k X_k \tag{1}$$

$$\frac{p}{1-p} = e^{\beta_0} \times e^{\beta_1 X_1} \times \ldots \times e^{\beta_k X_k} \tag{2}$$

  - p is the prob of response being 1, $\log(p/1-p)$ is called **log odds**
  - (1) interpretation: for one unit increase in X_i (other constant), expect a β_i increase in **log odds** (similar interp to LR)
  - (2) interpration: if $e^{\beta_j} > 1$ – for one unit inc in X_j, the **odds** of the event occurring inc by factor of $e^{\hat{\beta}_j}$ (multiplicative)
  - note: when $e^{\hat{\beta}_j} < 1$, it's easier to interp $1/e^{\hat{\beta}_j}$ (flip target too)
  - ex. `exp_studentYes = 0.524`, then that means $1/0.524 = 1.908$, odds <u>non-default</u> of <u>non-student</u> is 90% more likely

```
log_reg <- glm(
    formula = y ~ x, data = data, family = binomial
) # formula like LR, family = binomial (logistic), or poisson
tidy(log_reg, exponentiate = FALSE) # = TRUE for exp version
```

- inference: same stuff as LR

$$H_0 : \beta_j = 0 \qquad H_a : \beta_j \neq 0 \qquad z_j = \hat{\beta}_j / \text{SE}(\hat{\beta}_j) \sim N(0,1)$$

  - $CI = \hat{\beta}_j \pm z_{\alpha/2}\text{SE}(\hat{\beta}_j)$ (use `tidy(model, conf.int = TRUE)`)
- prediction: plug X into eq (either ver) and solve for probability p

```
log_odds <- predict(model, newdata = new_data, type = "link")
odds <- exp(log_odds) # exponentiate to get odds
```

  - use `type = "response"` if you want straight probability
- overdispersion: IRL, data variance $\neq$ model's assumed variance
  - fix: estimate dispersion parameter ϕ to correct SE of estimators
  - code: change `family = "quasibinomial"` (same coeffs, diff SE)

Poisson Regression

- used when response is count (i.e # of crabs) – assume $Y_i \sim Pois(\lambda_i)$
- the model: after R, you get

$$\log(\lambda) = \beta_0 + \beta_1 X_1 + \ldots \beta_k X_k \tag{3}$$

$$\lambda = e^{\beta_0} \times e^{\beta_1 X_1} \times \ldots \times e^{\beta_k X_k} \tag{4}$$

  - λ is the expected number of occurrences ($\log(\lambda)$ is log of that)
  - (1) interpretation: inc in X_i by one unit (all else constant) results in a β_i increase in the log of expected counts
  - (2) interpretation: if $e^{\hat{\beta}_i} > 1$, for one unit increase in X_i, expected count of event by a factor of $e^{\hat{\beta}_i}$ (multiplicative)

Classification Metrics

- classification: log-reg give probability $\hat{p}_i$, we define a threshold p_0 where we'd predict 1 if $\hat{p}_i > p_0$, 0 otherwise (usually 0.5)
- error rate is your accuracy (# right / total)
  - training error rate likely underestimate out-of-sample error rate
  - use cross-validation to estimate out-of-sample error rate

```
cv_logistic <- cv.glm(glmfit = model, data = data,
    K = 10, cost = self_defined_cost_function)
```

- confusion matrix: show you type of errors made my model
  - True Positive (TP): # obs correctly predicted as 1
  - False Positive (FP): # obs incorrectly predicted as 1 (truly 0)
  - True Negative (TN): # obs correctly predicted as 0
  - False Negative (FN): # obs incorrectly predicted as 0 (truly 1)
  - (pred, actual): TP $= (1,1)$, FP $= (1,0)$, FN $= (0,1)$, TN$= (0,0)$
  - precision: PR $=$ TP$/($TP $+$ FP$)$
  - accuracy: ACC $= ($TP $+$ TN$)/n$

```
cm <- confusionMatrix(data=as.factor(model_preds),# col of 0/1
    reference = as.factor(data$target), positive = "1")
```

- sensitivity and specificity: define relative measures
  - sensitivity: estimated prob of predicting 1 given true class is 1
    - math: SN $=$ TP$/$TP $+$ FN
    - crucial in scenarios where missing a positive case could have serious consequences (i.e medical diagnosis)
  - specificity: estimated prob of predicting 0 given true class is 0
    - math: SP $=$ TN$/$TN $+$ FP
    - crucial when it's important to be correct when identifying negatives (i.e drug screening - don't want to falsely accuse)
  - when dec threshold, SN inc and SP dec
- AUC and ROC: sometimes we're not sure of threshold p_0 to choose
  - evaluate pred performance of model for all val of $p_0 \in [0,1]$
  - area under the curve (AUC) measures the classification ability of the model, ranges from 0 to 1, higher is better

```
ROC_output <- roc( # ROC TAKES PROBABILITY IN THE PREDICTOR
    response = data$target,
    predictor = predict(model, newdata, type = "response"))
plot(ROC_output, print.auc=TRUE) # actually plot
```

Regularized Logistic Regression

- can also use shrinkage methods in binary logistic regression
  - Regularized Loss $=$ Loss$(\beta) + \lambda||\beta||_i$
  - $||\cdot||$ is norm, use $||\cdot||_1$ for Lasso, $||\cdot||_2$ for Ridge
- note: `glmnet` takes variables as matrices
  - `object`: formula of your model (`-1` removes the intercept col)

```
X_train <- model.matrix(object = y~.-1, data=training_split)
y_train <- as.matrix(training_split$target, ncol = 1)
# do the same for testing split to get X_test, y_test
```

- CV with regularized logisitc regression

```
cv_lambda_ridge <- cv.glmnet(x = X_train, y = Y_train,
    alpha = 0,  # Ridge regression
    family = "binomial", # logistic regression
    type.measure = "auc", nfolds = 10)
ridge_max_AUC <- glmnet(x = X_test, y = Y_test, alpha = 0,
    family = "binmoial", lambda = lambda_min)
```

  - get λ from `cv_obj$lambda.min/1se` which gives max AUC score
  - sometimes, $\hat{\lambda}_{1\text{SE}}$ preferable bc we get much simpler model