Final Project for Math 390 at Queens College
May 24, 2020

By [Steven Grgas]
In collaboration with:
[Andrew Claros]

Abstract

When buying or selling an apartment, what is the first piece of information that one seeks out? Often, people would like to know the listing price. Everyone lives on a budget and needs to know how much things cost. But the listing price is not as important as the sale price. We would like to know how accurate the listing price is to the sale price when we buy or sell an apartment. Websites like Zillow.com design algorithms that give us an estimate value of each home. There are many factors that go into determining a fair price. For instance, condos tend to be more expensive than co-ops. Zillow's algorithms know this because of the plentiful data that goes into them that comprises of many variables. However, if you consider Zillow's rate of success, it is easy to see that they are fairly inaccurate. In order to beat Zillow's estimate, we use many tricks available in the machine learning toolbox.

1. Introduction

The motivation for using algorithms is that we don’t know the actual value of a home before it is sold. If we can get an estimate value of homes, we can help the seller and potential buyers influence their financial decision making. In order to provide an estimate value, we will need to use machine learning skills. First, we need a dataset; columns for our variables and rows for each home. What the algorithms will do is take in data and provide us with a function that will allow us to estimate the value of a new home, given some new data. This will be our response. The algorithms that were used were OLS, regression trees and random forests. The performance results were the best for random forests and worst for OLS. It was noticed that the data was negatively skewed, meaning that there were more homes with high costs. This would make the data points nonlinear, which means OLS is not an accurate model.

2. The Data

The data used for this project originally came from Amazon’s MTurk. This provided us with a data frame of size 2231 x 55 of some relevant information on characteristics of specific apartments and the final sale price. It provides some important information as to how certain data could influence the sale price, including certain features of the homes such as number of rooms, square footage, etc. However, the model was underfit and the RSME was very large. The data was helpful as it represented what people valued in a home but it was not enough. We decided to add data from zipdatamaps.com. We supplemented our data with fifteen columns of relevant data, making our new data frame have size 2231 x 70. This new data consisted of sociological data that are directly correlated with home prices. There were some outliers in the sale price for rich and poor neighborhoods. Extrapolation of the model is likely with some of the features. The sociological features such as unemployment rate, median household income, population density and race are destined to change over time. Renovation of apartments may change the number of floors, number of rooms, and taxes. This data should be kept up to date as one continues to run the model for future costs of apartments.

2.2 Featurization

The data provided by MTurk and zipdatamaps.com was direct for this project and we did not need to featurize any data provided.

2.3 Errors and Missingness

The data frame from MTurk occasionally contained misspelled words, ‘0’ or ‘1’ instead of ‘yes’ or ‘no,’ and missing data. To correct the mistypes, we simply changed the misspelled words to their correct spelling and turned a ‘0’ to ‘no’ and ‘1’ to ‘yes’ because we needed consistency in order to run the algorithms. When viewing the data we observed that many columns were missing essential data. If the column was at least 50% complete, we would impute the missing data via the random forest algorithm. This allows us to enjoy running algorithms that contain quality features for our experiment without computing nonsense results.

3.1 Regression Tree Modeling

The top ten features for regression trees were number of full bathrooms, apartment type, approximate year built, common charges, number of bedrooms, number of bathrooms, maintenance cost, number of floors in building, walk score, and kitchen type combo.

3.2 Linear Modeling

The OLS algorithm had in-sample RMSE = $71,000 and $R^2 = 0.83$. In-sample RMSE is high, but this is expected since the data is nonlinear. R^2 scored better than expected, but the model is not accurate enough to ship out. The ten best features for regression tree have the following coefficients for the linear model: Number of full bathrooms = 8.4×10^4, apartment type = 2.46×10^5, approximate year built = -2.27×10^5, common charges = 1.4×10, number of bedrooms = 5×10^4 , maintenance cost = 1.16×10^2, number of floors in building = 2.215×10^3, walk score = -1.8×10^2, kitchen type(eat-in) = -1.45×10^4, kitchen type(efficiency) = -2.02×10^4. If z represents the coefficient for one of these variables, then a change in one unit for the x value will result in a shift by z.

3.3 Random Forest Modeling

Since OLS is a linear model, it can be ruled out as the best model. Regression trees is a good algorithm, but random forests always beats it because it trains on subsets of the data and chooses the best one. Regression trees work on one data set. The oos RMSE = $74,403 for regression trees.

4. Performance Results for Random Forest Modeling

$R^2 = 0.908$ which means the model was very accurate. RMSE = $52,795. This was the lowest RMSE out of each algorithm. For generalization error, we would estimate $R^2 = 0.85$ and RMSE = $60,000 due to change in many factors of real estate.

5. Discussion

The biggest problem with this project was the lack of good data. There was a lot of missingness that had to be imputed. Once imputed, the errors were still high, so we needed to seek elsewhere for good data. This is when we thought about social issues such as population density and income. These factors largely influence housing costs so we felt they were important to include. In order to keep this model accurate in time, relevant information needs to be updated.

Acknowledgments

Family member Dan Price who helped with his knowledge of real state.

References

zipdatamaps.com

Data Munging

```r
pacman::p_load(dplyr, skimr, lubridate, stringr, tidyr,ggplot2,randomForest,c
aTools,missForest,mice, VIM,YARF,mlr,data.table)

## YARF can now make use of 7 cores.

require(caTools)

scraped = read.csv('scraped_demographics.csv')
scraped=scraped%>%
  select(-c(X,racial_majority,url,population,total_households,inc_below50,inc
_bet_50_100,inc_above_100,average_sale_price))%>%
  mutate(zipcode = as.factor(zipcode))
scraped

##    zipcode       town med_income pop_density
## 1    11361    bayside      74579    2.687271
## 2    11362     little      79329    2.538889
## 3    11363     little      89073    2.618209
## 4    11364    oakland      73011    2.600662
## 5    11354   flushing      47974    2.727535
## 6    11355   flushing      48502    2.977806
## 7    11356    college      63255    3.023868
## 8    11357 whitestone      73016    2.584329
## 9    11358   flushing      68523    2.806129
## 10   11359    bayside      78919          NA
## 11   11360    bayside      77626    2.156200
## 12   11365      fresh      69995    2.732458
## 13   11366      fresh      82431    2.994468
## 14   11367   flushing      55313    2.741584
## 15   11412      saint      62628    3.262744
## 16   11423     hollis      60339    3.137044
## 17   11432    jamaica      56699    3.296238
## 18   11433    jamaica      41093    3.195210
## 19   11434    jamaica      56783    2.920816
## 20   11435    jamaica      51678    3.029057
## 21   11436    jamaica      51051    3.353073
## 22   11101       long      36133    2.443571
## 23   11102    astoria      45715    2.423702
## 24   11103    astoria      51217    2.296033
## 25   11104  sunnyside      49244    2.301750
## 26   11105    astoria      50741    2.275931
## 27   11106    astoria      45208    2.273259
## 28   11374       rego      53555    2.257781
## 29   11375     forest      69665    2.111808
## 30   11379     middle      64453    2.508537
## 31   11385  ridgewood      47716    2.829607
```

```
## 32   11004        glen      74878     2.817286
## 33   11005      floral     146600     1.325991
## 34   11411     cambria      80416     3.148821
## 35   11413 springfield      72995     3.182465
## 36   11422    rosedale      76463     3.233950
## 37   11426   bellerose      75884     2.904557
## 38   11427      queens      66639     3.070806
## 39   11428      queens      71446     3.459928
## 40   11429      queens      42750     3.443759
## 41   11414      howard      67161     2.444881
## 42   11415         kew      60876     2.245298
## 43   11416       ozone      51482     3.440017
## 44   11417       ozone      56312     3.138353
## 45   11418    richmond      53300     3.286737
## 46   11419       south      55072     3.720917
## 47   11420       south      61621     3.433504
## 48   11421   woodhaven      58075     3.303250
## 49   11368      corona      45741     3.808981
## 50   11369        east      51217     3.449616
## 51   11370        east      44700     4.215401
## 52   11372     jackson      50985     2.787417
## 53   11373    elmhurst      48378     3.216154
## 54   11377    woodside      49306     2.859735
## 55   11378     maspeth      56961     2.741028
```

```
housing_orig = read.csv("housing_data_2016_2017.csv")
set.seed(10000)

housing= housing_orig[,29:ncol(housing_orig)]



housing= housing %>%
  mutate(common_charges =as.numeric(gsub('[$, ]','',common_charges))) %>%
  mutate(dogs_allowed= factor(replace(dogs_allowed, str_detect(tolower(dogs_a
llowed),pattern='y'),'yes'))) %>%
  mutate(cats_allowed= factor(replace(cats_allowed, str_detect(tolower(cats_a
llowed),pattern='y'),'yes')))%>%
  mutate(fuel_type= na_if(fuel_type,c('Other')))%>%
  mutate(fuel_type= na_if(fuel_type,c('other')))%>%
  mutate(fuel_type= replace(fuel_type, str_detect(fuel_type,pattern='oil'),'g
as'))%>%
  mutate(fuel_type= replace(fuel_type, str_detect(fuel_type,pattern='none'),N
A))%>%
  mutate(fuel_type= factor(fuel_type))%>%
  mutate(kitchen_type= na_if(kitchen_type,c('1955')))%>%
  mutate(kitchen_type= na_if(kitchen_type,c('none')))%>%
  mutate(kitchen_type= replace(kitchen_type, str_detect(kitchen_type,pattern=
'Eat in'),'eatin'))%>%
  mutate(kitchen_type= replace(kitchen_type, str_detect(kitchen_type,pattern=
```

```
'Eat In'),'eatin'))%>%
  mutate(kitchen_type= replace(kitchen_type, str_detect(kitchen_type,pattern=
'eat in'),'eatin'))%>%
  mutate(kitchen_type= replace(kitchen_type, str_detect(kitchen_type,pattern=
'Combo'),'combo'))%>%
  mutate(kitchen_type= replace(kitchen_type, str_detect(kitchen_type,pattern=
'efficiemcy'),'efficiency'))%>%
  mutate(kitchen_type= replace(kitchen_type, str_detect(kitchen_type,pattern=
'efficiency kitchen'),'efficiency'))%>%
  mutate(kitchen_type= replace(kitchen_type, str_detect(kitchen_type,pattern=
'efficiency kitchene'),'efficiency'))%>%
  mutate(kitchen_type= replace(kitchen_type, str_detect(kitchen_type,pattern=
'efficiency ktchen'),'efficiency'))%>%
  mutate(kitchen_type= factor(kitchen_type))%>%
  mutate(maintenance_cost =as.numeric(gsub('[$, ]','',maintenance_cost)))%>%
  mutate(parking_charges =as.numeric(gsub('[$, ]','',parking_charges)))%>%
  mutate(sale_price =as.numeric(gsub('[$, ]','',sale_price)))%>%
  mutate(total_taxes =as.numeric(gsub('[$, ]','',total_taxes)))%>%
  mutate(listing_price_to_nearest_1000 = as.numeric(gsub('[$, ]','',listing_p
rice_to_nearest_1000)))%>%
  mutate(zipcode = str_extract(full_address_or_zip_code,'[0-9]{5}'))



summary(housing$town)

## Length  Class   Mode
##      0   NULL   NULL

test_df = housing%>%
  drop_na(sale_price)

housing = housing %>%
  drop_na(sale_price) %>%
  drop_na(approx_year_built) %>%
  drop_na(kitchen_type)%>%
  drop_na(num_bedrooms)%>%
  drop_na(num_total_rooms)%>%
  drop_na(zipcode)


#to beat
housing = select(housing, -c('model_type','full_address_or_zip_code','url','g
arage_exists','pct_tax_deductibl','num_half_bathrooms','date_of_sale','listin
g_price_to_nearest_1000','community_district_num','dining_room_type'))
housing= inner_join(housing,scraped)

## Joining, by = "zipcode"
```

```
## Warning: Column `zipcode` joining character vector and factor, coercing in
to
## character vector
```

```
housing=select(housing,-('zipcode'))
housing=housing%>%
  group_by(town)%>%
  filter(n()>10)

#housing = dplyr::select(housing, -c('model_type','full_address_or_zipcode','
url','garage_exists','pct_tax_deductibl','num_half_bathrooms','date_of_sale',
'zipcode','listing_price_to_nearest_1000','community_district_num','dining_ro
om_type','fuel_type','sq_footage'))




y= housing$sale_price
sale_ind = which(colnames(housing)=='sale_price')
impute=mice(housing[,-sale_ind],seed = 10000)
```

```
##
##  iter imp variable
##   1   1  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   1   2  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   1   3  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   1   4  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   1   5  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   2   1  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   2   2  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   2   3  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   2   4  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   2   5  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   3   1  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   3   2  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   3   3  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   3   4  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
```

```
ng  parking_charges  sq_footage  total_taxes
##   3   5  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   4   1  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   4   2  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   4   3  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   4   4  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   4   5  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   5   1  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   5   2  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   5   3  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   5   4  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes
##   5   5  common_charges  fuel_type  maintenance_cost  num_floors_in_buildi
ng  parking_charges  sq_footage  total_taxes

## Warning: Number of logged events: 175

colnames(housing)

##  [1] "approx_year_built"      "cats_allowed"           "common_charges"
##  [4] "coop_condo"             "dogs_allowed"           "fuel_type"
##  [7] "kitchen_type"           "maintenance_cost"       "num_bedrooms"
## [10] "num_floors_in_building" "num_full_bathrooms"     "num_total_rooms"
## [13] "parking_charges"        "sale_price"             "sq_footage"
## [16] "total_taxes"            "walk_score"             "town"
## [19] "med_income"             "pop_density"

#sum(is.na(housing_cols_removed$total_taxes))/nrow(housing_cols_removed)

housing_imp=complete(impute,1)
housing_imp$sale_price = y
housing_df= housing_imp %>%
    drop_na(sale_price)
train_indices = sample(1 : nrow(housing_df), round((1 - .2) * nrow(housing_df
)))
housing_train = housing_df[train_indices, ]
test_indices=setdiff(1 : nrow(housing_df), train_indices)

housing_test= housing_df[test_indices,]

Y_test = housing_test$sale_price
X_test = housing_test
```

```r
Y_train = housing_train$sale_price
X_train = housing_train
X_train$sale_price = NULL
n_train = nrow(X_train)


tree_mod=YARFCART(X_train, Y_train, calculate_oob_error = FALSE)
```

```
## YARF initializing with a fixed 1 trees...
## YARF factors created...
## YARF after data preprocessed... 38 total features...
## Beginning YARF regression model construction...done.
```

```r
#get training performance
y_hat_train = predict(tree_mod, housing_train)
```

```
## Warning in predict.YARF(tree_mod, housing_train): Prediction set column na
mes did not match training set column names.
## Attempting to subset to training set columns.
```

```r
e = Y_train - y_hat_train
tree_train_perf = 1 - sd(e) / sd(Y_train)
#test performance
y_hat_test = predict(tree_mod, housing_test)
```

```
## Warning in predict.YARF(tree_mod, housing_test): Prediction set column nam
es did not match training set column names.
## Attempting to subset to training set columns.
```

```r
e = Y_test - y_hat_test
tree_test_perf = 1 - sd(e) / sd(Y_test)
#linear train
linear_mod = lm(sale_price ~ ., housing_train)
y_hat_train_linear = predict(linear_mod, housing_train)
e = Y_train - y_hat_train_linear
linear_train_perf = 1 - sd(e) / sd(Y_train)
#linear test
y_hat_test_linear = predict(linear_mod, housing_test)
e = Y_test - y_hat_test_linear
linear_test_perf = 1 - sd(e) / sd(Y_test)
SSE = sum(linear_mod$residuals^2)
MSE <- SSE / length(linear_mod$residuals)
RMSE <- sqrt(MSE)
```

```r
paste0('tree in sample: ',tree_train_perf)
```

```
## [1] "tree in sample: 0.863321950183148"
```

```r
paste0('tree oos: ',tree_test_perf)
```

```
## [1] "tree oos: 0.495599997678257"
```

```r
paste0('linear in sample: ',linear_train_perf)

## [1] "linear in sample: 0.594198348862615"

paste0('linear oos: ',linear_test_perf)

## [1] "linear oos: 0.593888242765756"

paste0("lm RMSE: ", RMSE)

## [1] "lm RMSE: 71094.9678873884"

paste0("lm r.squared:", summary(linear_mod)$r.squared)

## [1] "lm r.squared:0.835325019934172"

summary(linear_mod)

##
## Call:
## lm(formula = sale_price ~ ., data = housing_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -300038  -40020   -1863   35142  307269
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)               -2.277e+05  7.521e+05  -0.303 0.762308
## approx_year_built          2.130e+02  3.739e+02   0.570 0.569363
## cats_allowedyes            1.476e+03  1.402e+04   0.105 0.916225
## common_charges             1.407e+01  2.976e+01   0.473 0.636763
## coop_condocondo            2.467e+05  2.011e+04  12.268  < 2e-16 ***
## dogs_allowedyes            1.302e+04  1.523e+04   0.855 0.393017
## fuel_typegas               7.538e+04  2.764e+04   2.727 0.006759 **
## kitchen_typeeatin         -1.456e+04  1.305e+04  -1.116 0.265368
## kitchen_typeefficiency    -2.024e+04  1.251e+04  -1.617 0.106827
## maintenance_cost           1.159e+02  2.822e+01   4.107 5.16e-05 ***
## num_bedrooms               5.453e+04  1.002e+04   5.445 1.07e-07 ***
## num_floors_in_building     2.215e+03  8.739e+02   2.535 0.011740 *
## num_full_bathrooms         8.411e+04  1.459e+04   5.763 2.02e-08 ***
## num_total_rooms            8.456e+03  6.784e+03   1.247 0.213520
## parking_charges           -9.751e+01  8.826e+01  -1.105 0.270101
## sq_footage                -2.355e+01  1.486e+01  -1.585 0.114098
## total_taxes               -1.958e-01  3.027e+00  -0.065 0.948458
## walk_score                -1.826e+02  5.134e+02  -0.356 0.722255
## townbayside               -1.021e+05  7.155e+04  -1.427 0.154652
## towncorona                -2.675e+05  7.792e+04  -3.434 0.000678 ***
## townflushing              -1.274e+05  3.983e+04  -3.200 0.001519 **
## townforest                -2.737e+04  5.761e+04  -0.475 0.635096
## townglen                  -7.091e+04  7.414e+04  -0.956 0.339598
## townhoward                -2.364e+05  5.707e+04  -4.142 4.45e-05 ***
```

```
## townjackson            -2.943e+04  4.209e+04  -0.699 0.484840
## townjamaica            -2.072e+05  5.489e+04  -3.776 0.000192 ***
## townkew                -1.544e+05  4.592e+04  -3.362 0.000873 ***
## townlittle             -9.024e+04  8.008e+04  -1.127 0.260668
## townoakland            -1.245e+05  6.728e+04  -1.851 0.065172 .
## townrego               -1.302e+05  4.111e+04  -3.166 0.001699 **
## townwhitestone         -1.224e+05  6.868e+04  -1.783 0.075626 .
## med_income             -1.978e+00  2.075e+00  -0.954 0.341017
## pop_density            -1.580e+04  4.710e+04  -0.336 0.737443
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 74830 on 306 degrees of freedom
## Multiple R-squared:  0.8353, Adjusted R-squared:  0.8181
## F-statistic: 48.51 on 32 and 306 DF,  p-value: < 2.2e-16

train_indices = sample(1 : nrow(housing_df), round((1 - .2) * nrow(housing_df
)))
test_indices=setdiff(1 : nrow(housing_df), train_indices)
housing_train = housing_df[train_indices,]

y=housing_train %>%
  dplyr::select(sale_price)

rf <- makeLearner("regr.randomForest", predict.type = "response", par.vals =
list(ntree = 200, mtry = 3))
modeling_task_train = makeRegrTask(data = housing_train, target = "sale_price
")

## Warning in makeTask(type = type, data = data, weights = weights, blocking
=
## blocking, : Empty factor levels were dropped for columns: town

modeling_task_test = makeRegrTask(data = housing_test, target = "sale_price")

## Warning in makeTask(type = type, data = data, weights = weights, blocking
=
## blocking, : Empty factor levels were dropped for columns: town

#instantiate the task
algorithm = makeLearner("regr.randomForest")
validation = makeResampleDesc("CV", iters = 5)
learner= makeLearner("regr.randomForest")
rf_param = makeParamSet(
makeIntegerParam("ntree",lower = 50, upper = 500),
makeIntegerParam("mtry", lower = 3, upper = 10),
makeIntegerParam("nodesize", lower = 10, upper = 50)
)
rancontrol = makeTuneControlRandom(maxit = 50L)
set_cv = makeResampleDesc("CV",iters = 3L)
```

```
rf_tune = tuneParams(learner = algorithm, resampling = set_cv, task = modelin
g_task_train, par.set = rf_param, control = rancontrol)

## [Tune] Started tuning learner regr.randomForest for parameter set:

##              Type len Def    Constr Req Tunable Trafo
## ntree    integer   -   - 50 to 500   -    TRUE     -
## mtry     integer   -   -   3 to 10   -    TRUE     -
## nodesize integer   -   -  10 to 50   -    TRUE     -

## With control class: TuneControlRandom

## Imputation value: Inf

## [Tune-x] 1: ntree=128; mtry=9; nodesize=29

## [Tune-y] 1: mse.test.mean=7749229657.7481327; time: 0.0 min

## [Tune-x] 2: ntree=91; mtry=3; nodesize=12

## [Tune-y] 2: mse.test.mean=7600745431.9054537; time: 0.0 min

## [Tune-x] 3: ntree=337; mtry=9; nodesize=13

## [Tune-y] 3: mse.test.mean=6664777976.1915483; time: 0.0 min

## [Tune-x] 4: ntree=164; mtry=10; nodesize=25

## [Tune-y] 4: mse.test.mean=7506188461.1436682; time: 0.0 min

## [Tune-x] 5: ntree=216; mtry=4; nodesize=22

## [Tune-y] 5: mse.test.mean=8017497816.2785578; time: 0.0 min

## [Tune-x] 6: ntree=177; mtry=8; nodesize=23

## [Tune-y] 6: mse.test.mean=7520331450.3883829; time: 0.0 min

## [Tune-x] 7: ntree=429; mtry=8; nodesize=37

## [Tune-y] 7: mse.test.mean=8337884232.0532780; time: 0.0 min

## [Tune-x] 8: ntree=223; mtry=5; nodesize=39

## [Tune-y] 8: mse.test.mean=8881813980.1585846; time: 0.0 min

## [Tune-x] 9: ntree=318; mtry=5; nodesize=40

## [Tune-y] 9: mse.test.mean=8743730708.1664810; time: 0.0 min

## [Tune-x] 10: ntree=465; mtry=3; nodesize=49

## [Tune-y] 10: mse.test.mean=10231063501.3951530; time: 0.0 min

## [Tune-x] 11: ntree=310; mtry=8; nodesize=31
```

```
## [Tune-y] 11: mse.test.mean=8010794247.8236704; time: 0.0 min
## [Tune-x] 12: ntree=113; mtry=7; nodesize=33
## [Tune-y] 12: mse.test.mean=8234353188.7563391; time: 0.0 min
## [Tune-x] 13: ntree=425; mtry=4; nodesize=26
## [Tune-y] 13: mse.test.mean=8241526517.1582489; time: 0.0 min
## [Tune-x] 14: ntree=119; mtry=5; nodesize=44
## [Tune-y] 14: mse.test.mean=9155793458.8954239; time: 0.0 min
## [Tune-x] 15: ntree=474; mtry=9; nodesize=30
## [Tune-y] 15: mse.test.mean=7871056999.5831127; time: 0.0 min
## [Tune-x] 16: ntree=383; mtry=5; nodesize=45
## [Tune-y] 16: mse.test.mean=9162921917.1973724; time: 0.0 min
## [Tune-x] 17: ntree=98; mtry=9; nodesize=18
## [Tune-y] 17: mse.test.mean=7104876676.8122301; time: 0.0 min
## [Tune-x] 18: ntree=162; mtry=3; nodesize=15
## [Tune-y] 18: mse.test.mean=7963562330.2995329; time: 0.0 min
## [Tune-x] 19: ntree=339; mtry=9; nodesize=49
## [Tune-y] 19: mse.test.mean=9136753717.5987949; time: 0.0 min
## [Tune-x] 20: ntree=260; mtry=3; nodesize=24
## [Tune-y] 20: mse.test.mean=8447849156.4050350; time: 0.0 min
## [Tune-x] 21: ntree=445; mtry=9; nodesize=41
## [Tune-y] 21: mse.test.mean=8645932527.9744568; time: 0.0 min
## [Tune-x] 22: ntree=395; mtry=5; nodesize=24
## [Tune-y] 22: mse.test.mean=7823224055.2223272; time: 0.0 min
## [Tune-x] 23: ntree=294; mtry=10; nodesize=42
## [Tune-y] 23: mse.test.mean=8710023385.0719490; time: 0.0 min
## [Tune-x] 24: ntree=436; mtry=3; nodesize=36
## [Tune-y] 24: mse.test.mean=9417094187.6445255; time: 0.0 min
## [Tune-x] 25: ntree=213; mtry=9; nodesize=38
```

```
## [Tune-y] 25: mse.test.mean=8484690579.3582964; time: 0.0 min
## [Tune-x] 26: ntree=378; mtry=4; nodesize=20
## [Tune-y] 26: mse.test.mean=7771374104.1363297; time: 0.0 min
## [Tune-x] 27: ntree=365; mtry=3; nodesize=19
## [Tune-y] 27: mse.test.mean=8148368520.4292431; time: 0.0 min
## [Tune-x] 28: ntree=216; mtry=10; nodesize=30
## [Tune-y] 28: mse.test.mean=7851329397.6048946; time: 0.0 min
## [Tune-x] 29: ntree=497; mtry=8; nodesize=30
## [Tune-y] 29: mse.test.mean=7883132992.6114616; time: 0.0 min
## [Tune-x] 30: ntree=160; mtry=5; nodesize=14
## [Tune-y] 30: mse.test.mean=7162962875.7875929; time: 0.0 min
## [Tune-x] 31: ntree=374; mtry=4; nodesize=35
## [Tune-y] 31: mse.test.mean=8877329258.2792606; time: 0.0 min
## [Tune-x] 32: ntree=305; mtry=3; nodesize=22
## [Tune-y] 32: mse.test.mean=8481418415.1297550; time: 0.0 min
## [Tune-x] 33: ntree=161; mtry=9; nodesize=19
## [Tune-y] 33: mse.test.mean=7144511191.6793413; time: 0.0 min
## [Tune-x] 34: ntree=464; mtry=6; nodesize=23
## [Tune-y] 34: mse.test.mean=7657371085.8276796; time: 0.0 min
## [Tune-x] 35: ntree=102; mtry=4; nodesize=46
## [Tune-y] 35: mse.test.mean=9495121616.4848099; time: 0.0 min
## [Tune-x] 36: ntree=421; mtry=7; nodesize=34
## [Tune-y] 36: mse.test.mean=8250192634.4319096; time: 0.0 min
## [Tune-x] 37: ntree=130; mtry=6; nodesize=27
## [Tune-y] 37: mse.test.mean=7827586017.0716200; time: 0.0 min
## [Tune-x] 38: ntree=424; mtry=4; nodesize=12
## [Tune-y] 38: mse.test.mean=7166332256.5014849; time: 0.0 min
## [Tune-x] 39: ntree=385; mtry=6; nodesize=22
```

```
## [Tune-y] 39: mse.test.mean=7495472215.1052637; time: 0.0 min

## [Tune-x] 40: ntree=97; mtry=4; nodesize=31

## [Tune-y] 40: mse.test.mean=8625350472.4815712; time: 0.0 min

## [Tune-x] 41: ntree=284; mtry=3; nodesize=13

## [Tune-y] 41: mse.test.mean=7645052555.3795910; time: 0.0 min

## [Tune-x] 42: ntree=181; mtry=8; nodesize=15

## [Tune-y] 42: mse.test.mean=6727845551.9023275; time: 0.0 min

## [Tune-x] 43: ntree=487; mtry=7; nodesize=14

## [Tune-y] 43: mse.test.mean=6946887359.0446835; time: 0.0 min

## [Tune-x] 44: ntree=445; mtry=5; nodesize=33

## [Tune-y] 44: mse.test.mean=8377638581.1030493; time: 0.0 min

## [Tune-x] 45: ntree=280; mtry=10; nodesize=45

## [Tune-y] 45: mse.test.mean=8910260251.9687214; time: 0.0 min

## [Tune-x] 46: ntree=105; mtry=6; nodesize=38

## [Tune-y] 46: mse.test.mean=8745470813.6658974; time: 0.0 min

## [Tune-x] 47: ntree=458; mtry=9; nodesize=37

## [Tune-y] 47: mse.test.mean=8281123488.5573654; time: 0.0 min

## [Tune-x] 48: ntree=180; mtry=10; nodesize=29

## [Tune-y] 48: mse.test.mean=7777915296.0087576; time: 0.0 min

## [Tune-x] 49: ntree=280; mtry=4; nodesize=44

## [Tune-y] 49: mse.test.mean=9499838644.6054516; time: 0.0 min

## [Tune-x] 50: ntree=329; mtry=4; nodesize=24

## [Tune-y] 50: mse.test.mean=8161302303.4739180; time: 0.0 min

## [Tune] Result: ntree=337; mtry=9; nodesize=13 : mse.test.mean=6664777976.1
915483
```

```
sqrt(rf_tune$y)
```

```
## mse.test.mean
##      81638.09
```

```
rf.tree <- setHyperPars(rf, par.vals = rf_tune$x)
```

```
#train a model
makeatree = makeLearner("regr.randomForest", predict.type = "response")
rforest = train(rf.tree, modeling_task_train)

#make predictions
rfmodel <- predict(rforest, modeling_task_test)

#submission file
RMSE= sqrt(sum((rfmodel$data$truth-rfmodel$data$response)^2)/length(rfmodel$d
ata$response))
Rsqred = 1 - (sum((rfmodel$data$truth-rfmodel$data$response)^2)/sum((rfmodel$
data$truth-mean(rfmodel$data$response))^2))
paste0('RMSE: ',RMSE)

## [1] "RMSE: 52795.0071359748"

paste0('R.squared: ',Rsqred)

## [1] "R.squared: 0.907785473092444"
```