**Tutorial Sheet: Building an Infidelity Predictor in R**

# Overview

This tutorial walks you through building a predictive model using logistic LASSO regression, bagging, and classification trees. We will use the Affairs dataset from the AER package to predict whether someone has had an affair based on demographic and relationship factors.

# Learning Outcomes

By the end of this tutorial, you should be able to:

* Load and clean data from a package
* Convert variables to the correct type for modeling
* Train and **tune** logistic LASSO regression, bagging, and classification tree models
* Evaluate models using confusion matrices and out-of-bag error
* Interactively predict new data using a custom R menu

# Tutorial

## Part 1: Initial Setup

A black screen with green text

AI-generated content may be incorrect.

**What Each Package Does:**

* **tidyverse**: A collection of R packages for data manipulation (dplyr), visualization (ggplot2), and more. Great for cleaning and reshaping data.
* **caret**: Short for *Classification and Regression Training*. A wrapper package that streamlines training and tuning machine learning models.
* **ipred**: Implements ensemble methods like bagging. Used here to train bagged decision trees.
* **rpart**: Builds classification and regression trees (CART models). Simple, interpretable decision trees.
* **AER**: Provides datasets and functions for applied econometrics. Supplies the Affairs dataset we use in this tutorial.

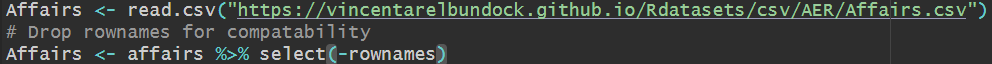
## Part 2 : Data Cleaning

A screen shot of a computer program

AI-generated content may be incorrect.

We turn the affair count into a yes/no target and make sure categorical variables are treated properly.

**Troubleshooting Tip:** If data("Affairs") doesn’t work (e.g., the AER package fails to load), you can try this alternative method:



## Part 3: Data Splitting and Balancing

Before training models, we need to split our dataset into training and testing subsets. This helps us evaluate performance fairly and avoid overfitting.

We'll use 70% of the data for training and 30% for testing. Additionally, because our target class (cheater vs. non-cheater) is imbalanced, we'll apply **upsampling** to the training set. Upsampling duplicates examples from the minority class so that the model doesn’t become biased toward predicting the majority.

A computer screen shot of a program code

AI-generated content may be incorrect.

Part 4: Model Training and Tuning

### Logistic LASSO Regression

A computer screen shot of text

AI-generated content may be incorrect.

Let’s break it down:

* set.seed(20346554): Ensures reproducible results across runs.
* 10^seq(-4, 1, length = 100): Creates 100 lambda values from 0.0001 to 10 to test.
* method = "glmnet": Uses the glmnet engine to train generalized linear models.
* alpha = 1: Specifies LASSO (L1 regularization). Alpha = 0 would be ridge.
* lambda: Controls how strongly to shrink coefficients. Larger = more shrinkage.
* trainControl(): Uses 10-fold cross-validation repeated 5 times to make training robust.
* tuneGrid: Tries out each lambda and picks the best.
* bestLamba: sweet spot lambda value that balances shrinking coefficients enough to avoid overfitting without losing important info. **This is our tuned parameter.**
* VerboseIter just shows the training progress

Do the loop again with best Lamba

A screen shot of a computer code

AI-generated content may be incorrect.

### Bagging (Bootstrap Aggregating)

A computer screen shot of a code

AI-generated content may be incorrect.

Let’s break it down:

* set.seed(20346554): Locks in reproducibility.
* bagging(): Trains an ensemble of decision trees by bootstrapping the data multiple times. In this case, this is a test bagging run before we tune it.
* grid : a Tuning map for use later

A computer screen shot of a program code

AI-generated content may be incorrect.

* nbagg: Number of trees to aggregate; more trees can reduce variance but cost more compute.
* cp (complexity parameter): Controls tree pruning—higher means simpler trees.
* minsplit: Minimum samples required to split a node in each tree.
* The for-loop tests different combos of nbagg, cp, and minsplit to find the best setup.
* OOB.accuracy: Out-of-bag accuracy—like a built-in test on unseen data during bagging.
* test.accuracy: Accuracy on the training set predictions.
* btree.bestTune: Picks the top combo with the best OOB accuracy.
* Final bagging() call trains the tuned bagged tree ensemble.
* Note: for the for loop, add the cat() statement to show its progress (it takes ages on older pcs)

### Classification Tree

A screen shot of a computer code

AI-generated content may be incorrect.

Let’s break it down:

* set.seed(20346554): Keeps results repeatable.
* rpart(): Fits a decision tree model on your training data, splitting data by features to classify cheating behaviour.
* train(): Uses caret’s framework to run cross-validation (here, 10-fold) to find the best complexity parameter (cp).
* tuneLength = 15: Tests 15 different values of cp to find the sweet spot between underfitting and overfitting.
* cp (complexity parameter): Controls how much the tree prunes — higher means simpler trees with fewer splits.
* ctree.tuned: Re-trains the tree with the best cp value found during tuning.

Part 5: The Menu system

This section lets users enter custom info and compare predictions across models.

### The main Loop

A screen shot of a computer

AI-generated content may be incorrect.

* Extremely similar to a python style menu loop. Feel free to edit the title
* Note the usage of cat() over print()

### The Prediction part

#### Hmmm

* **Here’s the interesting part:** R has a input function similar to python, but id like for it to be able to be safely handled. So heres a function for that  
  A computer screen shot of a program code

  AI-generated content may be incorrect.
* Obviously put this OUTSIDE the main menu loop
* Feel free to just use readline(), this is to make it feel more polished as a program.

#### User input

* Now for the inputing from the user, we can specify what we need in particular from them, as well as the limits as to what they can input (I am NOT predicting the infidelity of a 3000 year old)

A black background with white text

AI-generated content may be incorrect.

* This is an example for the input. Youll have to do it for gender, age, years married, children, religiousness, education, occupation and rating. If you are stuck or lazy feel free to use the answer script provided in the github.

#### Building a prediction frame

* Now that inputs sorted, we would now have to build a data frame with all of the details.

A screen shot of a computer program

AI-generated content may be incorrect.

#### Optional: Reminding the user what they’ve entered

* No comments here, just as stated

A black background with white text

AI-generated content may be incorrect.

#### Predicting ✨

* This is pretty simple, just use predict(model , data ) and an optional type for bagging and classification trees.
* Also a mapping of tune and false to cheater non cheater, have fun with the labels lol

A computer code on a black background

AI-generated content may be incorrect.

#### Prediction results.

* This just uses cat()

A screen shot of a computer screen

AI-generated content may be incorrect.

# Congratulations, you’ve built an infidelity predictor 😭