Group 3: Modelling Dependent ~ Independent Variables

Ethan Walker, Drew Faturos, Maggie Weinroth, Kate Huebner

December 4, 2017

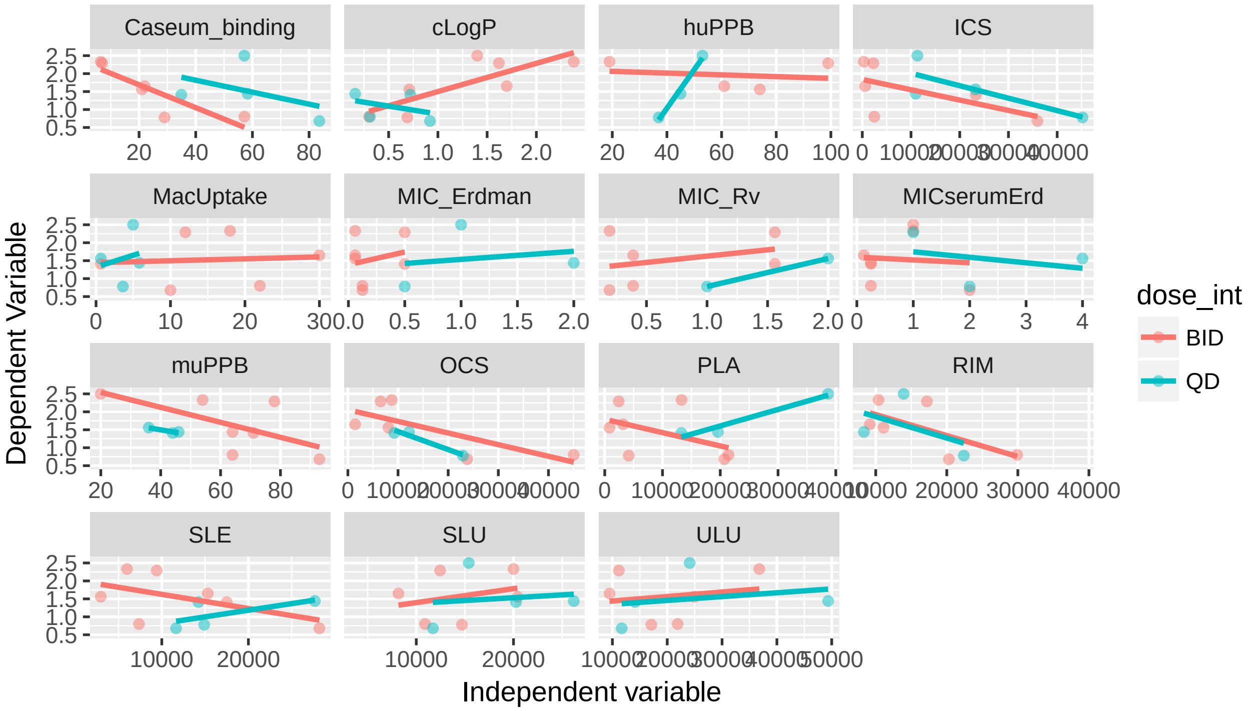
Introduction

The objectives of this analysis were to create summaries and visualizations of how the dependent variable is associated with the different independent variables. To this end, we developed different models to analyze these associations in the data. Our work included creating scatterplots, linear models, and supervised learning methods, including RandomForest, Regression Trees, and Lasso. The rationale for this is because if we can discover specific characteristics of the drugs that are associated with effectiveness against Tuberculosis (TB), this will help researchers understand possible drug mechanisms in the body using a mouse model. Different models were used to see if the significant associations were consistent between models.The dependent variables tested included drug concentrations in the lung tissue (`ELU`) and spleen (`ESP`). The independent variables included several *in vivo* (mouse model) and *in vitro* tests that were performed by the TB research group. We explored thesevarious models to see if there were associations detected between independent and dependent variables. It was challenging to make strong conclusions about associations based on this preliminary analysis because the dataset provided at this stage was small and did not include many drugs, however, we did discover similar patterns with certain *in vitro* or *in vivo* tests and ELU. We also explored changes in these associations by different drug doseage, and interestingly drug dose appears to be an effect modifier on some of the relationships, so that will be interest to follow up on as more data is provided.

Visualize independent dependent variable correlations in a scatterplot by drug dose

The goal of this function was to visualize how univariate measurement variables correlate with the dependent variables in a scatterplot object. A faceted scatterplot colored by dose was created to explore potential effect modification by dose. The function name was `univar_plot`. The input dataframe for this function, `efficacy summary`, was set as a default from the inputs team. Required user inputs included `peak_trough` (`Cmax` or `Trough`), which evaluates the peak or trough, respectively, of drug following injection. The `dep_var` options are either `ELU` (lung efficacy) or `ESP` (spleen efficacy). The function first filters the efficacy_summary dataframe rows to include either Cmax or Trough, depending on the user's specifications, then it uses the dplyr package `gather` function to create a new column called `independent_var` and each independent variable associated measurement in `independent_measure`. `Select` is then subsequently used to retain columns `drug`, `dosage`, `dose_int`, `level`, `dep_var`, `indep_measure`, and `independent_var`). The output of this function is a ggplot2 object that is faceted by independent variable, with the dependent variable (ELU or ESP) on the y axis, and colored by dose.

```r
#Sample code for function, univar_plot
univar_plot(peak_trough = "Cmax", dep_var = "ELU")
```



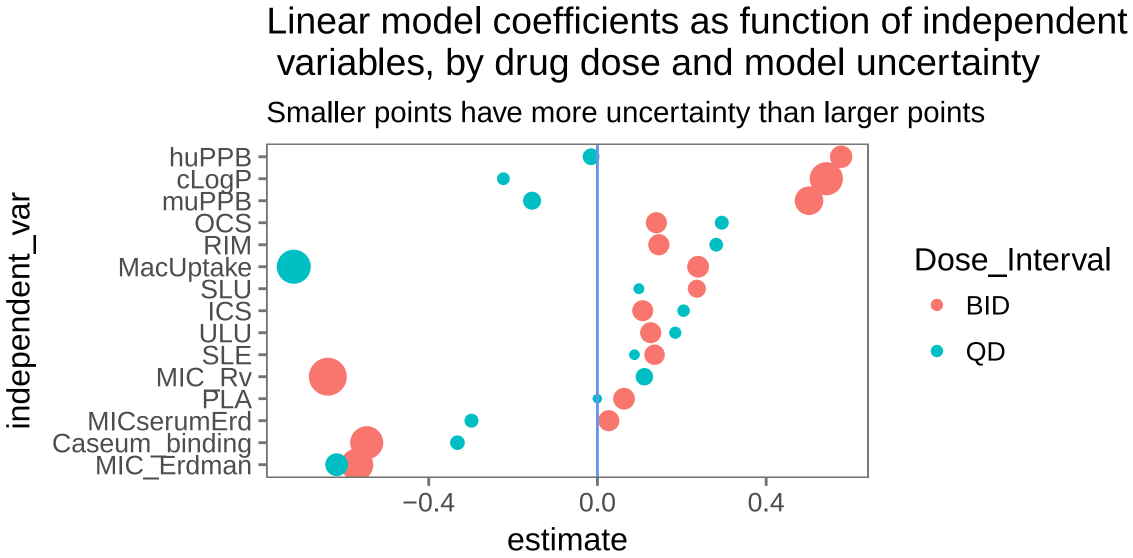
When interpreting the output from the preliminary data, there appears to be stronger linear relationships with certain variables, including `Caseum_binding`, `ICS` and `RIM`, compared to others. Some relationships are positively correlated, such as `QD` and `huPPB`, whereas others are negatively correlated, such as `Caseum_binding` and `ICS`. It appears that the relationship between independent and dependent variables are dependent on dose for variables such as `cLogP`, `huPPB`, `PLA`, and `SLE`. The others are not dependent on dose in this preliminar dataset. Therefore, dose may be an important factor to consider when evaluating the relationships between independent and dependent variable.

Fit a linear model for each dependent variable regressed on independent variables

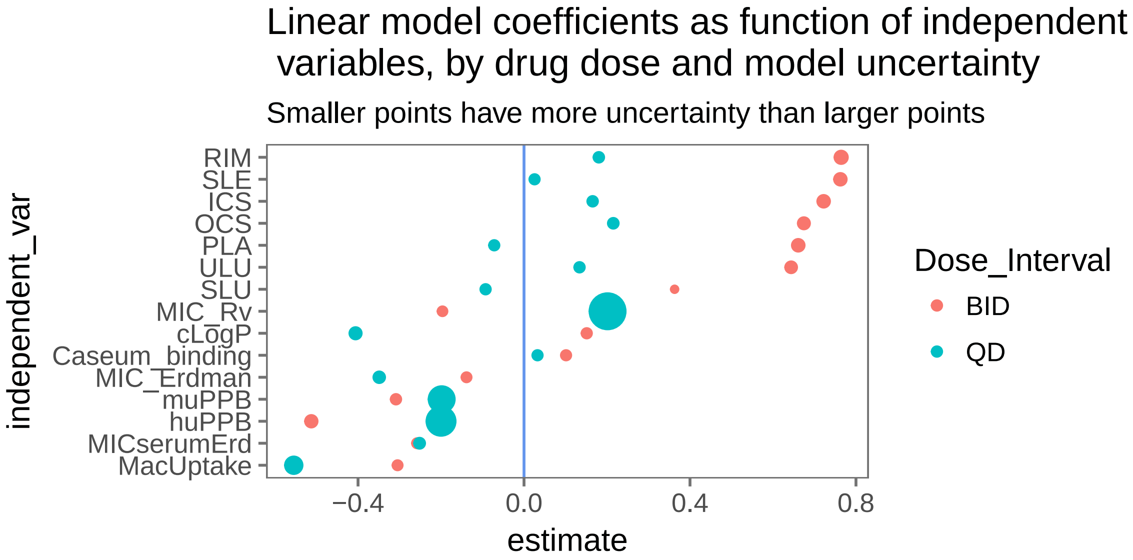
The goal of this function is to fit a linear regression model of dependent regressed on independent variables to assess the associations between them. Assumptions of this model are the observations are idependently and identically distributed, the error has a normal distribution, and there is a linear relationship between independent and dependent variables The function name is `linear_model`. The default input dataframe for this function is the `efficacy_summary` dataframe. Required user inputs to this function include `peak_trough` (options are Cmax or Trough), or the `dep_var` (options are ELU (lung efficacy) or ESP(spleen efficacy)). The output from this function provides a plot of the normalized model coefficients by each independent variable. The units of scale for each independent variable were normalized so that we could compare across coeficients. The model uncertainty (which is related to the sample size), is reflected by the size of the points, with smaller points having more uncertainty then larger points. This function utilizes the package, `dplr`, to filter the level specified in peak_trough to either `Cmax` or `trough`, then gathers and selects based on independent variable names and measurements, as described for the above function. A function within this function defines the `lm` function, which uses `scale` scales the independent variable columns of the matrix, then regresses the dependent variable (ELU or ESP) on the independent measure. Using `dplr` package `group_by` and `nest` functions to create one row per group, and the individual observations are stored in a column that is a list of data frames. This is a useful structure when you have lists of other objects (like models) with one element per group, see Hadley Wickam's post about this. We then use the `purrr` package to map the model function previously defined to the listed data, and then the `broom` package to create a tidy output of model coefficients.

We then unnest the dataframes and create a plot of the model estimates. In the plot, we utilize the package `forcats` to order the independent variable coeficients from low to high, and color by dose interval. the size of the points are plotted as the inverse of the standard error.

```
#Sample code for function, linear_model (Cmax and ELU)
linear_model(peak_trough = "Cmax", dep_var = "ELU")
```



```
#Sample code for function, linear_model (Cmax and ESP)
linear_model(peak_trough = "Cmax", dep_var = "ESP")
```

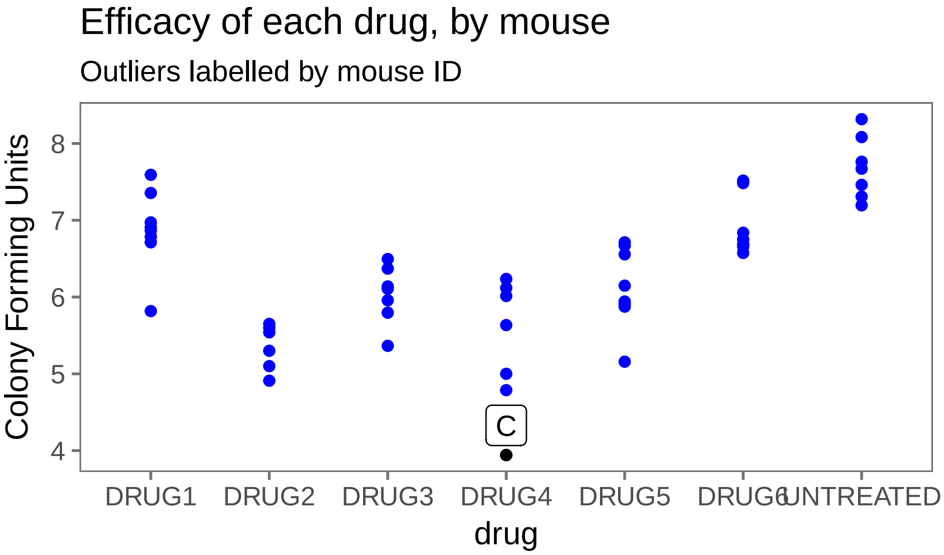


The coefficient plot generated by this function shows each independent variable on the y axis and the respective model coefficient on the x axis, for either ELU or ESP. If the coefficient is negative, for example, as it is with MacUptake in the ELU model linear regression model, an interpretation would be for every unit of change in the MacUptake, the ELU will decrease by 0.5 Units. Therefore, MacUptake has a negative relationship with ELU, decreasing the ELU. The diameter of the point represents the level of certainty of the coeficient in this model. This may change as more data is collected for each drug. Variables that had a strong positive association with ELU were huPPB (BID only), cLogP (BID only), and muPPB (BID only), and variables with a strong negative association with ELU were MacUptake(QD only), MIC_Rv (BID only), Caseum_binding (BID only), and MIC ERdman (both doseages. The variables that had a strong positive association with ESP were BID dose for RIM, SLE, ICS, OCS, PLAand ULU, but with low model certainty. The variables that had a strong negative association with ESP were MacUptake (QD) and huPPB (BID).

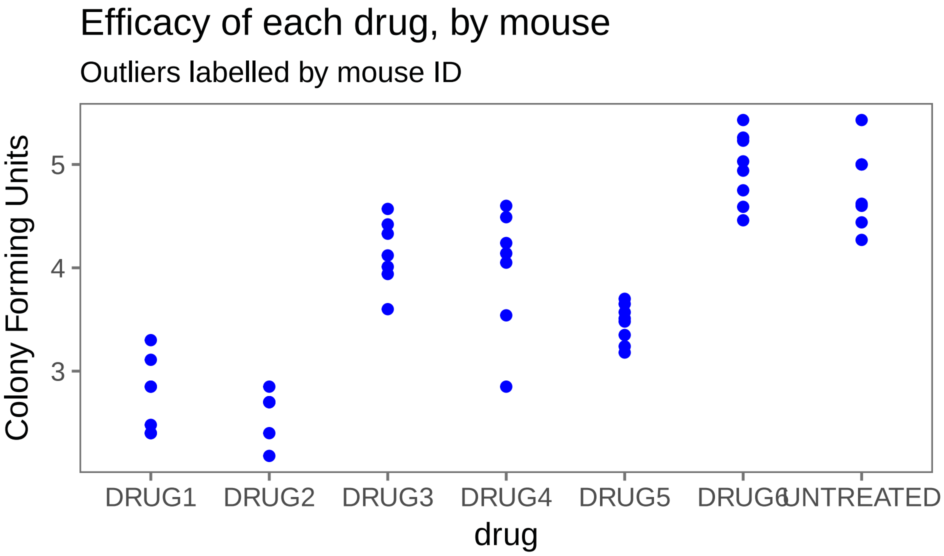
Visualizing individual drug efficacy by mouse

The goal of this function, which now is using individual drug data, is to visualize efficacy in the lung or spleen and detect mouse data outliers in the dataset. The function name is `drug_mouse`, and the default input dataframe is `cleaned_2_combined`. Required user inputs to this function are the `dep_var`, so lung_efficacy or spleen_efficacy. Output from this function provides a plot of the efficacy by individual mouse. If there are individual outliers (calculated using 1.5xIQR rule), it will be labelled with mouse ID.

```
#Sample code for function, drug_mouse
drug_mouse(dep_var = "lung_efficacy")
```



```
#Sample code for function, drug_mouse
drug_mouse(dep_var = "spleen_efficacy")
```



From this plot, it is clear that there is variation in the efficacy of each drug on lung efficacy or spleen efficacy using the mouse model. This plot shows the individual mouse variation in efficacy for each drug. Outliers (calculated using the 1.5 X IQR rule), are automatically labelled with the mouse ID. In this dataset for `lung_efficacy`, mouse C for drug 4 was an outlier. There were no outliers present in the `spleen_efficacy` dataset. You can see that untreated mice were had the least efficacy to decrease bacteria in the lung or spleen, respectively.

LASSO:

The *Least Absolute Shrinkage Selector Operator* or LASSO function is a penalized maximum likelihood model. While in past model selection was traditionally performed with step-wise, forward, or backward selection; penalized maximum likelihood models are the next step forward. This group of models uses a restricted fir that shrinks the parameters (either to smaller numbers or to zero depending on the method used). The three models commonly used are ridge, LASSO, and elastic net. Ridge regression shrinks coefficients that are not meaningful in the prediction but it leave these small number in a model. The issue with this is that you still are using the same number of predictors and not simplifying a complex model. As a result, even though that coefficients are smaller; the model is still very close to saturated.

```
This is where LASSO comes in. LASSO shrinks estimates down to absolute zero and eliminates them from th
```

For this model set, LASSO was chosen to simplify the model and there were not major concerns for predictors that needed to stay in the model that may have been dropped as a result of multicollinearity. The package we used for LASSO regression was `glmnet`. To prepare this data for input into the function; a few steps had to be taken. First, all missing data was removed via the `na.omit` call. From there, all predictors that were not numeric were removed via `select_if(is.numeric)`. From there, the original data had to be reformatted into a numeric list and a vector for the correct formatting of the function. Once the data was correctly formatted, the LASSO model was very straight forward to fit, `fit = glmnet(x, y)`. While there was an option to visualize the coefficients, the final function only outputs the point estimates. One parameter I had to set in the function was the `s` value in the `coef` call. The `s` value is the lambda parameter, which is pulled from the minimum lambda I observed while building the model. The Gaussian linear model was the default fit for this package; which was appropriate for these data.

Regression Tree Function:

The regression tree function is used as a way to helps us determine which independent variables have the largest influence on our outcomes of lung efficacy (ELU) or spleen efficacy (ESP). We begin with a tidy dataset of all observations (in this case, 20) called `efficacy_summary`. We then choose an independent variable of interest; here we have chosen ELU. To run the function, we input ELU along with all of the independent variables from the dataset into the `rpart()` function, as seen in the code below. In the function, we also specify the dataset from which to pull the variables, as well as control parameters that determine how the function runs.

Regression Tree code:

```
rpart(ELU ~  drug + dosage + level +
      plasma + `Uninvolved lung` + `Rim (of Lesion)` +
      `Outer Caseum` + `Inner Caseum` +
        `Standard Lung` + `Standard Lesion` + cLogP +
        `Human Plasma Binding` +
        `Mouse Plasma Binding` + `MIC Erdman Strain` +
      `MIC Erdman Strain with Serum` +
        `MIC rv strain` + `Caseum binding` +
        `Macrophage Uptake (Ratio)`
```

After trial and error, the regression tree function was completed and ready to use with the following parameters:

- `dep_var`: This is where the user specifies the dependent variable of interest. Options include "ELU" (lung efficacy) or "ESP" (spleen efficacy).
- `min_split`: This is a numeric input indicating the minimum number of observations for a split to be attempted in the regression tree.
- `min_bucket`: This is another numeric input indicating the minimum number of observations in a terminal node of the regression tree.
- `data = efficacy_summary`: This is the default dataset that the function uses. The function will only run with a dataset of this name that has these specific variable names.

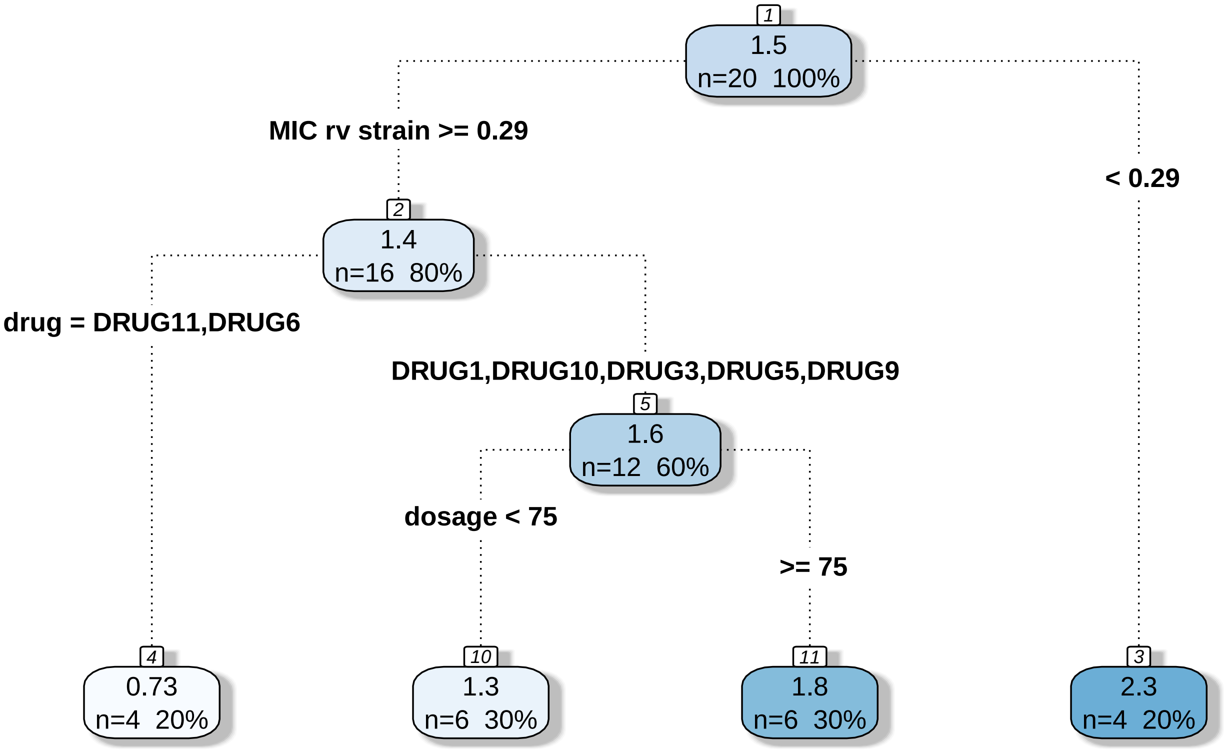
The following code gives an example of using the `regression_tree` function with the parameters that were explained above:

Regression Tree function input example:

```r
regression_tree(dep_var = "ELU", min_split = 8,
                min_bucket = 4)
```

Now that the code and function have been explained, we will run examples of the function and explain what the output is telling us. Consider the following plot given by the code printed above.

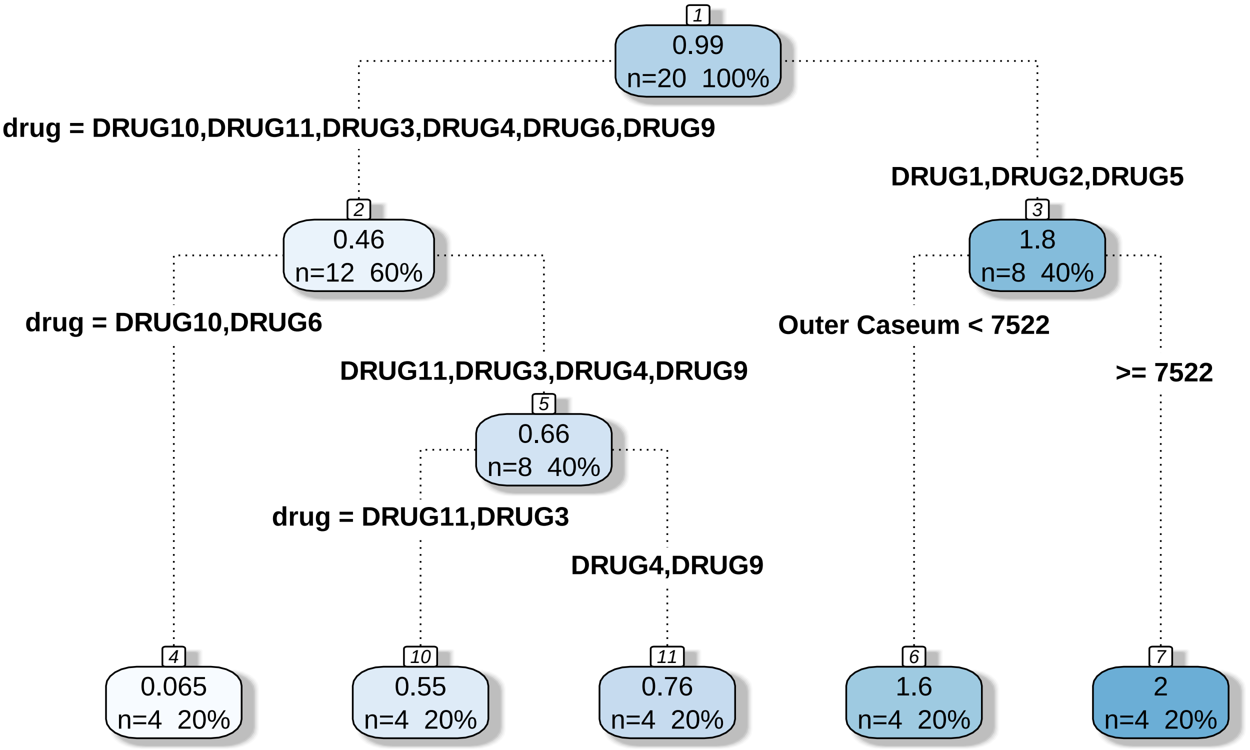
Regression Tree ELU Example:



Each "node" in the regression tree is numbered at the top. Within each node is a count of how many observations (for node 1, n=20) and the percent of total observations (for node 1, 100%) that are being explained by that node. The top number within each node is indicating the mean of the outcome variable for the observations in that node. In this case, ELU has a mean of 1.5 for node one.

Starting from node one, the first split is made so that it leads to the greatest possible reduction in residual sum of squares (RSS). At node one, the split was made at a level of 0.29 for variable `MIC rv strain`. This is saying that this variable at this level lead to the greatest reduction in RSS for the outcome of ELU. Continuing on, node three is a terminal node because it only has 4 observations. This happened because in our function parameters we set `min_bucket` = 4. Similarly, node two splits again because it has 16 observations. This happened because in our function parameters we set `min_split` = 8. This process continues until either the `min_split` or the `min_bucket` parameters are fulfilled for each node. The final regression tree above is showing us a step-by-step selection of the most important variables in determining ELU. Below is a similar regression tree for the outcome of ESP.

Regression Tree ESP Example:



Room for errors

These functions may be prone to several errors as new data is used as inputs. This includes possible errors if input datasets have low or high number of observations. In addition, errors could occur if missing data are recorded differently. We noticed in the individual drug data, the "NA", or missing data for spleen_efficacy had a space before the NA. This type of variation in how missing data is recorded could cause problems for the functions, although we tried to make them robust to these types of variations. Another potential problem could arise if drug names or independent variable column names change, as the functions are designed to handle the specified default dataframes. Most of the current functions in this project intake a "tidy" form of the dataframe, `efficacy summary`, with column names including: "drug", "dosage", "dose_int", "level", "PLA", "ULU", "RIM", "OCS", "ICS", "SLU", "SLE", "ELU", "ESP", "cLogP", "huPPB", "muPPB", "MIC_Erdman", "MICserumErd", "MIC_Rv", "Caseum_binding", and "MacUptake". Also, new independent variables or measurements are added to the dataframe could change the intput or output of the plot or model functions. Lastly, the dataset provided included two dose frequency combinations, 50 BID and 100 QD. If these dose and frequency combinations change, it could cause problems with some of the functions as they specifically look at those two dose-frequency combinations in the outputs.

Next steps

The next logical step for modelling dependent and independent variable associations will be to include more data representing more drugs. This preliminary analysis included 11 drugs, so more drug measurements and individual mouse data will make the models more robust. Next steps for the model development as new data are included are validating the models (for RandomForest and Lasso) through subsetting the data and training the model to determine predictive power for new datasets in the future. Model validation would additionally help us understand the predictive power of the model to determine drug efficacy. Ultimately, a nice feature would be if a new function could be written that runs all of the models and outputs the coefficients for each in one output so that they could be compared.