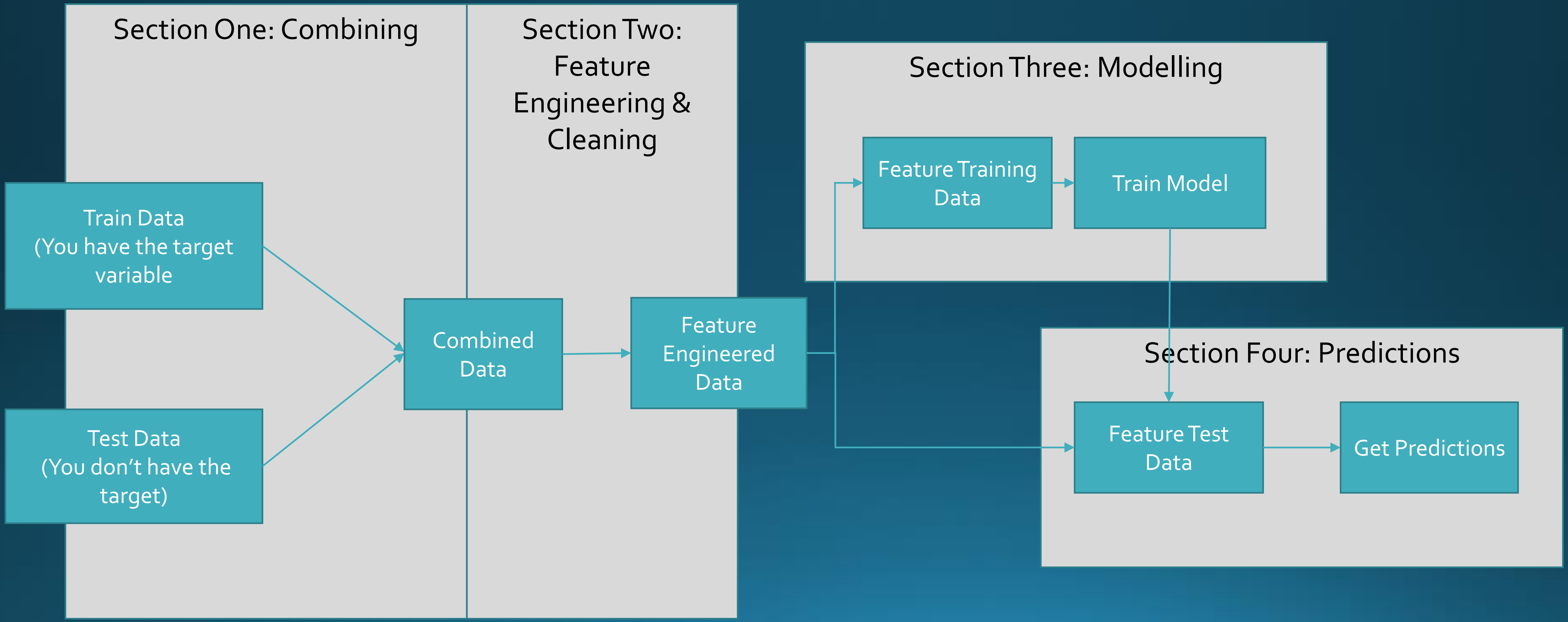
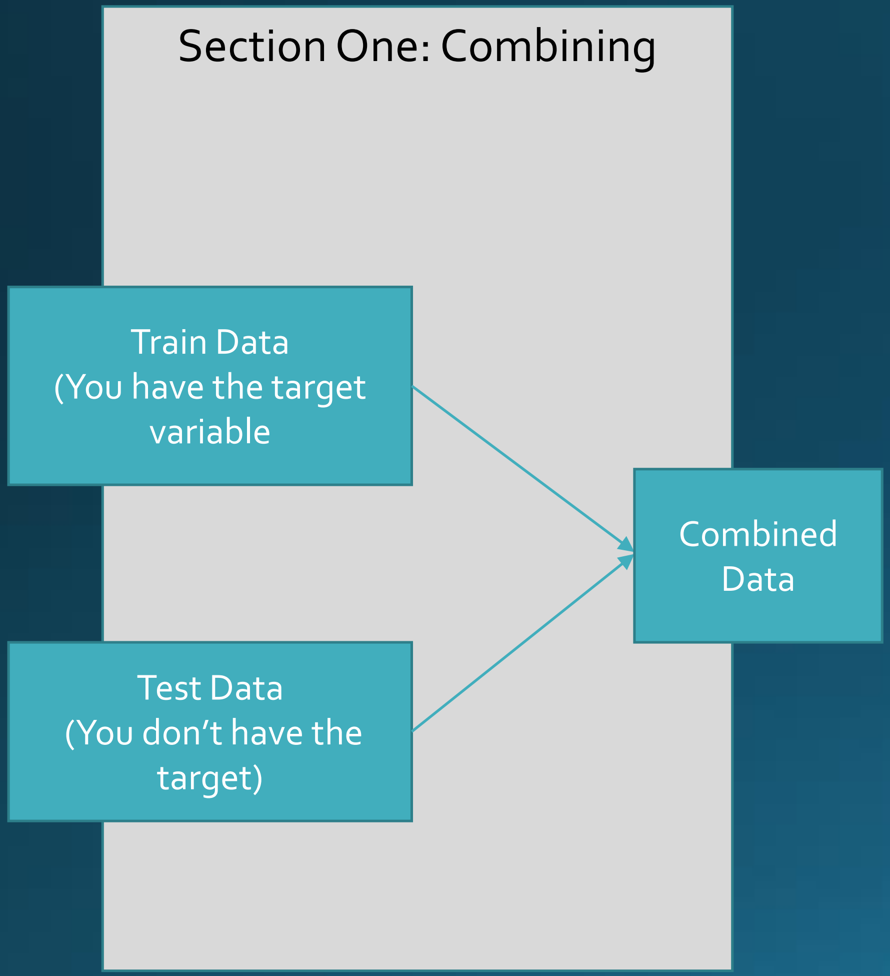
A brief guide on predictive modelling with xgboost and R

Michael Rodda
mrod0004@student.monash.edu

How I structure my process & code



Section One: Combining



This is a quick step but massively helps in keeping your code tidy and simple.

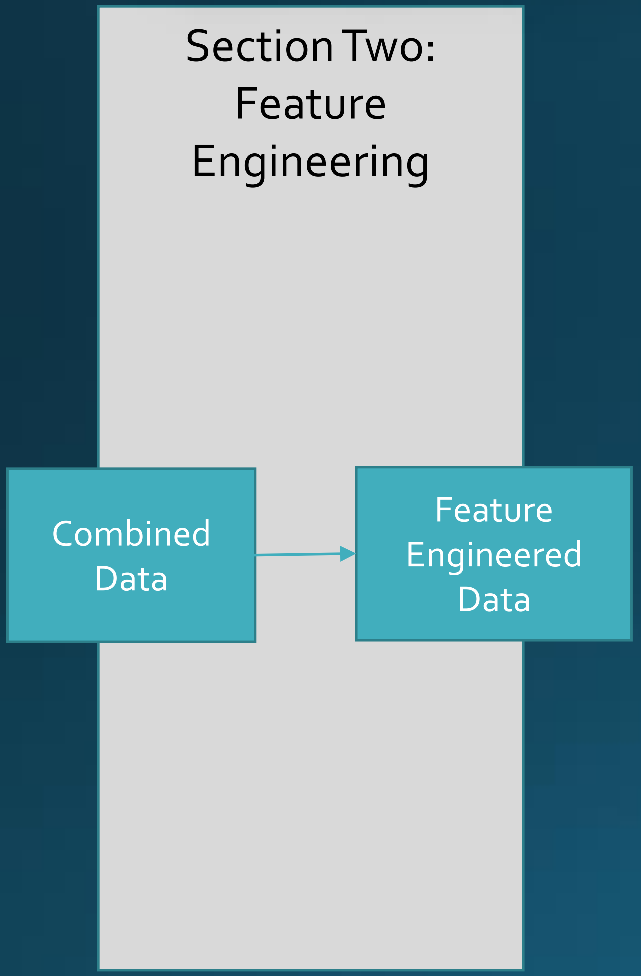
Why combine?
Its so in section two, all your feature engineering applies to both test and train sets and you don't have multiple bits of code (I learnt this the hard way in the Melbourne Datathon)

Use the rbind.fill()* package from plyr to combine the datasets. Your test data will then be all the observations with the target variable as NA

Later on when you need to split the feature engineered data back into test and train sets, it just becomes
train_df <- df %>% filter(!is.na(target))
test_df <- df %>% filter(is.na(target))

*Always load plyr before dplyr otherwise weird stuff happens

Section Two : Feature Engineering & Cleaning



- In my unqualified opinion, this is where 95% of your predictive power (read: Kaggle score) comes from
- The more data you have, the more features (attributes) you can throw in
- In our uni courses, we learn so much on avoiding overfitting by having simple models with fewer attributes

FORGET THAT FOR KAGGLE!

- We control the overfitting through the parameter tuning in Section Three
- xgboost is good in that it drops the bad features - If a variable split on the tree doesn't result in loss reduction, it wont do it (I think!)

Remember to clean your data and check for errors!

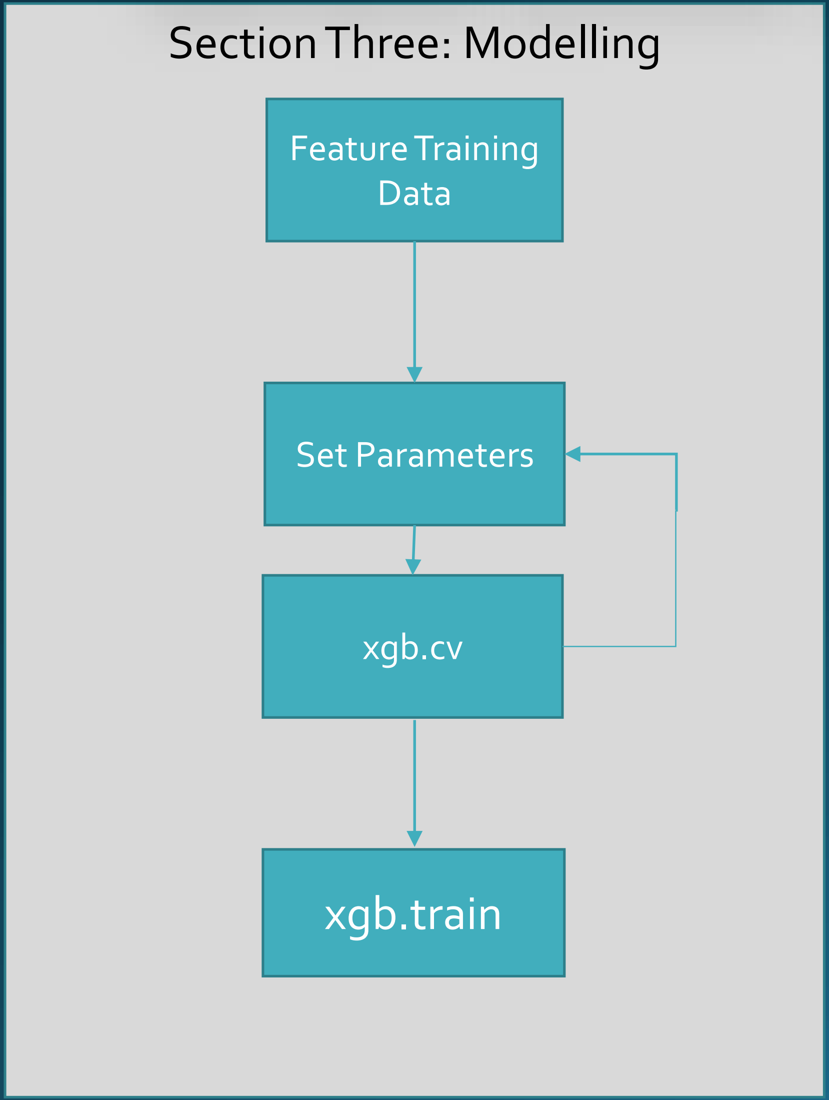
Section Two : Feature Engineering & Cleaning

Adding in Features:
If you think something from the data may even have a TINY influence on the target variable…
ADD IT IN AS A FEATURE

- Add in ratios of different columns?
- For each car model, time since first Year of Registration (ie a possible approximate of when a car model was released?)
- Breakdown postalcode into how rich an area is?
- NOT APPLICABLE TO THIS DATA: For time series, you can do lagging values, rolling mean values, standard deviations, rate of change of a value (velocity), rate of rate of change (acceleration), sum of last month, sum of last year, sum of last day…. Go crazy!

Transforming Features:
- Log transformation?
- Shifting data to mean of zero?
- Rankings?
- Standardised by maximum value?
- This barely scratches the surface, get googling!

Section Three: Modelling



The intimidating part. Try and understand my code after reading this

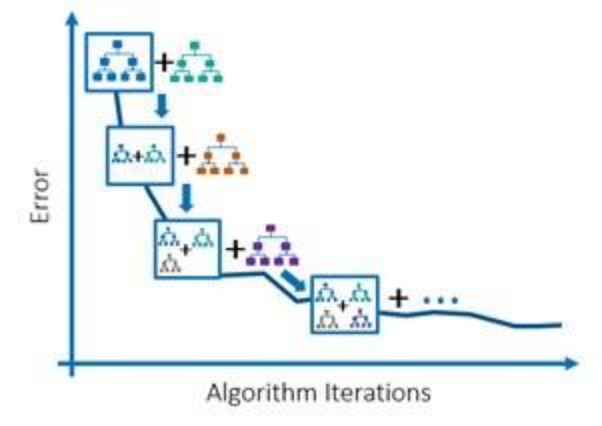
Things to note about xgboost

- It only takes numerical values
- Categorical values need to be changed to dummy variables/one hot encoded
- Dataframes in R need to have as.matrix() wrapped around them when used as inputs to xgboost
- NOT APPLICABLE: If doing binary classification, the target variable needs to be an integer

- xgb.cv is doing k-fold validation (what you learn in modelling/algorithms)
- Use set.seed().
- Its an iterative process: Set some parameters, see result, change one parameter and note the change.
- Don't get bogged down on spending too much time tweaking parameters, most of your score improvements will be from feature engineering.

Section Three: Modelling

- Xgboost is an implementation of gradient boosted trees
- Recall doing a tree model in Modelling for Data Analysis (if you haven't done that unit – look up CART models)
- Xgboost builds another tree on the errors of the first tree
- So its still a tree based algorithm – ie majority of parameters deal with how it grows the trees

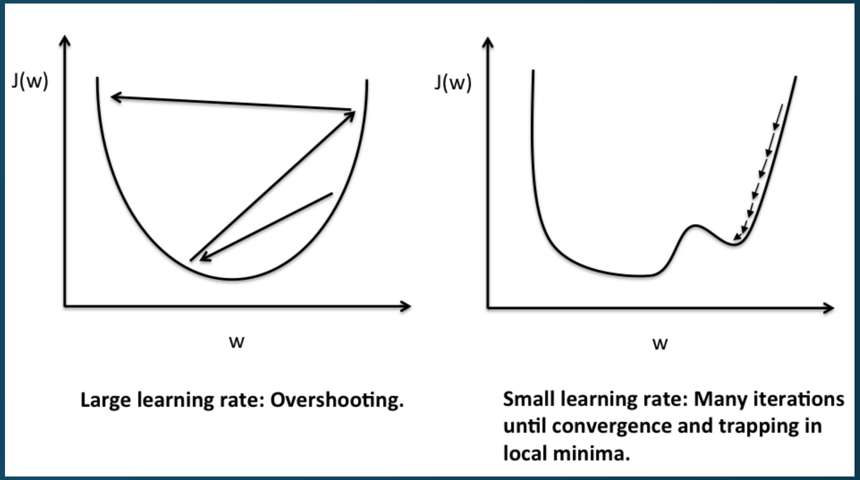


https://github.com/dmlc/xgboost/blob/master/doc/parameter.md
Parameters that I use (there's a lot more for different implementations of xgboost)

- eta (aka learning rate) – default = 0.3, range = [0,1]
- max_depth – default = 6, range = [0, inf]
- gamma – default = 0, range = [0, inf]
- min_child_weight – default = 1, range = [0,inf]
- max_delta_step – default = 0, range = [0, inf]
- subsample – default = 1, range = (0, 1]
- colsample_bytree –default = 1, range = (0,1]
- colsample_bylevel – default = 1, range = (0,1]

Section Three: Modelling

- eta (aka learning rate) – default = 0.3, range = [0,1]
  - The learning rate. Going lower increases the time it takes to converge
  - Go as low as your patience/computer allows you to
  - I keep this around default (maybe a little bit lower, eg 0.1) when I'm still experimenting with other parameters



- max_depth – default = 6, range = [0, inf]
  - How many levels the trees can grow
  - Increase – Massive increase in overfitting
  - Decrease – Reduces overfitting
  - In my (limited) experience, the biggest influence on overfitting

- gamma – default = 0, range = [0, inf]
  - The minimum loss reduction required to make a split on the tree
  - This influences both test and training rate
  - Increasing it reduces overfitting
  - I find this is the nuclear option – use other parameters to reduce overfitting before trying this

Section Three: Modelling

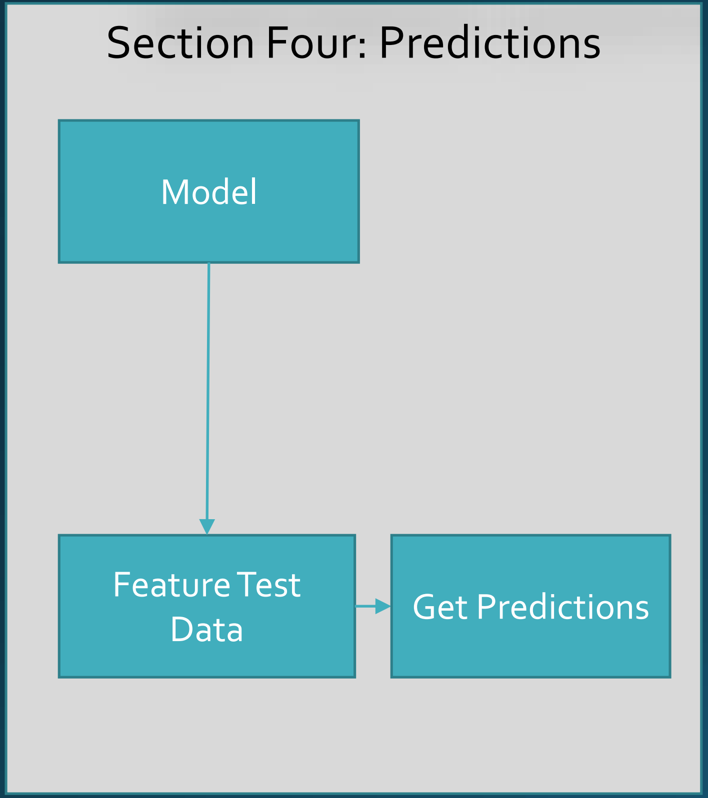
- min_child_weight – default = 1, range = [0,inf]
  - I haven't used this much
  - Increasing makes model more conservative
- max_delta_step – default = 0, range = [0, inf]
  - I don't really use this much
  - Increases this makes model more conservative
- subsample – default = 1, range = (0, 1]
  - How much of the data is used to grow a tree
  - Eg 0.5 means it randomly subsamples half of the data for each tree
  - Reducing this makes model more conservative
- colsample_bytree –default = 1, range = (0,1]
  - Column subsampling
  - Reducing this makes model more conservative
- colsample_bylevel – default = 1, range = (0,1]
  - Same as above but on the level basis
  - Reducing this makes model more conservative
- NOT APPLICABLE TO THIS KAGGLE – scale_pos_weight
  - Use when dealing with unbalanced classes for binary classification

Section Three: Modelling

- My process for parameter tuning (This may be completely anecdotal, so try your own process)
  1. Change max_depth
  2. If max_depth is low, experiment with subsample, colsample_bytree
  3. If max_depth is high, experiment with min_child_weight, max_delta_step then the subsampling
  4. Finally, see how gamma influences the learning rate
  5. Reduce eta and see how it influences the test error

- If you want to be lazy, do an automated grid search (google this)
  - Use expand.grid() with all inputs as the ranges of parameters you want to try
  - Write a loop that goes through all the different combinations and records the best error for each combination
  - Let it run for a few hours / overnight

Section Four: Predictions



A relatively simple step – You take the trained model from xgb.train() and use it with predict() to get a vector column of predictions

- With linear:regression – It's a vector of predictions
- With binary:logistic – It's a vector of probabilities

A few things to note

- The test data needs to have as.matrix() wrapped around it
- If your Kaggle score is WAY OFF compared to your own test.cv score
  - Check that all columns in the test data match your train data
  - Make sure the order of columns in your Kaggle upload is in the correct order, ie "id", "price"

- Keep your process organised!
- Make a log of your internal test score and what you features/parameters you changed

Getting started:

- Attached to this github repo is a starting script
- If it works, no need to touch section one and four
- Focus on creating features for section two and parameter tuning on section three
  - I've only changed two features in the starter script
    - One Hot encoded the brand
    - Combined Year and Month of registration
  - Parameters are left as default