Logistics & Data Exploration

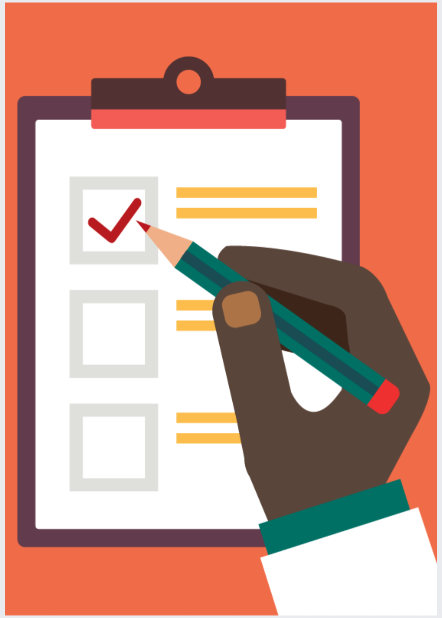
📋 🛠️ 📊

Applied Machine Learning in R
Pittsburgh Summer Methodology Series

Lecture 1-B July 19, 2021

Overview

Lecture Topics



Packages

- `caret` - primary package for this course
- `tidymodels`

Simple Data Split

- Training and testing datasets
- Data splitting in `caret`

Exploratory Data Analysis

- Data distributions
- Missing data
- Feature correlations
- Linearity and nonlinearity

Packages

How do we implement machine learning in R?

There are many packages for building and evaluating machine learning models in R.

Each implements specific ML models (e.g., `glmnet` for lasso and elastic net regularization, `rpart` for decision trees, `randomforest` for random forests).

These packages were built by different people over time, so syntax and conventions differ.

This can be confusing to remember!

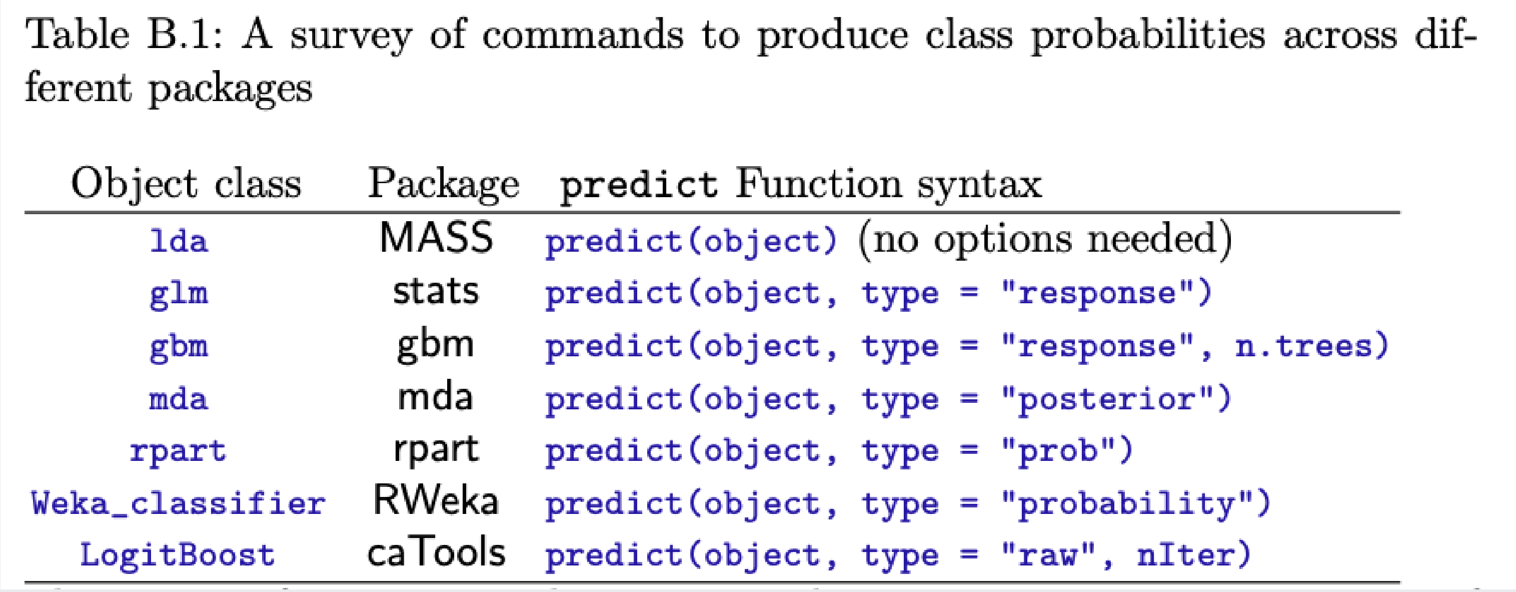


Table B.1: A survey of commands to produce class probabilities across different packages

Object class	Package	`predict` Function syntax
`lda`	MASS	`predict(object)` (no options needed)
`glm`	stats	`predict(object, type = "response")`
`gbm`	gbm	`predict(object, type = "response", n.trees)`
`mda`	mda	`predict(object, type = "posterior")`
`rpart`	rpart	`predict(object, type = "prob")`
`Weka_classifier`	RWeka	`predict(object, type = "probability")`
`LogitBoost`	caTools	`predict(object, type = "raw", nIter)`

`caret`

Recognizing the need for standardizing and streamlining the process of building and evaluating machine learning models, Max Kuhn and others developed the `caret` (**C**lassification **A**nd **RE**gression **T**raining) package.

This package allows researchers to quickly build and compare many different models.

There are 200+ machine learning models available in `caret`.

`caret` includes functions for:

- data visualization
- data pre-processing
- feature selection
- data splitting
- model training & testing
- variable importance estimation

caret

The `train()` function is the primary function for training models and tuning hyperparameters.

This same function and syntax is used to train any and all machine learning models.

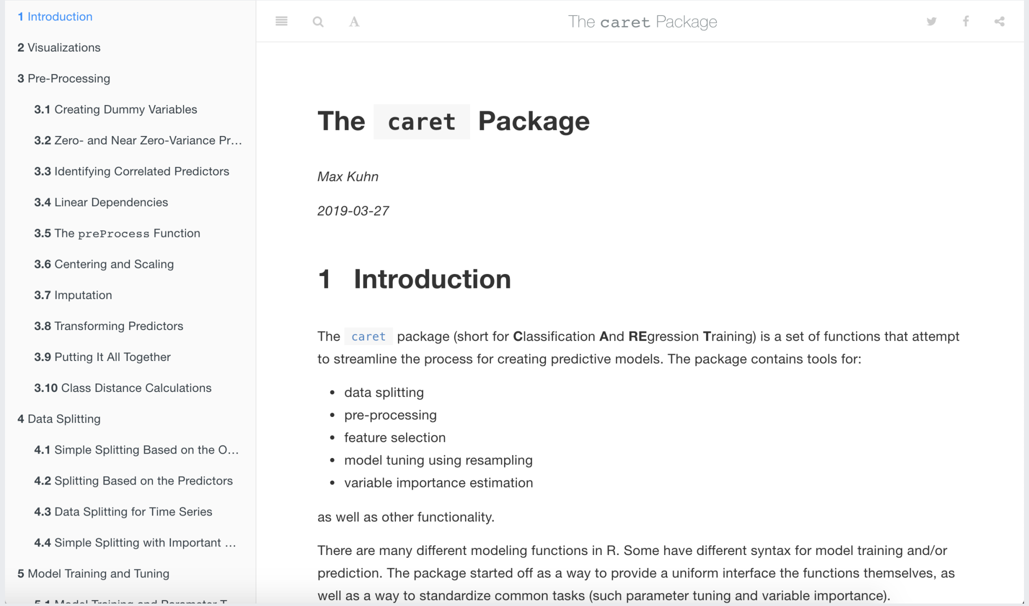
Users should also specify tuning parameter values and resampling method (e.g., *k*-fold cross-validation).

```
Define sets of model parameter values to evaluate
for each parameter set do
    for each resampling iteration do
        Hold–out specific samples
        [Optional] Pre–process the data
        Fit the model on the remainder
        Predict the hold–out samples
    end
    Calculate the average performance across hold–out predictions
end
Determine the optimal parameter set
Fit the final model to all the training data using the optimal parameter set
```

```r
iris_fit <- train(Species ~., data = iris,
                  method = 'glmnet',
                  trControl = trainControl(method = "cv", number = 10))
```

caret

Because `caret` has historically been the most popular package for machine learning in R, there are many freely available resources, solutions, and answers to questions online.



The caret Package

The caret Package

Max Kuhn

2019-03-27

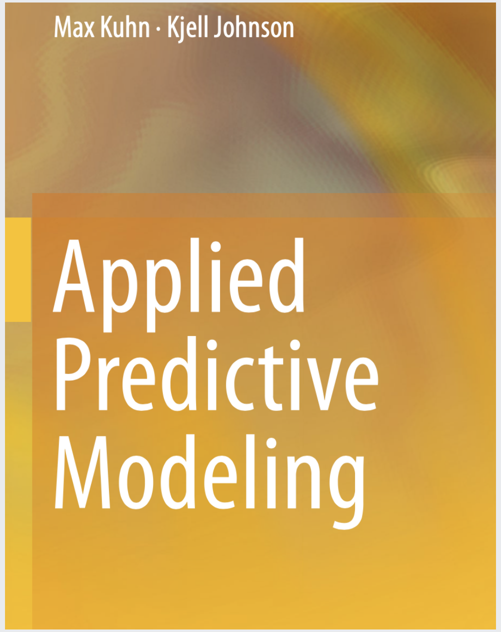
1 Introduction

The caret package (short for **C**lassification **A**nd **RE**gression **T**raining) is a set of functions that attempt to streamline the process for creating predictive models. The package contains tools for:

- data splitting
- pre-processing
- feature selection
- model tuning using resampling
- variable importance estimation

as well as other functionality.

There are many different modeling functions in R. Some have different syntax for model training and/or prediction. The package started off as a way to provide a uniform interface the functions themselves, as well as a way to standardize common tasks (such parameter tuning and variable importance).



tidymodels

The newer `tidymodels` package is the `tidyverse` version of `caret`. Both packages were developed by the same author (Max Kuhn)! `tidymodels` is a meta-package and includes a collection of many packages:

- `rsample` for data splitting and resampling
- `recipes` for pre-processing
- `parsnip` for trying out many models
- `workflows` to streamline the pre-processing, modeling, and post-processing
- `tune` to optimize model hyperparameters
- `yardstick` for model performance metrics
- `broom` for converting information to user-friendly formats
- `dials` for creating and managing tuning parameters

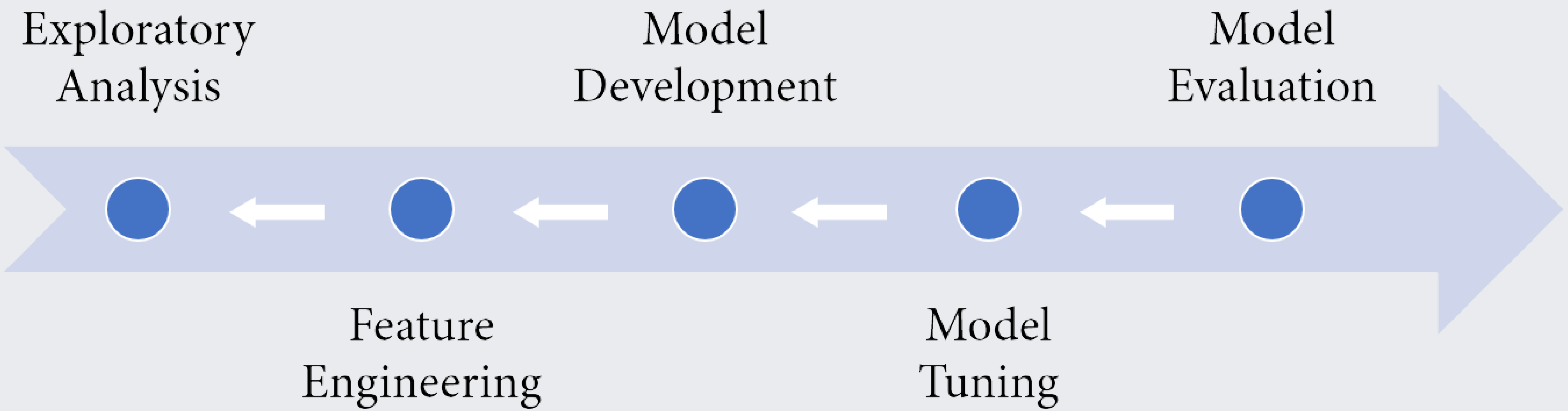
We will use the older `caret` but incorporate *aspects* of the newer `tidymodels`[1]

- This will give us access to some new features without overwhelming beginners
- It will also ease the transition to `tidymodels` if you decide to go that route

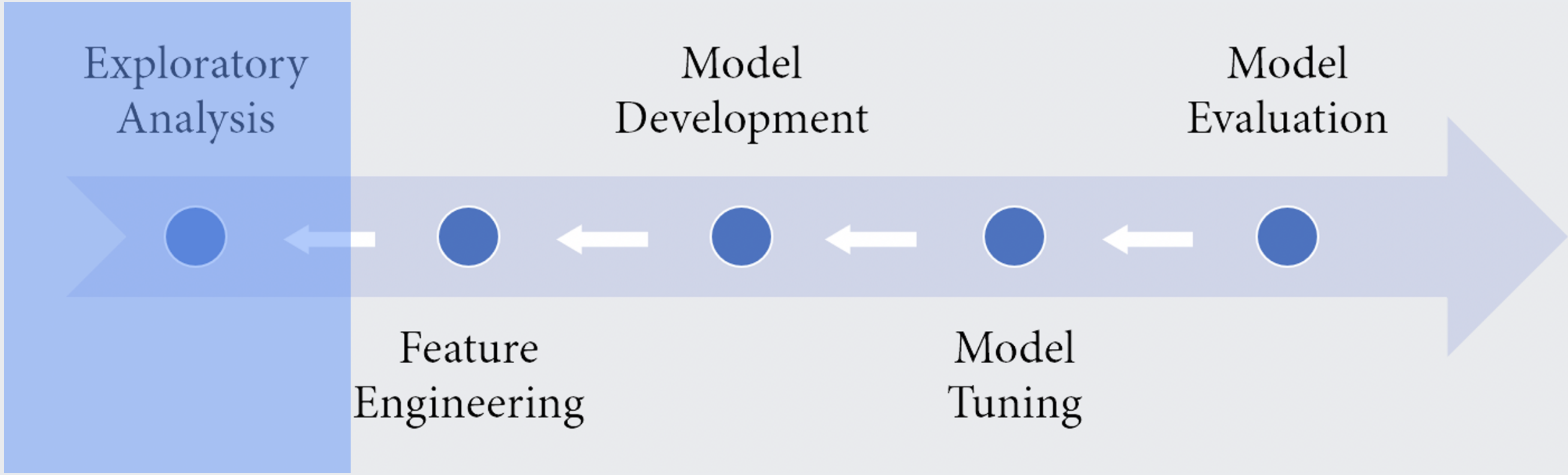
[1] We will use `recipes` and `yardstick` but not `workflows`, `rsample`, `tune`, `parsnip`, or `dials`.

Exploratory Data Analysis

Typical Workflow



Typical Workflow



Exploratory Data Analysis

Goals

- Develop an understanding of your data
- Make informed model building decisions (e.g., feature selection)

Questions

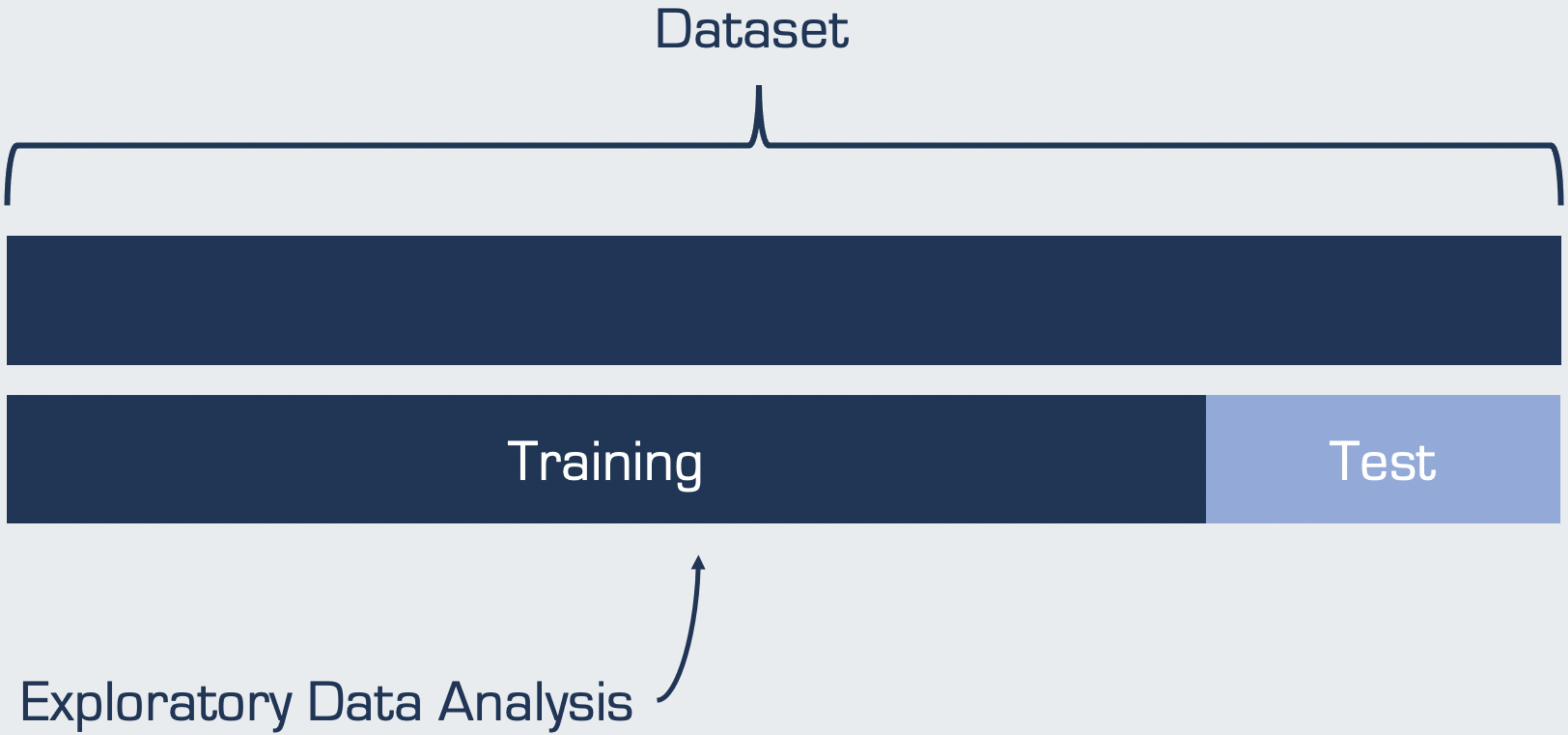
- What type of variation occurs in my variables?
- Are there any anomalies, errors, or outliers?
- How much missing data do I have?
- What type of covariation occurs between my variables?
- Are there any nonlinearities in my data?
- Are my data appropriate for the task?

WAIT!

ideally on training data *only*

Simple Holdout Set

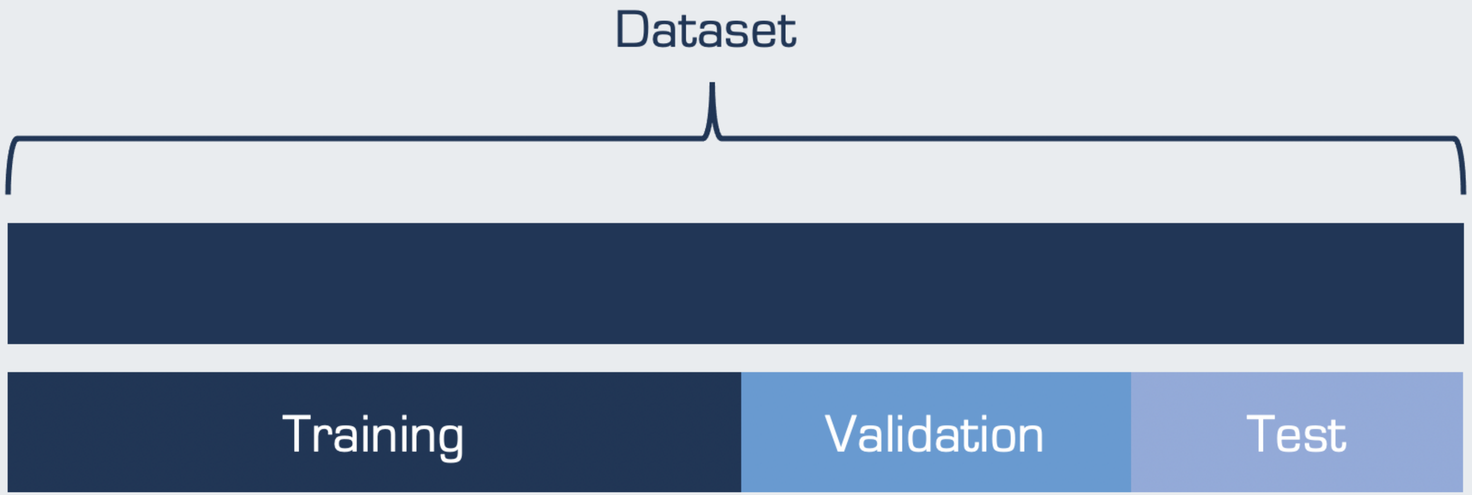
Simple Train/Test Split



Data Splitting

An important note on terminology beyond simple train/test data splits:

- **Training**: The data subsample used to explore the data and fit the model.
- **Validation**: Used for model evaluation while tuning hyperparameters; often implicitly split via cross-validation.
- **Test**: Entirely held-out from model training/tuning; used to provide a unbiased evaluation of the final model.



Simple Train/Test Split

Use the `caret::createDataPartition()` function to create balanced training and testing splits based on the outcome variable. Random sampling occurs within each factor level to maintain class distribution in the datasets.

Specify the proportion of data you want in the training split (e.g., `p = 0.8`) for an 80%/20% data split.

Remember to set a seed so your results are reproducible!

```
library(caret)
set.seed(2021)
trainIndex <- createDataPartition(iris$Species, p = .8,
                                  list = FALSE,
                                  times = 1)
```

Simple Train/Test Split

Use the `createDataPartition` row indices to split your data into single train and test sets.

```
irisTrain <- iris[trainIndex, ]
irisTest <- iris[-trainIndex, ]
```

Now 80% of the data is designated for model training and can be used for exploratory data analysis. 20% of the data is held out and should not be explored before testing the model, to avoid overly optimistic results.

```
dim(irisTrain)
```

```
## [1] 120   5
```

```
dim(irisTest)
```

```
## [1] 30  5
```

Why EDA on training data only?

The **ultimate goal** of exploratory data analysis is to gain insights into data to make informed modeling decisions.

We split our data into training and testing subsets to evaluate the accuracy of our model in predicting *unseen* data.

This gives us a sense for how our model might perform in the **future** on new datasets.

If modeling decisions are made based on data patterns we observe in the test set, we risk artificially inflating model performance estimates in the test set.

When possible, it is ideal to perform exploratory data analysis only on your training data only.

But note that doesn't mean we can't check the test data for coding errors or data anamolies!

Knowledge check

Taylor is interested in building a machine learning model to predict future risk of depression. How should they explore these elements of their dataset?

Question 1

Looking for outliers or data anomolies:

a) Training data only

b) Test data only

c) Both training and test

d) Neither

Question 2

Finding features that correlate with the outcome:

a) Training data only

b) Test data only

c) Both training and test

d) Neither

Exploratory Data Analysis

Data Distributions and Error Detection

Data distributions and error detection

Typically, the first step in exploratory data analysis is to explore data distributions.

This provides insight into:

- Whether features are normally distributed
- Concerning or extreme skewness
- Potential data anomalies or errors
- Data outliers
- Features with low variance
- Imbalanced data (categorical variables)

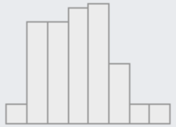
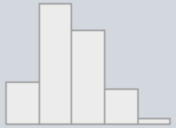
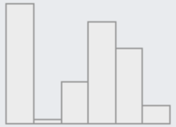
Methods:

- Summary statistics
- Histograms
- Bar charts

Data distributions and error detection in R

The `dfsummary()` function from `summarytools` is useful for quickly identifying trends and anomalies at a glance.

```
print(dfSummary(irisTrain), method = 'render')
```

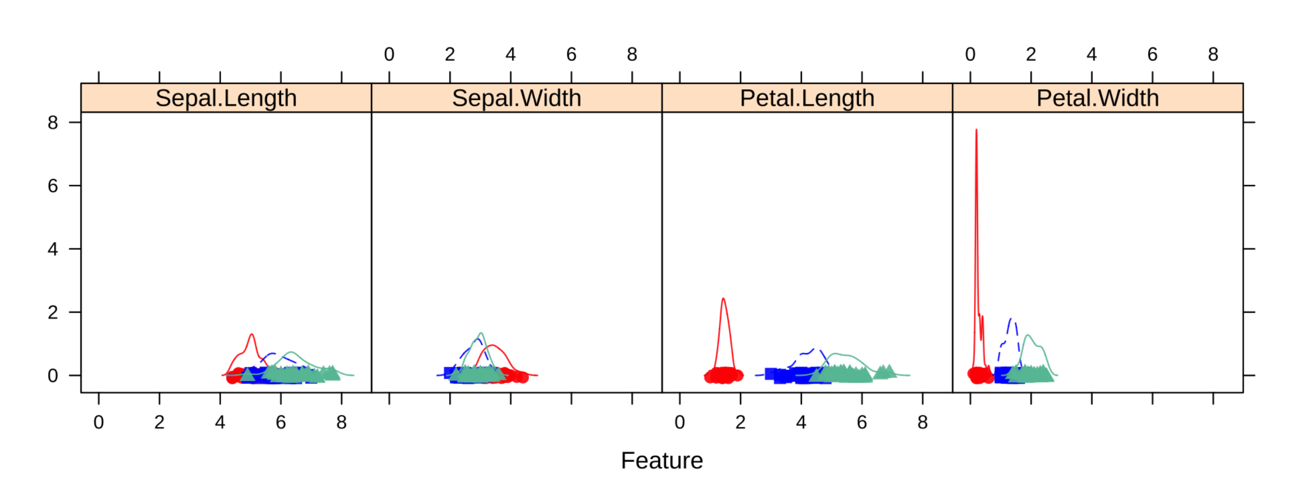
No	Variable	Stats / Values	Freqs (% of Valid)	Graph	Valid	Missing
1	Sepal.Length [numeric]	Mean (sd) : 5.8 (0.8) min < med < max: 4.4 < 5.8 < 7.7 IQR (CV) : 1.3 (0.1)	32 distinct values		120 (100.0%)	0 (0.0%)
2	Sepal.Width [numeric]	Mean (sd) : 3 (0.4) min < med < max: 2 < 3 < 4.4 IQR (CV) : 0.5 (0.1)	22 distinct values		120 (100.0%)	0 (0.0%)
3	Petal.Length [numeric]	Mean (sd) : 3.7 (1.7) min < med < max: 1 < 4.3 < 6.9 IQR (CV) : 3.5 (0.5)	39 distinct values		120 (100.0%)	0 (0.0%)

Generated by summarytools 0.9.9 (R version 4.1.0)
2021-07-13

Data distributions and error detection in R

Overlaying distributions on the same plot can also be helpful. We can use the `featurePlot()` function in `caret`.

```
# basic density plot
featurePlot(x = irisTrain[, 1:4], y = irisTrain$Species, plot = 'density')
```

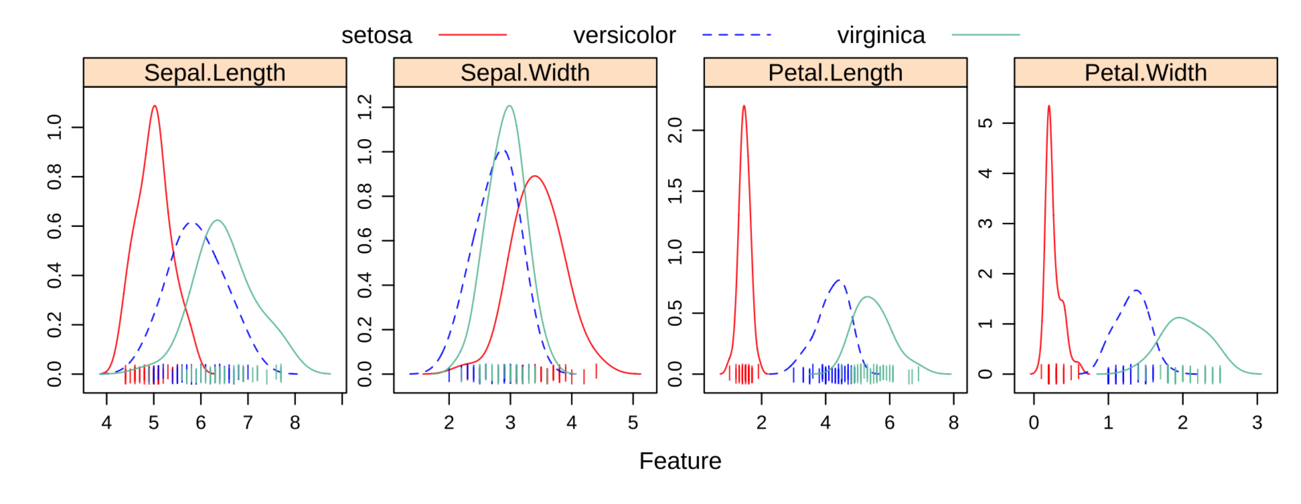


credit to https://topepo.github.io/caret/visualizations.html

Data distributions and error detection in R

Overlaying distributions on the same plot can also be helpful. We can use the `featurePlot()` function in `caret`.

```
# make it prettier: free x & y axis, add legend, change plot character
featurePlot(x = irisTrain[, 1:4], y = irisTrain$Species, plot = 'density',
            scales = list(x = list(relation = "free"), y = list(relation = "free")),
            adjust = 1.5, pch = "|", auto.key = list(columns = 3))
```

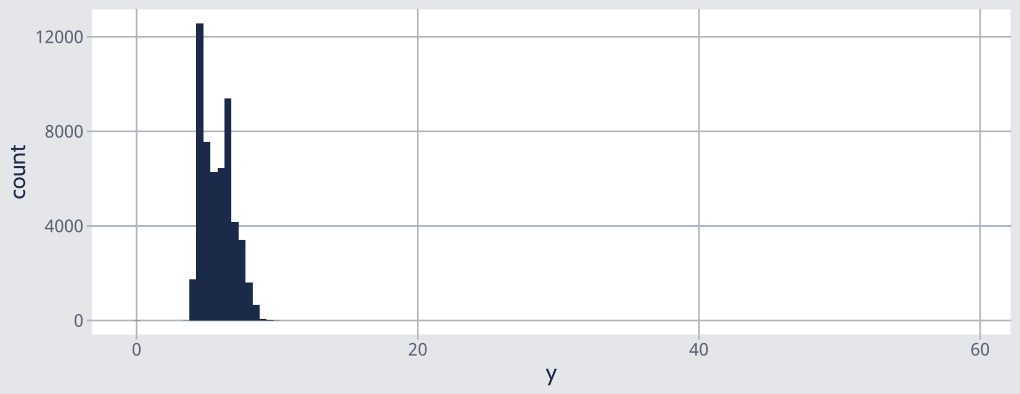


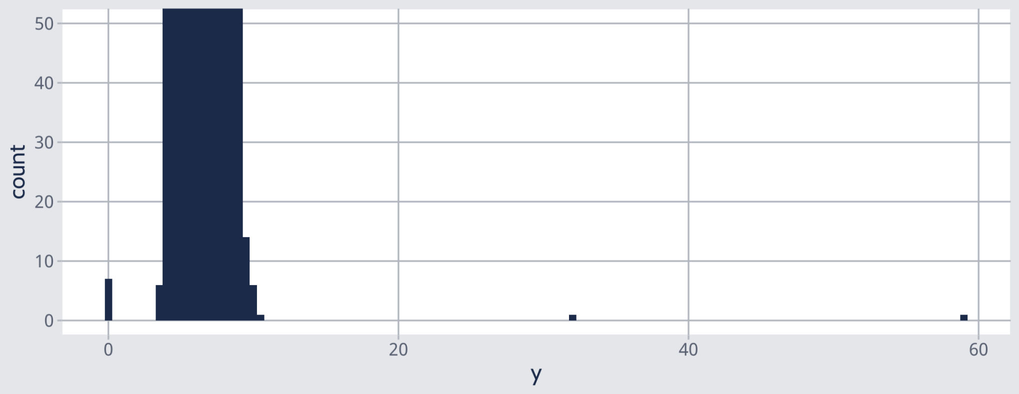
credit to https://topepo.github.io/caret/visualizations.html

Are there data anomalies, errors, or outliers?

Check data distributions and summary statistics for:

- Extreme values
- Nonsensical values
- Inconsistencies
- Low variance
- You may need to adjust plot margins or axes!





Exploratory Data Analysis

Missing Data

A technical explanation of missing data

Missing completely at random (MCAR)

- No systematic pattern of missing data; the probability of an observation being missing does not depend on any observed or missing data values.
- E.g., If a weighing scale sometimes runs out of batter, missing data on weight is only due to bad luck and not any measured or missing data.

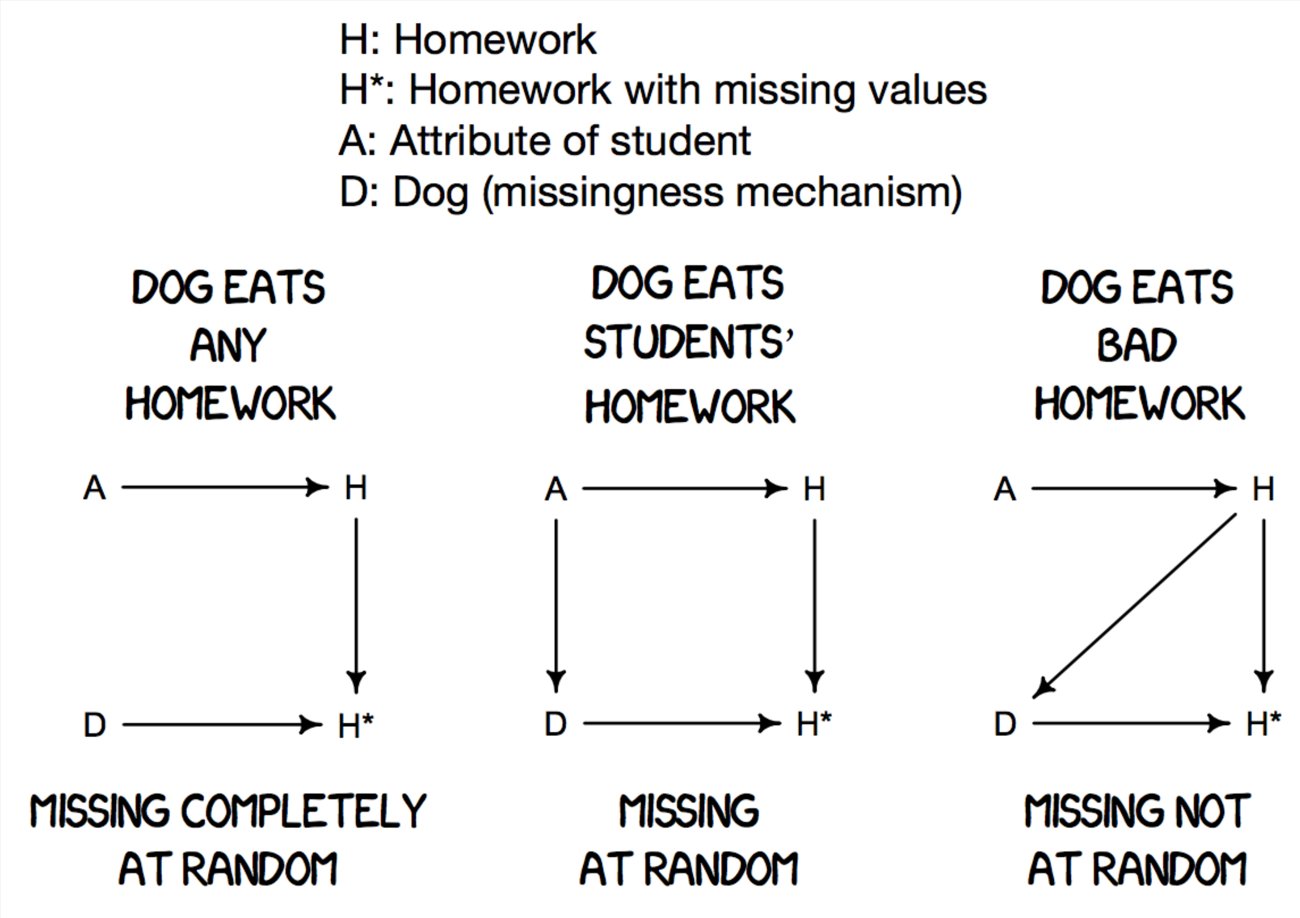
Missing at random (MAR)

- Systematic relationship between missing values and the *observed* data, but *not* the missing data.
- E.g., If people with eating disorders are more likely to decline being weighed, missing data on weight is systematically related to eating disorder diagnosis.

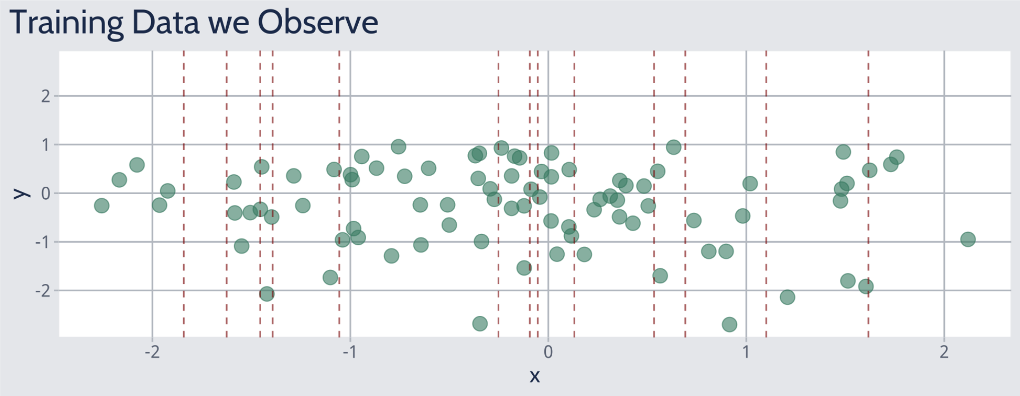
Missing not at random (MNAR)

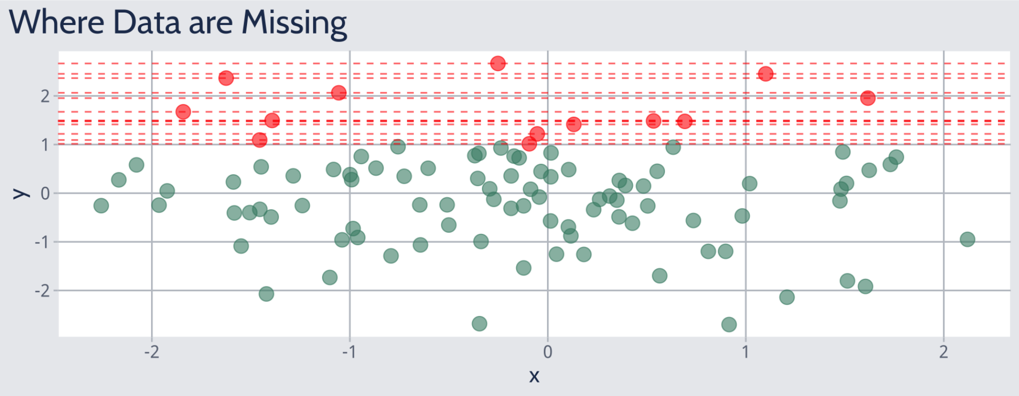
- Systematic relationship between missing values and those values themselves.
- E.g., If people with higher weights are more likely to decline being weighed, missing data on weight is systematically related to *weight itself*.

An intuitive explanation of missing data



Missing Data





The `VIM` package is particularly helpful for visualizing patterns of missing data.

Two helpful questions to guide missing data visualization:

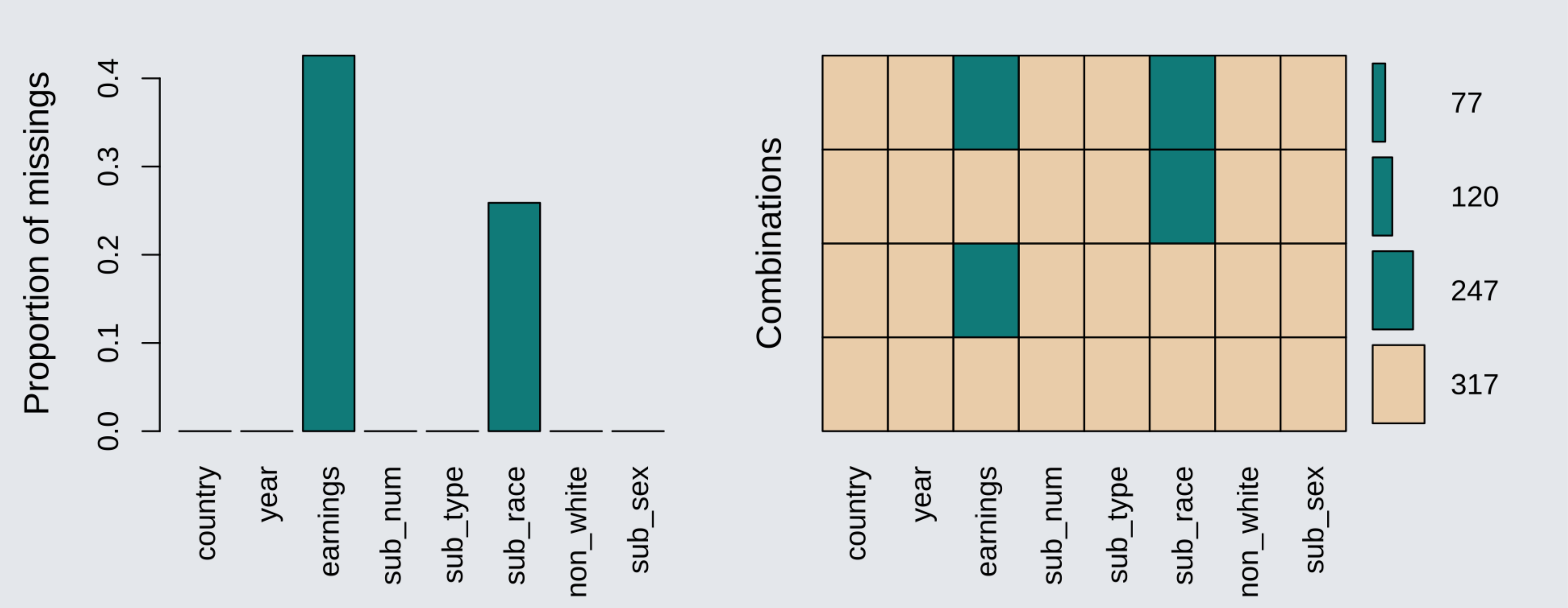
- Which variables have missing observations (and how many)?
- Does missing data in one variable depend on other variables?

credit to https://www.datacamp.com/community/tutorials/visualize-data-vim-package

Missing data visualization in R

Aggregation plots are useful for inspecting the prevalence of missing data.

```
aggr(biopics, numbers = TRUE, prop = c(TRUE, FALSE), col = c("bisque2","darkcyan"))
```

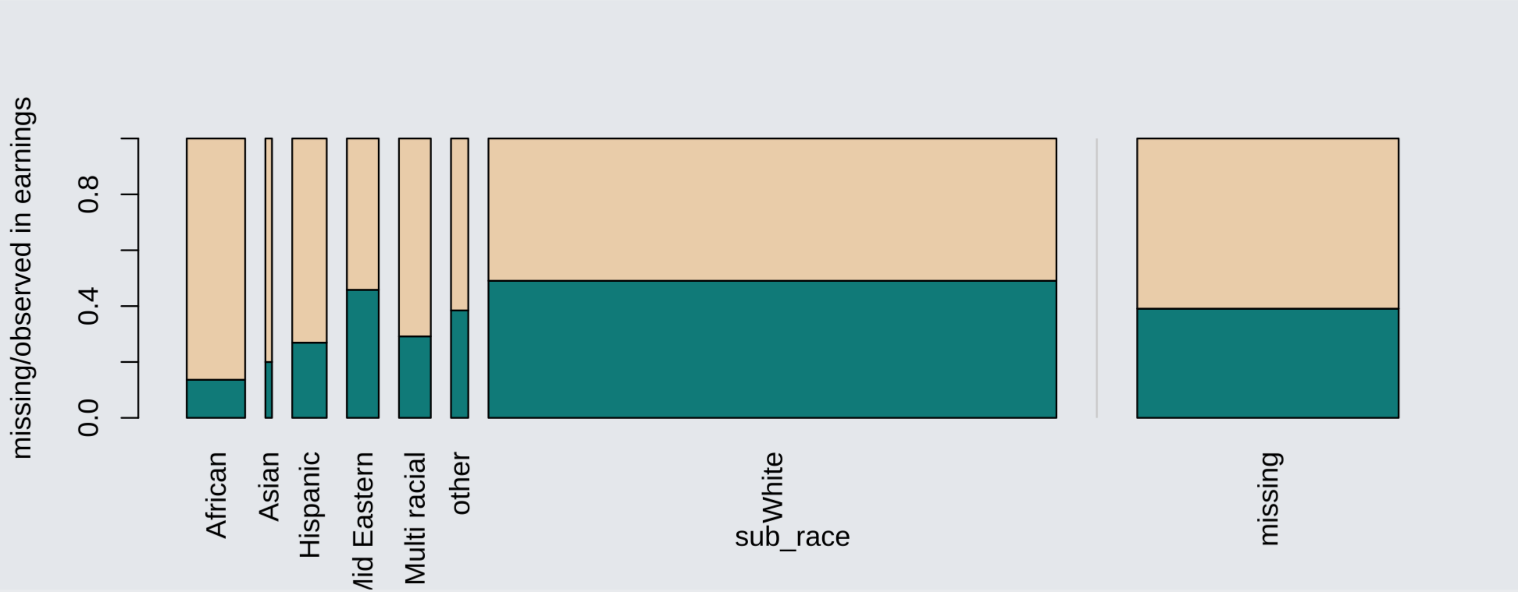


credit to https://www.datacamp.com/community/tutorials/visualize-data-vim-package

Missing data visualization in R

We also want to know if missing data systematically vary by other observed data. If the other data are numeric we use a **spinogram**; if categorical we can use a **spineplot**.

```
spineMiss(biopics[, c("sub_race", "earnings")], col = c("bisque2","darkcyan"))
```



credit to https://www.datacamp.com/community/tutorials/visualize-data-vim-package

Missing data visualization in R

Let's flip the two variables to ask: does the percentage of missing data in `sub_race` differ by `earnings`?

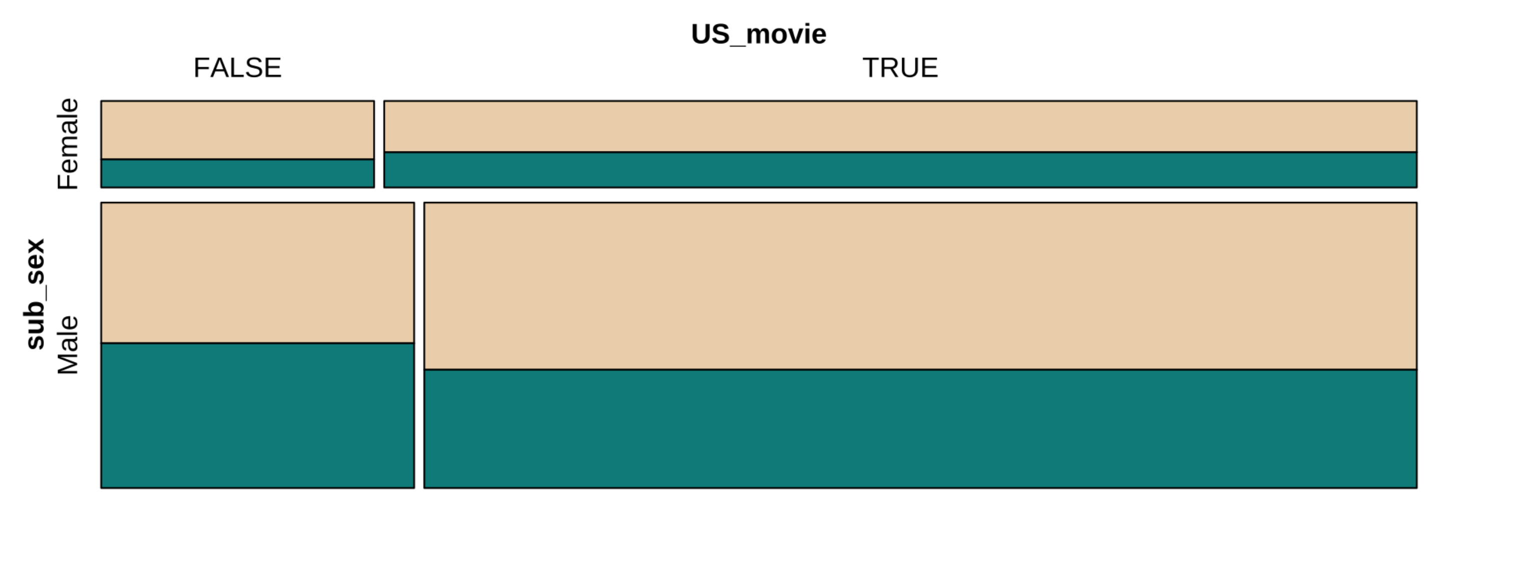
```
spineMiss(biopics[, c("earnings", "sub_race")], col = c("bisque2","darkcyan"))
```

credit to https://www.datacamp.com/community/tutorials/visualize-data-vim-package

Missing data visualization in R

Mosiac plots generalize spineplots and spinograms (which only plot two variables at a time) to multiple variables.

```
mosaicMiss(biopics[, c("sub_sex", "US_movie", "earnings")], highlight = 3,
           plotvars = 1:2, miss.labels = FALSE, col = c("bisque2","darkcyan"))
```

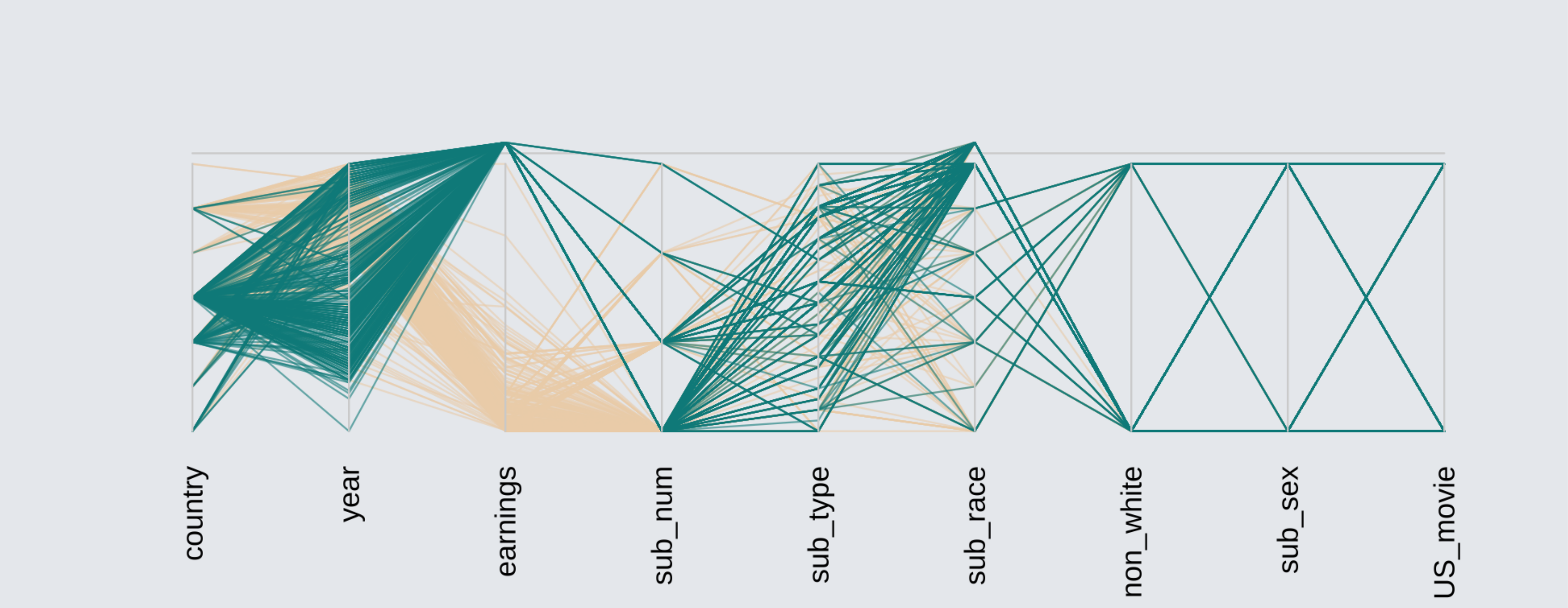


credit to https://www.datacamp.com/community/tutorials/visualize-data-vim-package

Missing data visualization in R

Parallel coordinate plots allow us to look at patterns of missingness across the entire dataset.

```
parcoordMiss(biopics, highlight = 'earnings', alpha = 0.6, col = c("bisque2","darkcyan"))
```



credit to https://www.datacamp.com/community/tutorials/visualize-data-vim-package

Exploratory Data Analysis

Feature Covariation and Correlations

Feature covariation and correlations

In addition to asking what type of variation occurs *within* features, we should also explore the covariation that occurs *between* features (as well as covariation between features and outcome variables).

This provides insight into:

- Highly correlated features (multicollinearity)
- Potential clusters of features that could be reduced into a single feature
- Features with strong relationships to the outcome (feature selection)

Methods:

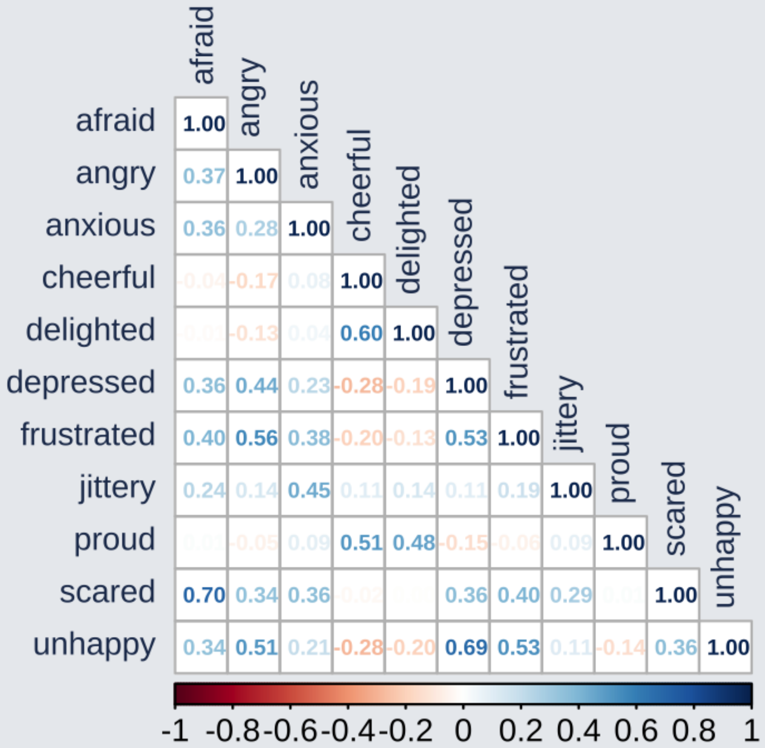
- Correlation matrices
- Correlation matrices with clustering
- Scatterplot matrices

Feature covariation and correlations in R

```
data(msq)
cormat <- cor(subset(msq, select = c("afraid", "angry", "anxious", "cheerful", "delighted", "depress
corrplot(cormat, tl.col = '#23395b', type = 'lower', tl.cex = 0.8)
```

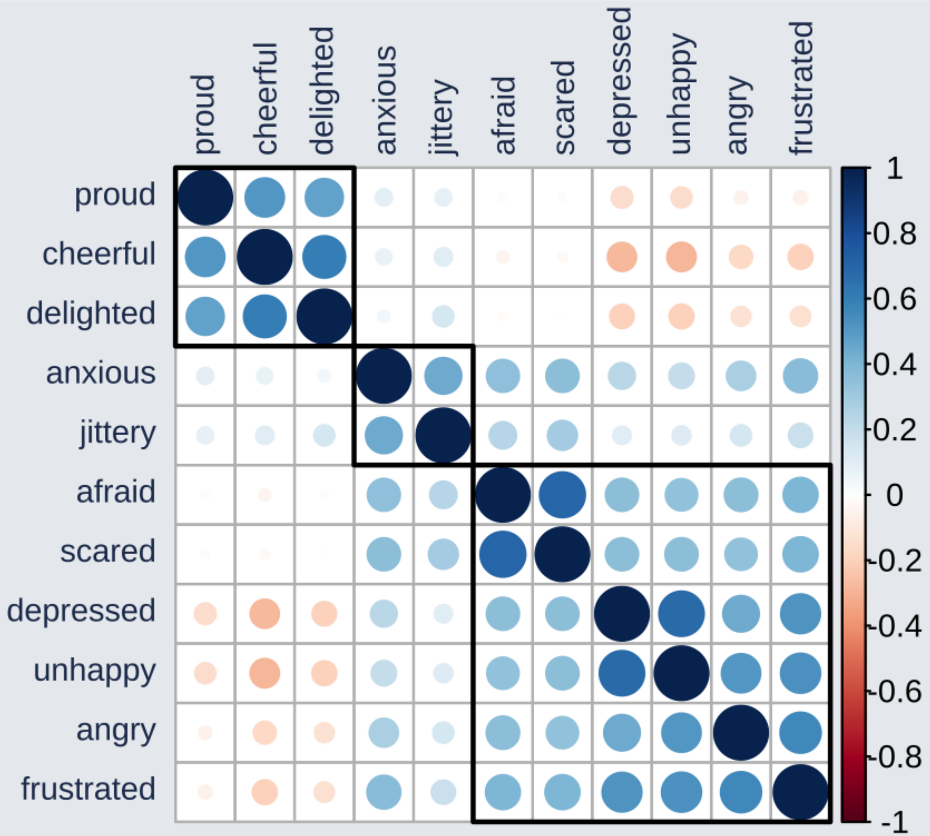
Feature covariation and correlations in R

```r
# Correlation matrix with numbers
corrplot(cormat, type = 'lower', method = 'number',
         tl.col = '#23395b', tl.cex = 0.8, pch.cex = 0.5, number.cex = 0.55)
```



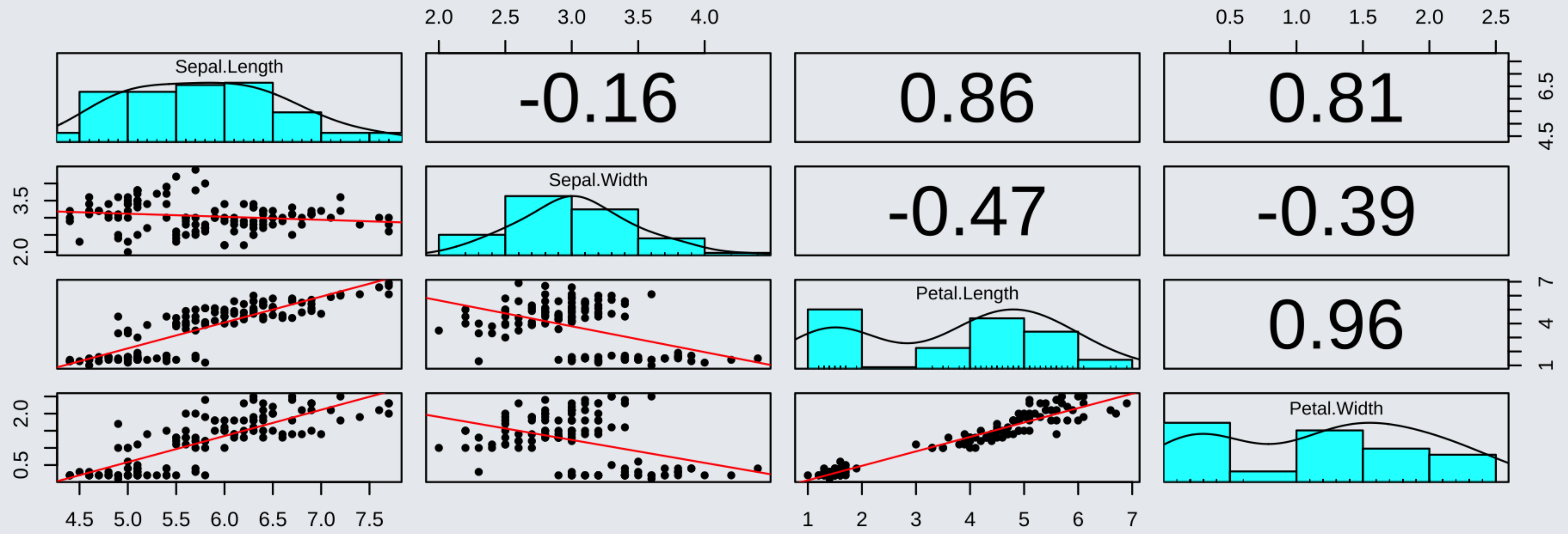
Feature covariation and correlations in R

```r
# Correlation matrix with hierarchical clustering
corrplot(cormat, tl.col = '#23395b', order = 'hclust', addrect = 3,
         tl.cex = 0.8, pch.cex = 0.5, number.cex = 0.55)
```



Feature covariation and correlations in R

```
# Scatterplot matrix
pairs.panels(irisTrain[, 1:4], method = 'pearson', density = TRUE, ellipses = FALSE,
             lm = TRUE, cex.cor = 0.8, cex.labels = 0.9)
```



Exploratory Data Analysis

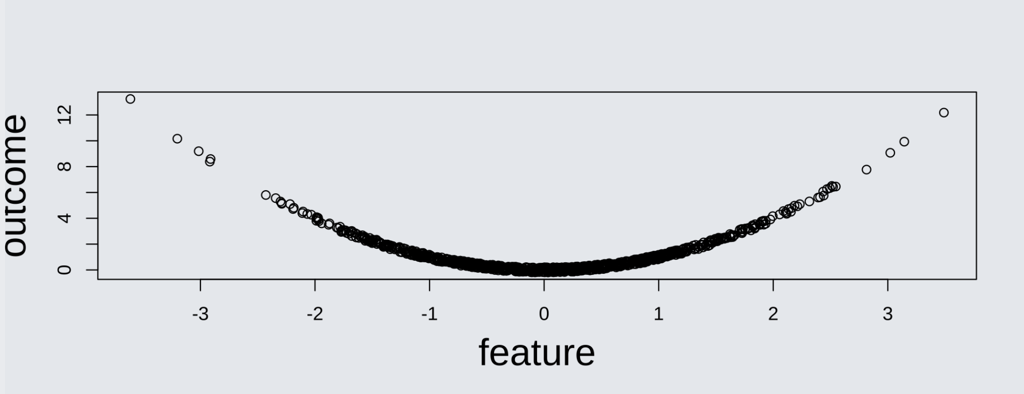
Linearity and Nonlinearity

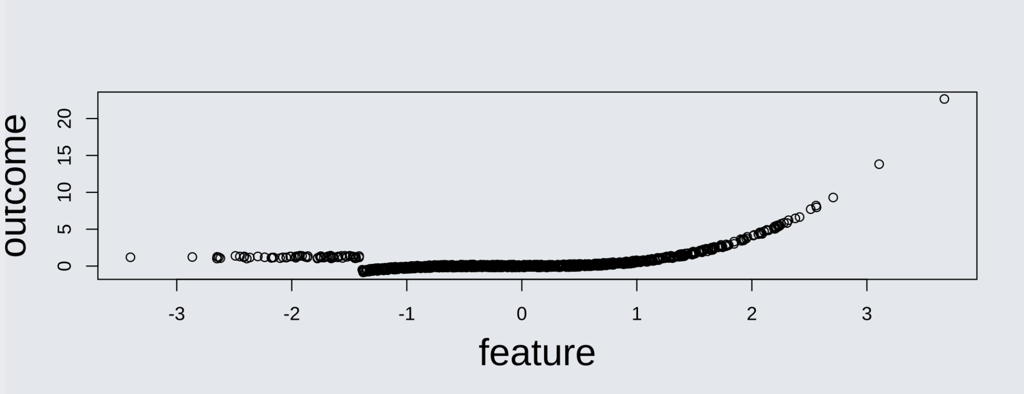
Linearity and nonlinearity

Nonlinearity between features and outcome variables are important to pay attention to, because these data patterns inform algorithm selection.

While some algorithms (e.g., decision trees, random forests) can capture and model nonlinearity, other algorithms (e.g., lasso, ridge, elastic net) cannot.

The specific **form** of nonlinearity is also important.

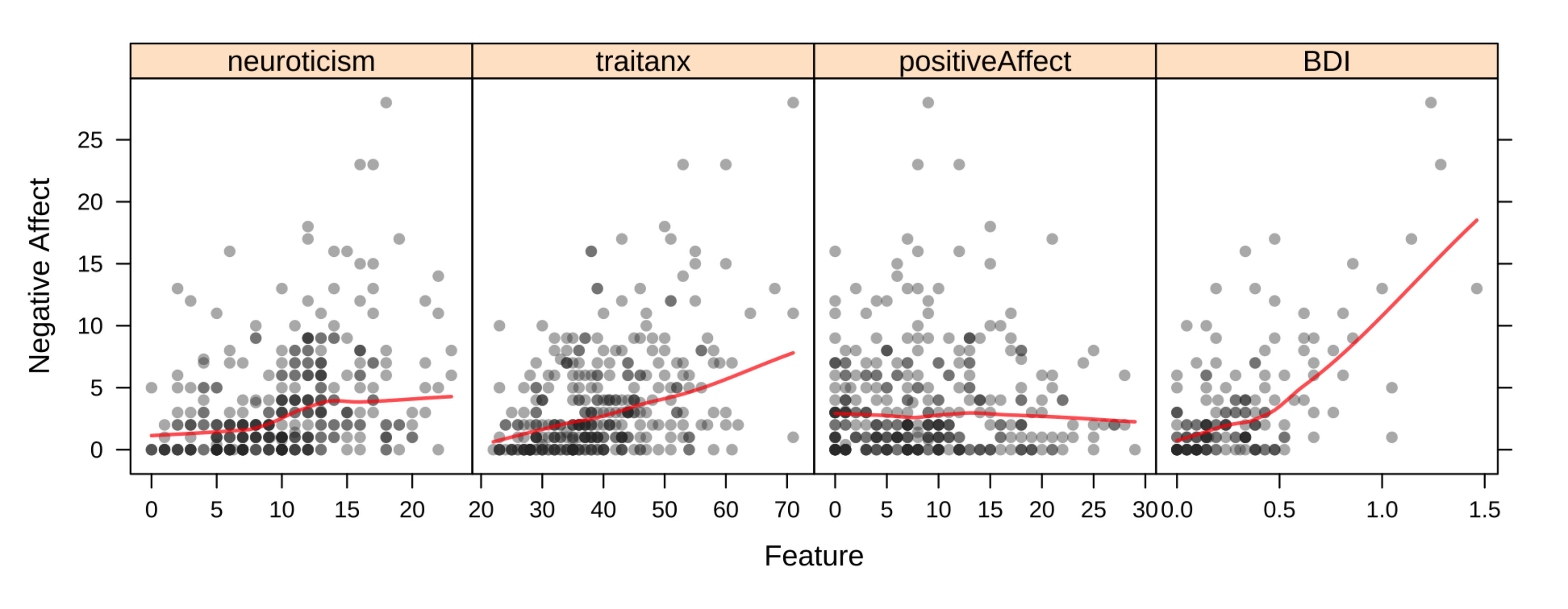




Linearity and nonlinearity in R

We can use the `featurePlot()` function in `caret` to look for nonlinearities between features and outcomes.

```
featurePlot(x = affect[, c(4, 8, 12, 20)], y = affect$NA1, plot = "scatter",
            type = c("p", "smooth"), labels = c("Feature", "Negative Affect"))
```



Exploratory Data Analysis

Takeaways

Using EDA to inform model building

Exploratory Data Analysis

- Data distributions
- Anomalies, errors, outliers
- Missing data patterns
- Feature covariation
- Nonlinearity between features and outcome

Modeling Decisions

- Standardizing
- Scaling
- Imputation
- Feature selection
- Algorithm selection

Small Group Activity

We will assign you to a small breakout room.

We will jump between rooms to join discussions and answer questions.

Please work through `day_1B_activity.R` to practice data splitting and EDA.

Our goal is for everyone to gain experience with all modeling processes, so everyone should work **individually** on their own code, rather than assign one person to share their screen and do the coding.

If you're stuck or have questions, please feel free to consult your group members!

Also feel free to discuss the topics covered with your group members. We hope you also learn from each other!