Spaceship Titanic: Applying Stacking Methods to a Kaggle Competition

Jordan Boswell, Thuy Nguyen

Abstract

Stacking is an effective ensemble method, yet there is a lack of literature exploring its best practices and implementation details. In this paper, we explore the effectiveness of various stacking and non-stacking models when applied to the Spaceship Titanic classification competition of Kaggle. The results show that stacking is generally better than single models, but the effectiveness of stacking depends on the training method used.

Introduction

Stacking, first proposed by Wolpert in 1992 [1], is an ensemble method for supervised learning problems that combines multiple strong base models into a meta model. The main idea is to use the predictions of the base models as input predictors to the meta model. The meta model, recommended to be from the family of regularized linear models [2], works best when its predictors are uncorrelated and span the solution space [3]. A learned combination of the base models' predictions is used to produce a response that ideally is more accurate than any prediction from a single model.

Empirical results from data science competition winners have shown stacking to be an optimal method for predictive modeling [4]. For example, the winner of the $1,000,000 Netflix Prize used stacking, which improved upon Netflix's existing model's RMSE by over 10% [5, 6]. There are also some theoretical results supporting the efficacy of stacking: research has proven that stacking can produce an asymptotically optimal estimator [7].

Stacking has many valid variations. Despite the widespread use and success of stacking in competitions, there is no consensus about the best type of stacking to perform. The number of base models, the types of base models, the choice of predictors, the type of meta model, and the model training and selection strategy are just some of the hyperparameters that get introduced when building a stacking model. The choice of model training and selection strategy is especially relevant to performance. Since stacking adds an additional layer of models, care must be taken to balance the amount of data each layer gets to train on as well as the amount of data leakage (a situation where the model is trained with information that is not available during prediction, which can degrade results).

In this paper, we apply multiple stacking and non-stacking methods to the Spaceship Titanic competition, an active supervised learning competition hosted by Kaggle [8]. The competition uses a synthetic dataset, and is loosely modeled after the passenger logs of the real-life Titanic. The story behind the Spaceship Titanic competition is that, in the future, a spaceship is transferring passengers from the solar system to nearby star systems when it encounters a spacetime anomaly that causes some of the passengers to be transported to an alternate dimension. The goal is to use the ship's passenger logs to build a model that can predict whether or not a passenger was transported. Competition contestants are given a training dataset with the transported label included, and a test dataset that has the transported label hidden. Contestants are judged based on classification accuracy of the transported response for the test set.

Our results show that our stacking models generally perform better than the single models, although some stacking variations perform relatively poorly. Our best stacking model achieved position of 285 out of 2400 on the competition's leaderboard, and we see many areas for potential improvement. The following sections outline the dataset, the feature engineering performed, the missing data imputation method used, the model building methodology, and the results.

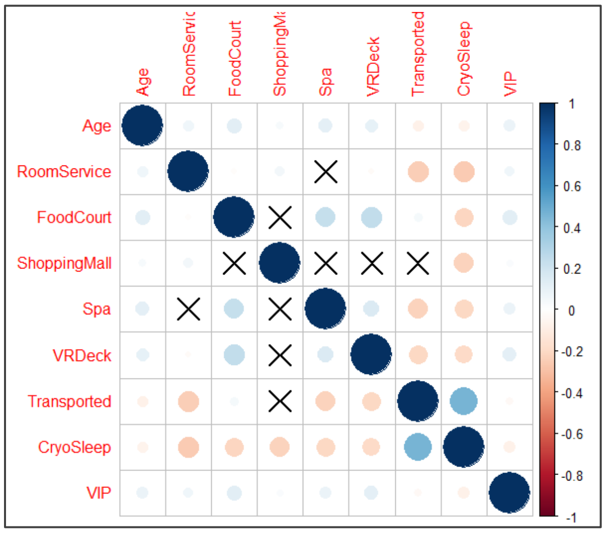
Data

Overview

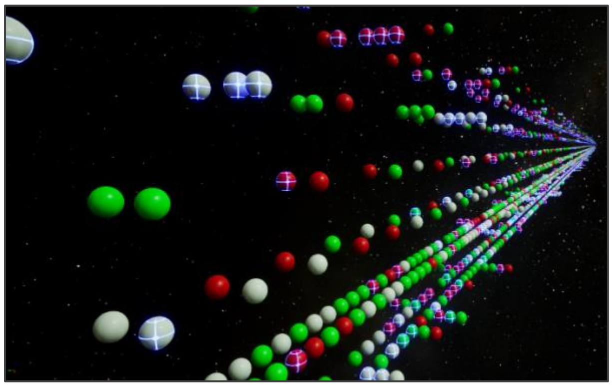
The data are split by Kaggle into a train and test set having 8693 and 4277 rows, respectively, with each row representing a unique passenger. The train set has the Boolean response, Transported, but the test set doesn't. Both sets have the following 13 predictors:

- PassengerId: The passenger's unique ID. It consists of a group ID portion and an individual (within group) portion.
- HomePlanet: The planet that the passenger departed from.
- Destination: The celestial body that the passenger is travelling to.
- CryoSleep: A Boolean value representing whether the passenger is in suspended animation or not.
- Cabin: The passenger's room, consisting of deck, side, and room number components.
- Age: The passenger's age in years.
- VIP: A Boolean value representing whether the passenger has VIP status or not.
- RoomService, FoodCourt, ShoppingMall, Spa, VRDeck: The amount of money the passenger spent at various locations of the ship.
- Name: The name of the passenger.

Roughly 2% of the data are missing, with the missingness seemingly being randomly, uniformly distributed (Figure 1). There exist relationships between the predictors and the response, as can be seen in the following correlation plot of the numeric variables:



There are significant positive correlations between three of the spending predictors and Transported, and there is significant negative correlation between CryoSleep and Transported. A more detailed view of the data was produced using Unreal Engine 5, a real-time rendering program. The following figure shows a render:



The passengers were mapped to their estimated spatial locations, based on their cabin. Each sphere represents a passenger. Decks are represented by position on the up/down axis, sides are represented by the separation on the left/right axis, and cabin numbers are represented in order by position on the forward/backward axis. Red represents transported passengers, while green represents those not transported, and gray represents passengers in the test set. Passengers with a luminous blue cross indicate cryosleep. One takeaway from this visualization is that transportation seems to occur in spatial clusters. Overall, our data exploration shows some relationships between the variables, but the actual cause of the transportation is unclear and seems to require a machine learning model produce accurate predictions.

Imputation

Manual

Exploration of the data found some patterns that held in all or almost all cases. The patterns found are:

- Passengers in crysleep don't spend any money
- Children under age 13 don't spend any money (Figure 2)
- Some decks exclusively contain passengers from certain home planets
- People in the same group are on the same side of the ship
- People in a group usually share the same cabin
- Groups always have one home planet, although they can have multiple destinations
- No one from Earth has a VIP status

The certainty of these rules allowed us to manually impute missing values that were subject to them. For example, for any passenger who was in cryosleep and who had a missing value related to spending, we set the spending value to zero.

Random Forest Iterative Imputation

After performing manual imputation of the data, about 2% of the data were still missing. We first used Little's MCAR test [9] to test if the missing data were missing completely at random. This is important because the MCAR property allows the imputation of data without the introduction of bias. The results of the test were in favor of not rejecting the null hypothesis, and thus favored the assumptions that the missing data were missing completely at random (p-value of 0.47).

We imputed the remainder of the missing data using the missForest R library, which provides non-parametric iterative imputation functionality that has been shown to be a state-of-the-art method for data imputation [10]. We first stacked the train and test set together, and removed the response variable. The stacked dataset was fed into missForest, which performed the following steps to impute the data:

1. Impute missing numeric values using column-wide means, and impute categorical values using column-wide modes
2. Iterate the following until the between-iteration changes are below a small threshold:
    a. For each column C:
        i. Fit a random forest of 1500 trees with C as the response and all other columns as predictors
        ii. Impute all missing values in C using the random forest

Feature Engineering

A new dataset was created with one row per Cabin. It contained features related to the number and percentage of transported passengers in the cabin as well as neighboring cabins. A similar dataset was created with one row per group. These datasets were joined back on to the original dataset to add columns related to group and cabin composition.

Additional features were added to the dataset:

- The cabin was parsed and split into three columns:
    - Deck: The deck letter

    - Side: The side of the ship; either S (starboard) or P (port)
    - Num: The room number
- PassengerId was split into IID (individual) and GID (group) components
- The spending columns were added to create a Spending column
- A Boolean HasSpent column was created to indicate if any spending had occurred
- The name was split into FirstName and LastName columns
- A Boolean IsChild column was created to indicate if passengers were 12 or younger (Figure 2)

Methodology

Three modeling strategies were used. The first strategy created single models that were tuned via cross-validation on the train set; the best models were chosen and then retrained on the entire train set. The other two types of models were stacking models. They were similar in all aspects except for the method of splitting the data for model training.

Before building any models, all of the data were transformed to either numeric variables or categorical variables encoded as dummy variables. All methods tuned LASSO logistic regression, random forest, and XGBoost models via 10-fold cross-validation. The random forest and XGBoost families of models were chosen because of their success over the last decade in data science competitions [4], as well as their empirical performance on tabular datasets [11, 12, 13]. The hyperparameters tuned for the models are shown below:

- LASSO Logistic Regression: L1 penalty
- Random Forest: Number of trees, number of sampled predictors, and minimum node size
- XGBoost: Number of trees, number of sampled predictors, minimum node size, tree depth, learning rate, minimum loss reduction, and the sample size proportion

During the cross-validation tuning process, each of the three model families tested 200 models; this number was shared across all models to prevent one model from overfitting the validation set relative to the others by chance. The hyperparameter sets were created using a latin-hypercube space-filling algorithm. The single model approach used the three models directly, while the stacking approaches used them as base models, utilizing their predictions as the (only) inputs to a logistic regression meta model. Both stacking methods used the out-of-fold predictions, generated during 10-fold cross-validation, as the input to meta model.

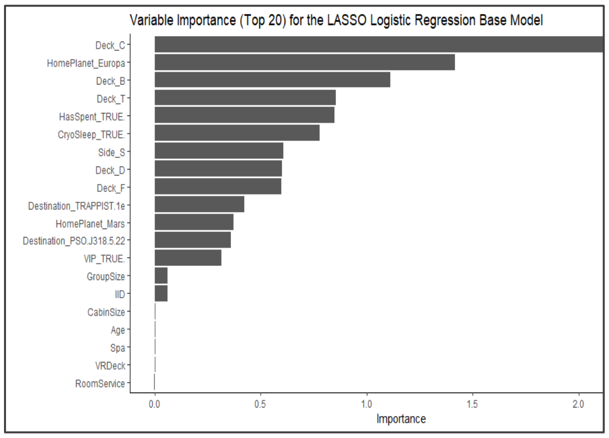
One stacking method, which we will refer to as method A, used the entire train set to use for cross-validation tuning of the base models. The other stacking method, method B, first split the train set into two parts, using 25%/75%, 50%/50%, and 75%/25% splits. The first split was used to tune the base models, and the second split was used to generate predictions that would be used to train the meta model. Method B reduces data leakage at the expense of using smaller training data. Each stacking method had two variants: one used the base models' probability-based predictions, and the other used the Boolean predictions. After training the meta models, both stacking methods refit the base models on the entire training data, generated predictions on the test set, and utilized the test set predictions with the meta model to make the final predictions, which were then submitted to Kaggle for scoring.

Results

LASSO Logistic Regression

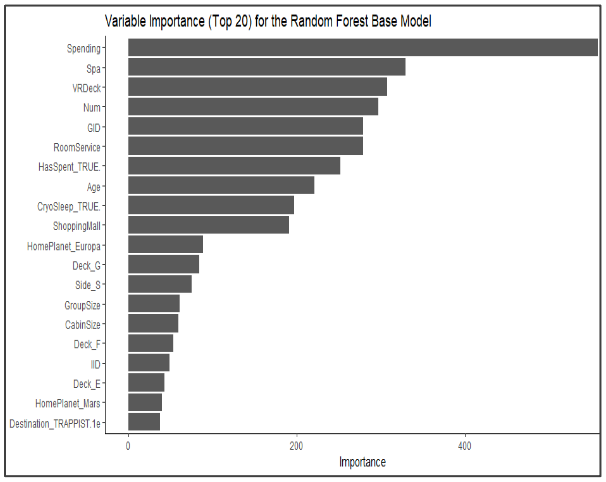
The best performing LASSO model had an L1 penalty of 0.0002437. Its out-of-fold prediction accuracy was 79.34%, and its ROC AUC was 0.8825. Its penalty caused some coefficients to get pushed towards zero, and it performed automatic variable selection, setting two coefficients to exactly zero. The exact coefficients of the model can be seen in

Table 1. The variable importance plot clearly shows the relative importance of the variables in the model:



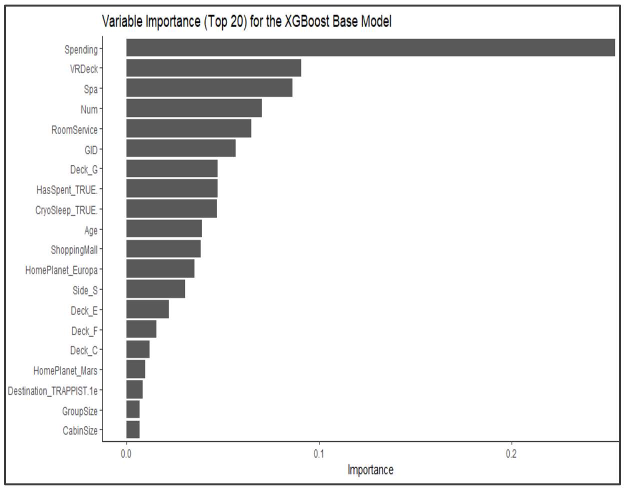
Random Forest

The best performing random forest model had 1683 trees, sampled seven predictors during node construction, and required a minimum of node size of five. It achieved an out-of-sample accuracy of 80.93% and had an ROC AUC of 0.8951. It had the following variable importance scores:



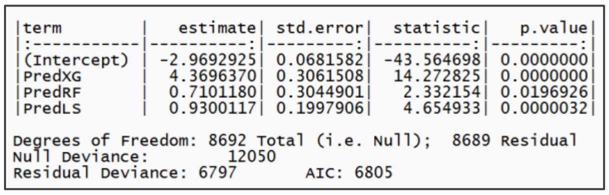
XGBoost

The best performing XGBoost model had 554 trees, sampled four predictors during node construction, required a minimum of node size of 20, had a learning rate of 0.0853, a loss reduction of 9e-06, a sample size proportion of 0.1620, and a maximum tree depth of seven. It achieved an out-of-sample accuracy of 79.31% and had an ROC AUC of 0.8803. It had the following variable importance scores:



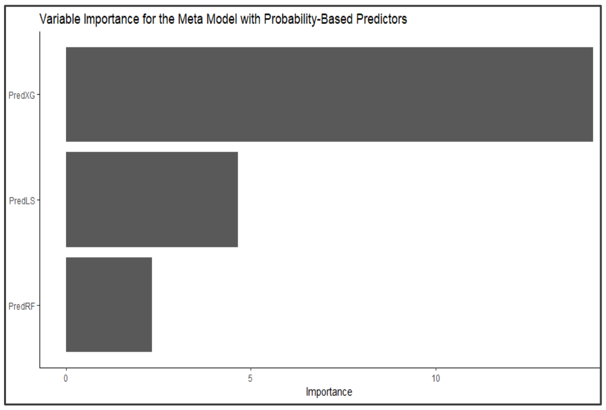
Stacking Method A

The method A stacking model that was trained with probability-based predictors had the following summary statistics:

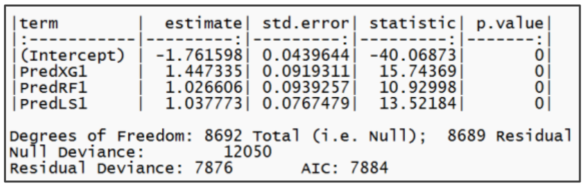

```
|term        |   estimate| std.error|  statistic|   p.value|
|:-----------|----------:|---------:|----------:|---------:|
|(Intercept) | -2.9692925| 0.0681582| -43.564698| 0.0000000|
|PredXG      |  4.3696370| 0.3061508|  14.272825| 0.0000000|
|PredRF      |  0.7101180| 0.3044901|   2.332154| 0.0196926|
|PredLS      |  0.9300117| 0.1997906|   4.654933| 0.0000032|

Degrees of Freedom: 8692 Total (i.e. Null);  8689 Residual
Null Deviance:       12050
Residual Deviance: 6797        AIC: 6805
```

It had the following variable importance scores:



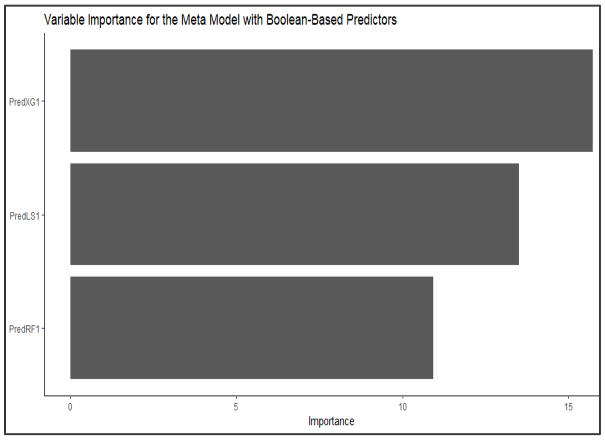
The method B stacking model that was trained with Boolean-based predictors had the following summary statistics:



```
|term         |  estimate| std.error| statistic| p.value|
|:------------|---------:|---------:|---------:|-------:|
|(Intercept)  | -1.761598| 0.0439644| -40.06873|       0|
|PredXG1      |  1.447335| 0.0919311|  15.74369|       0|
|PredRF1      |  1.026606| 0.0939257|  10.92998|       0|
|PredLS1      |  1.037773| 0.0767479|  13.52184|       0|

Degrees of Freedom: 8692 Total (i.e. Null);  8689 Residual
Null Deviance:      12050
Residual Deviance: 7876        AIC: 7884
```

It had the following variable importance scores:



Stacking Method B

Six models were produced with method B. They all had different estimated coefficients for the logistic regression meta model, although XGBoost was weighted highly in all of them.

Comparison of Test Set Classification Accuracy

LASSO	79.73%
Random Forest	79.50%
XGBoost	80.20%
Stacking A (Probability)	80.43%
Stacking B (Boolean)	80.22%
Stacking B 25%/75% (Probability)	79.99%
Stacking B 25%/75% (Boolean)	79.79%
Stacking B 50%/50% (Probability)	80.66%
Stacking B 50%/50% (Boolean)	80.64%
Stacking B 75%/25% (Probability)	79.89%
Stacking B 75%/25% (Boolean)	79.89%

Conclusions

XGBoost was the best performing base model, which is in accordance with empirical results shown in other competitions [4]. LASSO performed well and valued predictors differently than the other two base models. The different perspective that LASSO offered may have contributed to the effectiveness of the meta models.

The top four models were stacking models, followed by XGBoost, then the rest of the stacking models, and then the LASSO, and finally the random forest model. Stacking indeed seems to lead to improved performance. The best performing model, the stacking method B 50%/50% (probability) model, had an accuracy of 80.66%, which placed 285 out of 2400 on the competition's leaderboard.

The probability-based stacking models performed better than their Boolean-based counterparts. This makes sense since converting a numeric probability to a Boolean value results in significant data loss. The method A stacking model performed very well, as did the 50%/50% method B stacking model. There is no clear winner there. We suspect that if the data set was larger, method B would increase its performance gap between itself and method A, because the benefits of data-leakage reduction outweigh the reduction in training samples as the samples increase. Similarly, we suspect the converse would be true if the data set had fewer data points. The train splits chosen for method B seemed to have an impact on the final model, with an optimal choice somewhere near 50%/50%.

Some future improvements could be made to improve performance:

- Add some or all of the original predictors to the set of predictors for the meta models
- Build different stacking models by varying the training and cross-validation approaches
- Add additional base models
  - BART
  - NNs
  - XGBoost models with different hyperparameters

- Use sequential hyperparameter search instead of a fixed starting set of hyperparameters to test.
- Implement more of the advanced techniques used by Kaggle competition experts [3]

References

[1] D. H. Wolpert, "Stacked Generalization," *Neural Networks,* vol. 5, no. 2, pp. 241-259, 1992.

[2] M. Kuhn and J. Silge, "Ensembles of Models," in *Tidy Modeling with R*, O'Reilly Media, Inc., 2022.

[3] "Kaggle Ensembling Guide," MLWave, 11 June 2015. [Online]. Available: http://www.mlwave.com/kaggle-ensembling-guide/. [Accessed 16 November 2018].

[4] S. Russell and P. Norvig, "Learning from Examples," in *Artificial Intelligence: A Modern Approach, 4th Edition*, London, Pearson, 2021.

[5] "Netflix Prize," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Netflix_Prize. [Accessed 26 Novembetr 2022].

[6] A. Toscher, M. Jahrer and R. M. Bell, "The BigChaos Solution to the Netflix Grand Prize," *Netflix Prize Documentation,* pp. 1-52, 2009.

[7] M. J. van der Laan, E. C. Polley and A. E. Hubbard, "Super Learner," *Statistical Applications in Genetics and Molecular Biology,* vol. 6, no. 1, 2007.

[8] "Spaceship Titanic," Kaggle, [Online]. Available: https://www.kaggle.com/competitions/spaceship-titanic. [Accessed 25 Noveember 2022].

[9] R. J. Little, "A Test of Missing Completely at Random for Multivariate Data with Missing Values," *Journal of the American Statistical Association,* vol. 83, no. 404, pp. 1198-1202, 1988.

[10] D. J. Stekhoven and P. Buhlmann, "MissForest: Non-Parametric Missing Value Imputation for Mixed-Type Data," *Bioinformatics,* vol. 28, no. 1, pp. 112-118, 2012.

[11] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," *Journal of the Royal Statistical Society, Series B (Methodological),* vol. 58, no. 1, pp. 167-288, 1996.

[12] L. Breiman, "Random Forests," *Machine Learning,* vol. 45, no. 1, pp. 5-32, 2001.

[13] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* pp. 785-794, 2016.

Appendix: Additional Figures

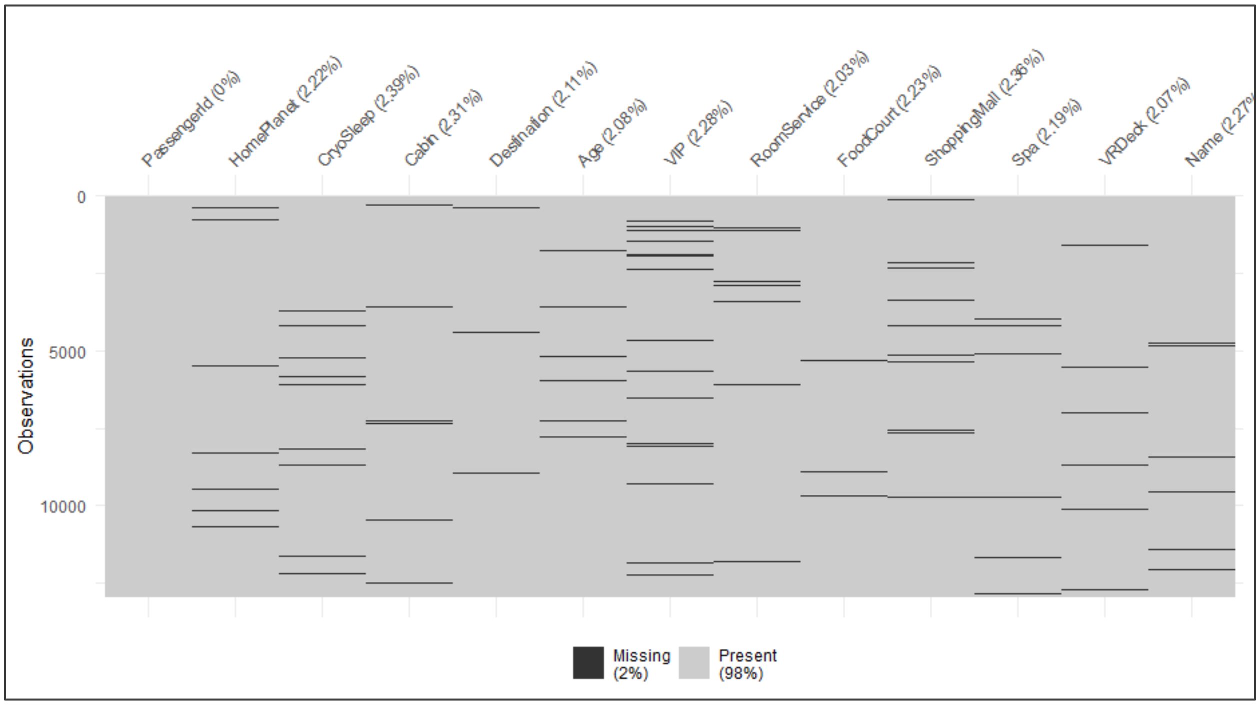


Figure 1. Missingness of the Predictors

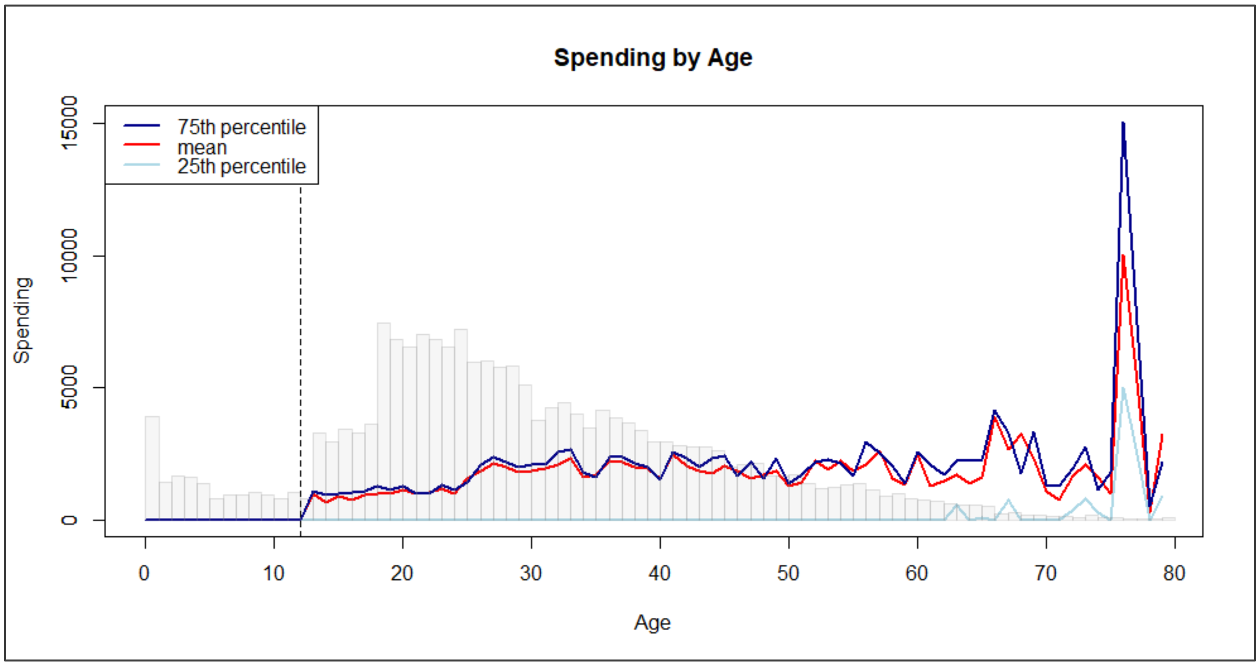


Figure 2. Spending Quartiles by Age Overlayed onto a Histogram of Age

Table 1. LASSO Logistic Regression Coefficients

Term	Estimate
(Intercept)	0.5106
Age	-0.0044
CabinSize	0.0038
GID	0.0002
GroupSize	-0.0615
IID	0.0614
Num	0.0001
RoomService	-0.0019
ShoppingMall	0.0001
Spa	-0.0025
Spending	0.0005
VRDeck	-0.0024
CryoSleep_TRUE.	0.7797
Deck_B	1.1103
Deck_C	2.3266
Deck_D	0.5987
Deck_E	0.0000
Deck_F	0.5967
Deck_G	0.0000
Deck_T	-0.8433
Destination_PSO.J318.5.22	-0.3570
Destination_TRAPPIST.1e	-0.4222
HasSpent_TRUE.	-0.8499
HomePlanet_Europa	1.4151
HomePlanet_Mars	0.3713
Side_S	0.6058
VIP_TRUE.	-0.3129

Appendix: Code

```r
numDesignMatColsFromDataset <- function(df) {
  num <- 0
  for (name in names(df)) {
    if (is.factor(df[[name]]))
      num <-
        num + length(unique(df[[name]][complete.cases(df[[name]])])) - 1
    else
      num <- num + 1
  }
  num
}

setColTypesForModeling <- function(df) {
  for (name in names(df)) {
    col_type <- typeof(df[, name])
    if (col_type == 'character')
      df[, name] <- as.factor(df[, name])
    else if (col_type == 'logical')
      df[, name] <- as.factor(df[, name])
    else if (col_type == 'integer')
      df[, name] <- as.numeric(df[, name])
  }
  df
}

updateDeckSideNum <- function (df) {
  df$Deck <-
    as.character(sapply(df$Cabin, function(x) {
      ifelse(is.na(x), NA, strsplit(x, '/')[[1]][1])
    }))
  df$Num <-
    as.integer(sapply(df$Cabin, function(x) {
      ifelse(is.na(x), NA, strsplit(x, '/')[[1]][2])
    }))
  df$Side <-
    as.character(sapply(df$Cabin, function(x) {
      ifelse(is.na(x), NA, strsplit(x, '/')[[1]][3])
    }))
  df
}

updateSpending <- function(df) {
  rsna <- is.na(df$RoomService)
  fcna <- is.na(df$FoodCourt)
  smna <- is.na(df$ShoppingMall)
  sna <- is.na(df$Spa)
  vrdna <- is.na(df$VRDeck)
  allna <- rsna & fcna & smna & sna & vrdna
  df$Spending <-
    ifelse(
      allna,
      NA,
      ifelse(rsna, 0, df$RoomService) + ifelse(fcna, 0, df$FoodCourt) + ifelse(smna, 0, df$ShoppingMall)
+ ifelse(sna, 0, df$Spa) + ifelse(vrdna, 0, df$VRDeck)
    )
  df$HasSpent <- df$Spending > 0
  df
}

updateSpendingOfCryo <- function(df) {
  df[df$CryoSleep %in% T, c("RoomService", "FoodCourt", "ShoppingMall", "Spa", "VRDeck")] <-
    0
  df$CryoSleep[df$HasSpent %in% T] <- F
  df
}

updateSpendingOfChildren <- function(df) {
  df[!is.na(df$Age) &
       df$Age <= 12, c("RoomService", "FoodCourt", "ShoppingMall", "Spa", "VRDeck")] <-
    0
  df <- updateSpending(df)
  df
}

updateSideFromGID <- function(df) {
  uni_GID <- unique(ship$GID)
  uni_side <-
    sapply(uni_GID, function(i) {
      df$Side[!is.na(df$Side) & df$GID == i][1]
    })
  dat <- data.frame(uni_GID, uni_side)
  for (i in uni_GID) {
    df$Side[is.na(df$Side) &
              df$GID == i] <-  dat$uni_side[dat$uni_GID == i]
  }
  df
}
```

```r
updateHomePlanetFromSurname <- function(df) {
  earth_sur <-
    unique(df$LastName[!is.na(df$LastName) &
                         !is.na(df$HomePlanet) & df$HomePlanet == "Earth"])
  mars_sur <-
    unique(df$LastName[!is.na(df$LastName) &
                         !is.na(df$HomePlanet) & df$HomePlanet == "Mars"])
  europa_sur <-
    unique(df$LastName[!is.na(df$LastName) &
                         !is.na(df$HomePlanet) & df$HomePlanet == "Europa"])
  uni_surname <- c(earth_sur, mars_sur, europa_sur)
  dat <-
    data.frame("home" = factor(c(
      rep("Earth", length(earth_sur)),
      rep("Mars", length(mars_sur)),
      rep("Europa", length(europa_sur))
    )),
    "LastName" = uni_surname)
  for (i in uni_surname) {
    df$HomePlanet[is.na(df$HomePlanet) &
                    !is.na(df$LastName) &
                    df$LastName == i] <- dat$home[dat$LastName == i]
  }
  df
}

updateHomePlanetFromDeck <- function(df) {
  df[!is.na(df$Deck) & df$Deck == "G", "HomePlanet"] <- "Earth"
  df[!is.na(df$Deck) &
       df$Deck %in% c("A", "B", "C", "T"), "HomePlanet"] <- "Europa"
  df
}

updateVIPFromHomePlanet <- function(df) {
  df$VIP[!is.na(df$HomePlanet) & df$HomePlanet == 'Earth'] <- F
  df
}

savePredictions <- function(passenger_ids, predictions, filename) {
    if (is.character(predictions))
        transported <- as.character(ifelse(predictions == 'TRUE', 'True', 'False'))
    else if (is.logical(predictions))
        transported <- as.character(ifelse(predictions, 'True', 'False'))
    else
```

```
        transported <- predictions
  write.table(
    data.frame(
      PassengerId = passenger_ids,
      Transported = transported
    ),
    file = paste0('predictions/', filename, '.csv'),
    quote = F,
    row.names = F,
    col.names = c("PassengerId", "Transported"),
    sep = ","
  )
}

library(missForest)
library(doParallel)
registerDoParallel(cores=12)

source('analysis/0 Functions.R')
ship <- read.csv('data/ship.csv')

ship <- setColTypesForModeling(ship)

imputation_cols <- c('Age', 'CabinNumCryo', 'CabinSize', 'CryoSleep', 'Deck', 'Destination',
'FoodCourt', 'GID', 'GroupSize', 'HasSpent', 'HomePlanet', 'IID', 'Num', 'RoomService', 'ShoppingMall',
'Side', 'Spa', 'Spending', 'VIP', 'VRDeck')
# Note: ntree values of 10,20,30,40,50 took 126,292,493,571,718 seconds, respectively
begin_time <- proc.time()
set.seed(1)
rf <- missForest(ship[, imputation_cols], maxiter=30, ntree=1500, parallelize="variables")
print(begin_time - proc.time())
ship_imputed_without_response <- rf$ximp

ship_imputed_without_response <- cbind(ship_imputed_without_response, ship[, !(names(ship) %in%
names(ship_imputed_without_response))])
ship_imputed_without_response <- updateSpending(ship_imputed_without_response)
ship_imputed_without_response <- updateSpendingOfCryo(ship_imputed_without_response)
ship_imputed_without_response <- updateSpendingOfChildren(ship_imputed_without_response)
ship_imputed_without_response <- updateHomePlanetFromDeck(ship_imputed_without_response)
ship_imputed_without_response <- updateVIPFromHomePlanet(ship_imputed_without_response)
ship_imputed_without_response$Cabin <- paste(ship_imputed_without_response$Deck,
as.integer(ship_imputed_without_response$Num), ship_imputed_without_response$Side, sep='/')

write.csv(ship_imputed_without_response, 'data/ship_imputed_without_response.csv', row.names=F)
```

```r
begin_time <- proc.time()
set.seed(1)
rf <- missForest(ship[, c(imputation_cols, 'Transported')], maxiter=30, ntree=1500,
parallelize="variables")
print(proc.time() - begin_time)
ship_imputed_with_response <- rf$ximp

ship_imputed_with_response <- cbind(ship_imputed_with_response, ship[, !(names(ship) %in%
names(ship_imputed_with_response))])
ship_imputed_with_response <- updateSpending(ship_imputed_with_response)
ship_imputed_with_response <- updateSpendingOfCryo(ship_imputed_with_response)
ship_imputed_with_response <- updateSpendingOfChildren(ship_imputed_with_response)
ship_imputed_with_response <- updateHomePlanetFromDeck(ship_imputed_with_response)
ship_imputed_with_response <- updateVIPFromHomePlanet(ship_imputed_with_response)
ship_imputed_with_response$Cabin <- paste(ship_imputed_with_response$Deck,
as.integer(ship_imputed_with_response$Num), ship_imputed_with_response$Side, sep='/')

write.csv(ship_imputed_with_response, 'data/ship_imputed_with_response.csv', row.names=F)


library(tidymodels)
library(vip)
library(xgboost)
library(randomForest)
library(glmnet)
library(doParallel)
doParallel::registerDoParallel()
source('analysis/0 Functions.R')

# Load the Data
ship_imp_nores <- read.csv('data/ship_imputed_no_response.csv')

# Format the Data
ship_imp_nores <- setColTypesForModeling(ship_imp_nores)
train <- ship_imp_nores[ship_imp_nores$Train == 'TRUE',]
test <- ship_imp_nores[ship_imp_nores$Train == 'FALSE',]

# Get the Columns and Calculate the Number of Columns in the Model Matrix
cols <-
  c(
    'Age',
    'CabinSize',
    'CryoSleep',
    'Deck',
    'Destination',
    'GID',
    'GroupSize',
    'HasSpent',
    'HomePlanet',
    'IID',
    'Num',
    'RoomService',
    'ShoppingMall',
    'Side',
    'Spa',
    'Spending',
    'Transported',
    'VIP',
    'VRDeck'
  )
num_cols <- numDesignMatColsFromDataset(train[, cols])

# Create the Recipe
rec <- recipe(Transported ~ ., data = train[, cols]) %>%
  step_dummy(all_nominal_predictors())

# Create the Folds
set.seed(1)
folds <- vfold_cv(train[, cols])

# Base Models

## XGBoost
xg_spec <- boost_tree(
  mtry = tune(),
  trees = tune(),
  min_n = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune(),
  sample_size = tune()
) %>%
  set_engine('xgboost') %>%
  set_mode('classification')

xg_wf <- workflow() %>%
  add_model(xg_spec) %>%
  add_recipe(rec)

set.seed(1)
```

```r
xg_grid <- grid_latin_hypercube(
  mtry() %>% range_set(c(1, num_cols)),
  trees(),
  min_n(),
  tree_depth(),
  learn_rate(),
  loss_reduction(),
  sample_size = sample_prop(),
  size = 200
)

set.seed(1)
xg_res <- xg_wf %>%
  tune_grid(resamples = folds,
            grid = xg_grid,
            control = control_grid(save_pred = T))

xg_best <- select_best(xg_res, 'accuracy')
xg_oos_pred <- do.call(rbind, lapply(xg_res$.predictions, function(x){x[x$.config == xg_best$.config,
c('.row', '.pred_TRUE')]}))
xg_oos_pred <- xg_oos_pred[order(xg_oos_pred$.row), ]
savePredictions(train$PassengerId, xg_oos_pred$.pred_TRUE, 'xg_oos')

xg_final_wf <- xg_wf %>%
  finalize_workflow(xg_best)
xg_final_fit <- xg_final_wf %>% fit(train)
xg_pred <-predict(xg_final_fit, test, type = 'prob')$.pred_TRUE
savePredictions(test$PassengerId, xg_pred, "xg_1")


## Random Forest
rf_spec <- rand_forest(mtry = tune(),
                       trees = tune(),
                       min_n = tune()) %>%
  set_engine('randomForest') %>%
  set_mode('classification')

rf_wf <- workflow() %>%
  add_model(rf_spec) %>%
  add_recipe(rec)

set.seed(1)
rf_grid <-
  grid_latin_hypercube(mtry() %>% range_set(c(1, num_cols)),
                       trees(),
                      min_n(),
                      size = 200)

set.seed(1)
rf_res <- rf_wf %>%
  tune_grid(resamples = folds,
            grid = rf_grid,
            control = control_grid(save_pred = T))

rf_best <- select_best(rf_res, 'accuracy')
rf_oos_pred <- do.call(rbind, lapply(rf_res$.predictions, function(x){x[x$.config == rf_best$.config,
c('.row', '.pred_TRUE')]}))
rf_oos_pred <- rf_oos_pred[order(rf_oos_pred$.row), ]
savePredictions(train$PassengerId, rf_oos_pred$.pred_TRUE, 'rf_oos')

rf_final_wf <- rf_wf %>%
  finalize_workflow(select_best(rf_res, "accuracy"))
rf_final_fit <- rf_final_wf %>% fit(train)
rf_pred <- predict(rf_final_fit, test, type = 'prob')$.pred_TRUE
savePredictions(test$PassengerId, rf_pred, "rf_1")


## Lasso
ls_spec <- logistic_reg(
  mode = "classification",
  penalty = tune(),
  mixture = 1) %>%
  set_engine ( "glmnet" )

ls_rec <- rec

ls_wf <- workflow() %>% add_model(ls_spec) %>% add_recipe(ls_rec)

ls_grid <- tibble(penalty=10^seq(-4, -0.5, length.out = 200))

set.seed(1)
ls_res <- ls_wf %>%
  tune_grid(resamples = folds,
            grid = ls_grid,
            control = control_grid(save_pred = T))

ls_best <- select_best(ls_res, 'accuracy')
ls_oos_pred <- do.call(rbind, lapply(ls_res$.predictions, function(x){x[x$.config == ls_best$.config,
c('.row', '.pred_TRUE')]}))
ls_oos_pred <- ls_oos_pred[order(ls_oos_pred$.row), ]
```

```
savePredictions(train$PassengerId, ls_oos_pred$.pred_TRUE, 'ls_oos')

ls_final_wf <- ls_wf %>%
  finalize_workflow(select_best(ls_res, "accuracy"))
ls_final_fit <- ls_final_wf %>% fit(train)
ls_pred <- predict(ls_final_fit, test, type = 'prob')$.pred_TRUE
savePredictions(test$PassengerId, ls_pred, "lasso_3")


# Meta Model
xg_oos_pred <- read.csv('predictions/xg_oos.csv')
rf_oos_pred <- read.csv('predictions/rf_oos.csv')
ls_oos_pred <- read.csv('predictions/ls_oos.csv')
oos_pred <- data.frame(PredXG = xg_oos_pred$Transported, PredRF = rf_oos_pred$Transported,
PredLS = ls_oos_pred$Transported)
test_pred <- data.frame(PredXG = xg_pred, PredRF = rf_pred, PredLS = ls_pred)
meta_train <- cbind(train, oos_pred)
meta_test <- cbind(test, test_pred)

meta_spec <- logistic_reg()
meta_rec <- recipe(Transported ~ PredXG + PredRF + PredLS, data = meta_train)
meta_wf <- workflow() %>% add_model(meta_spec) %>% add_recipe(meta_rec)
meta_final_fit <- meta_wf %>% fit(meta_train)

meta_pred <- (meta_final_fit %>% predict(meta_test, type = 'prob'))$.pred_TRUE

savePredictions(ship_imp_nores[ship_imp_nores$Train == 'FALSE', 'PassengerId'], meta_pred,
'meta_2')



nmod <- 200

for (pct_train_a in c(0.25, 0.50, 0.75)) {
  # Create the Splits.  Set A is for base model training; B is for oos predictions and training the meta
model
  set.seed(1)
  nrow_a <- round(nrow(train) * pct_train_a)
  nrow_b <- nrow(train) - nrow_a
  train_a_i <- sample(c(rep(T, times=nrow_a), rep(F, times=nrow_b)))
  train_b_i <- !train_a_i
  train_a <- train[train_a_i, ]
  train_b <- train[train_b_i, ]

  # Create the Folds
```

```r
set.seed(1)
folds_a <- vfold_cv(train_a[, cols])

# Base Models

## XGBoost
xg_spec <- boost_tree(
  mtry = tune(),
  trees = tune(),
  min_n = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune(),
  sample_size = tune()
) %>%
  set_engine('xgboost') %>%
  set_mode('classification')

xg_wf <- workflow() %>%
  add_model(xg_spec) %>%
  add_recipe(rec)

set.seed(1)
xg_grid <- grid_latin_hypercube(
  mtry() %>% range_set(c(1, num_cols)),
  trees(),
  min_n(),
  tree_depth(),
  learn_rate(),
  loss_reduction(),
  sample_size = sample_prop(),
  size = nmod
)

set.seed(1)
xg_res <- xg_wf %>%
  tune_grid(resamples = folds_a,
            grid = xg_grid,
            control = control_grid(save_pred = T))

xg_best <- select_best(xg_res, 'accuracy')
xg_final_wf <- xg_wf %>%
  finalize_workflow(xg_best)
```

```r
## Random Forest
rf_spec <- rand_forest(mtry = tune(),
                       trees = tune(),
                       min_n = tune()) %>%
  set_engine('randomForest') %>%
  set_mode('classification')

rf_wf <- workflow() %>%
  add_model(rf_spec) %>%
  add_recipe(rec)

set.seed(1)
rf_grid <-
  grid_latin_hypercube(mtry() %>% range_set(c(1, num_cols)),
                       trees(),
                       min_n(),
                       size = nmod)

set.seed(1)
rf_res <- rf_wf %>%
  tune_grid(resamples = folds_a,
            grid = rf_grid,
            control = control_grid(save_pred = T))

rf_best <- select_best(rf_res, 'accuracy')
rf_final_wf <- rf_wf %>%
  finalize_workflow(rf_best)


## Lasso
ls_spec <- logistic_reg(
  mode = "classification",
  penalty = tune(),
  mixture = 1) %>%
  set_engine ( "glmnet" )

ls_rec <- rec

ls_wf <- workflow() %>% add_model(ls_spec) %>% add_recipe(ls_rec)

ls_grid <- tibble(penalty=10^seq(-4, -0.5, length.out = nmod))

set.seed(1)
ls_res <- ls_wf %>%
  tune_grid(resamples = folds_a,
            grid = ls_grid,
            control = control_grid(save_pred = T))

  ls_best <- select_best(ls_res, 'accuracy')
  ls_final_wf <- ls_wf %>%
    finalize_workflow(ls_best)

  # Meta
  set.seed(1)
  folds_b <- vfold_cv(train_b[, cols])

  ## XGBoost
  xg_pred <- (xg_final_wf %>% fit(train) %>% predict(test, type='prob'))[[1]]
  xg_fit <- xg_final_wf %>%
    fit_resamples(folds_b, control=control_resamples(save_pred=T))
  xg_oos_pred <- do.call(rbind, lapply(xg_fit$.predictions, function(x){x[, c('.row', '.pred_TRUE')]}))
  xg_oos_pred <- xg_oos_pred[order(xg_oos_pred$.row), ]
  savePredictions(train_b$PassengerId, xg_oos_pred$.pred_TRUE, paste0('xg_oos_b_', pct_train_a))

  ## Random Forest
  rf_pred <- (rf_final_wf %>% fit(train) %>% predict(test, type='prob'))[[1]]
  rf_fit <- rf_final_wf %>%
    fit_resamples(folds_b, control=control_resamples(save_pred=T))
  rf_oos_pred <- do.call(rbind, lapply(rf_fit$.predictions, function(x){x[, c('.row', '.pred_TRUE')]}))
  rf_oos_pred <- rf_oos_pred[order(rf_oos_pred$.row), ]
  savePredictions(train_b$PassengerId, rf_oos_pred$.pred_TRUE, paste0('rf_oos_b_', pct_train_a))

  ## LASSO
  ls_pred <- (ls_final_wf %>% fit(train) %>% predict(test,type='prob'))[[1]]
  ls_fit <- ls_final_wf %>%
    fit_resamples(folds_b, control=control_resamples(save_pred=T))
  ls_oos_pred <- do.call(rbind, lapply(ls_fit$.predictions, function(x){x[, c('.row', '.pred_TRUE')]}))
  ls_oos_pred <- ls_oos_pred[order(ls_oos_pred$.row), ]
  savePredictions(train_b$PassengerId, ls_oos_pred$.pred_TRUE, paste0('ls_oos_b_', pct_train_a))

  xg_oos_pred <- read.csv(paste0('predictions/', 'xg_oos_b_', pct_train_a, '.csv'))
  rf_oos_pred <- read.csv(paste0('predictions/', 'rf_oos_b_', pct_train_a, '.csv'))
  ls_oos_pred <- read.csv(paste0('predictions/', 'ls_oos_b_', pct_train_a, '.csv'))
  oos_pred <- data.frame(PredXG = xg_oos_pred$Transported, PredRF = rf_oos_pred$Transported,
PredLS = ls_oos_pred$Transported)
  test_pred <- data.frame(PredXG = xg_pred, PredRF = rf_pred, PredLS = ls_pred)
  meta_train <- cbind(train_b, oos_pred)
  meta_test <- cbind(test, test_pred)

  meta_spec <- logistic_reg()
  meta_rec <- recipe(Transported ~ PredXG + PredRF + PredLS, data = meta_train)
  meta_wf <- workflow() %>% add_model(meta_spec) %>% add_recipe(meta_rec)
  meta_final_fit <- meta_wf %>% fit(meta_train)

  meta_pred <- (meta_final_fit %>% predict(meta_test, type = 'prob'))$.pred_TRUE > 0.5

  savePredictions(ship_imp_nores[ship_imp_nores$Train == 'FALSE', 'PassengerId'], meta_pred,
paste0('meta_b_', pct_train_a))

  # logical predictors
  oos_pred <- data.frame(PredXG = as.factor(xg_oos_pred$Transported>.5), PredRF =
as.factor(rf_oos_pred$Transported>.5), PredLS = as.factor(ls_oos_pred$Transported>.5))
  test_pred <- data.frame(PredXG = as.factor(xg_pred>.5), PredRF = as.factor(rf_pred>.5), PredLS =
as.factor(ls_pred>.5))
  meta_train <- cbind(train_b, oos_pred)
  meta_test <- cbind(test, test_pred)

  meta_spec <- logistic_reg()
  meta_rec <- recipe(Transported ~ PredXG + PredRF + PredLS, data = meta_train)
  meta_wf <- workflow() %>% add_model(meta_spec) %>% add_recipe(meta_rec)
  meta_final_fit <- meta_wf %>% fit(meta_train)

  meta_pred <- (meta_final_fit %>% predict(meta_test, type = 'prob'))$.pred_TRUE > 0.5

  savePredictions(ship_imp_nores[ship_imp_nores$Train == 'FALSE', 'PassengerId'], meta_pred,
paste0('meta_b_logical_', pct_train_a))
}
```