```r
# Alternate Class Project
#
# (c) Copyright 2019 - AMULET Analytics
# ---------------------------------------------------------------

install.packages("tidyverse")
library(tidyverse)

install.packages("reshape2")
library(reshape2)

# ---------------------------------------------------------------
# DATA ACCESS
# ---------------------------------------------------------------

housing = read.csv('housing.csv')
dim(housing)                          # 20640x10

head(housing)

# Note: 207 NAs found for total_bedrooms variable
summary(housing)

# Check levels for factor variable ocean_proximity
levels(housing$ocean_proximity)   # 5 levels found
# [1] "<1H OCEAN"  "INLAND"     "ISLAND"     "NEAR BAY"   "NEAR OCEAN"

colnames(housing)    # Show variables list

# ---------------------------------------------------------------
# EXPLORATORY DATA ANALYSIS (EDA)
# ---------------------------------------------------------------

#par(mfrow=c(2,5))

# Plot distributions for each variable
ggplot(data = melt(housing), mapping = aes(x = value)) +
  geom_histogram(bins = 30) + facet_wrap(~variable, scales = 'free_x')


# ---------------------------------------------------------------
# DATA TRANSFORMATION
# ---------------------------------------------------------------

# Impute missing values for total_bedrooms variable (only varialbe
# with NAs) with median instead of mean since it is less influenced
# by extreme outliers.
housing$total_bedrooms[is.na(housing$total_bedrooms)] =
  median(housing$total_bedrooms , na.rm = TRUE)

# Now fix up total variables and make them means
housing$mean_bedrooms = housing$total_bedrooms/housing$households
housing$mean_rooms = housing$total_rooms/housing$households

# Remove total_bedrooms, and total_rooms
housing <- subset(housing, select = -c(total_bedrooms, total_rooms ))

# Turn levels of ocean_proximity into binary category variables

# 1. Get a list of all the levels in the 'ocean_proximity' variable
# 2. Make a new empty dataframe of all 0s, where each level is its own variable
# 3. Use a for loop to populate the appropriate columns of the dataframe
# 4. Drop the original column from the dataframe.

categories = unique(housing$ocean_proximity)

cat_housing = data.frame(ocean_proximity = housing$ocean_proximity)

for(cat in categories){
  cat_housing[,cat] = rep(0, times= nrow(cat_housing))
}
head(cat_housing)

#
for(i in 1:length(cat_housing$ocean_proximity)){
  cat = as.character(cat_housing$ocean_proximity[i])
  cat_housing[,cat][i] = 1
}

head(cat_housing)

#
cat_columns = names(cat_housing)
keep_columns = cat_columns[cat_columns != 'ocean_proximity']
cat_housing = select(cat_housing,one_of(keep_columns))
```

```
tail(cat_housing)

# ------------------------------------------------------
# Another method for creating cat_houseing

# Set up matrix of zeroes
cat_housing <- data.frame(matrix(0,
                                 nrow = nrow(housing),
                                 ncol = length(unique(housing$ocean_proximity))))

# rename columns using factor levels in ocean_proximity
colnames(cat_housing) <- as.character(unique(housing$ocean_proximity))

# use sapply and ifelse to set value equal to one when the value in ocean_proximity is equal to the column name
cat_housing[] <- sapply(seq_along(cat_housing),
                        function(x) ifelse(names(cat_housing[x]) == as.character(housing$ocean_proximity),1,0))

tail(cat_housing)
# ------------------------------------------------------

colnames(housing)

# Remove ocean_proximity, and median_house_value
drops = c('ocean_proximity','median_house_value')
housing_num =  housing[ , !(names(housing) %in% drops)]

head(housing_num)


# Scale numerical variables
scaled_housing_num = scale(housing_num)

head(scaled_housing_num)

# Create cleaned data frame
cleaned_housing = cbind(cat_housing,
                        scaled_housing_num,
                        median_house_value=housing$median_house_value)
names(cleaned_housing)


# Create training and test sets
set.seed(314) # Set a random seed so that same sample can be reproduced in future runs

sample = sample.int(n = nrow(cleaned_housing), size = floor(.8*nrow(cleaned_housing)), replace = F)
train = cleaned_housing[sample, ] #just the samples
test  = cleaned_housing[-sample, ] #everything but the samples

head(train)

# Verify train and test sets size
nrow(train) + nrow(test) == nrow(cleaned_housing)


# -----------------------------------------------------------------
# MACHINE LEARNING
# -----------------------------------------------------------------

library('boot')

glm_house = glm(median_house_value~median_income+mean_rooms+population,
                data=cleaned_housing)
k_fold_cv_error = cv.glm(cleaned_housing , glm_house, K=5)

k_fold_cv_error$delta

glm_cv_rmse = sqrt(k_fold_cv_error$delta)[1]
glm_cv_rmse #off by about $83,000... it is a start

names(glm_house)

glm_house$coefficients

# Now try randomForest classifier

install.packages("randomForest")
library('randomForest')

names(train)

set.seed(1738)

train_y = train[,'median_house_value']
train_x = train[, names(train) !='median_house_value']
```

```
head(train_y)
head(train_x)

# Training the randomeForest algorithm may take several minutes.
# NOTE: we're using randomForest in a different way from class,
# we're not using a formula, but rather a data frame with the
# predictors, and another data frame containing just the response.
rf = randomForest(train_x, y = train_y ,
                        ntree = 500, importance = TRUE)

# Here is the method we saw in class:
#rf_model = randomForest(median_house_value~. ,
#              data = train, ntree =500, importance = TRUE)


# Model object components
names(rf)
# [1] "call"            "type"            "predicted"       "mse"
# [5] "rsq"             "oob.times"       "importance"      "importanceSD"
# [9] "localImportance" "proximity"       "ntree"           "mtry"
#[13] "forest"          "coefs"           "y"               "test"
#[17] "inbag"

# Higher number == more important predictor
rf$importance
#                       %IncMSE IncNodePurity
#NEAR BAY            470608423  1.381851e+12
#<1H OCEAN          1745109754  4.686542e+12
#INLAND             4188543365  3.118565e+13
#NEAR OCEAN          531771569  2.190025e+12
#ISLAND                1634216  6.907757e+10
#longitude          6750159588  2.515919e+13
#latitude           5491283654  2.213235e+13
#housing_median_age 1069742204  9.592682e+12
#population         1027892411  7.260313e+12
#households         1154234298  7.913642e+12
#median_income      8449660398  7.297434e+13
#mean_bedrooms       433299848  7.591618e+12
#mean_rooms         1925950188  2.196552e+13

# ----------------------------------------------------------------
# MODEL ACCURACY
# ----------------------------------------------------------------

# Calculate the out-of-bag (oob) error estimate. This is a special
# way to determine accuracy of randomForest models, unlike the
# misclassification rate examples we saw in class.

# Leaving out a data source forces OOB predictions
oob_prediction = predict(rf)

# Calculate MSE (same as rf$mse). So even using a random forest of
# only 1000 decision trees we are able to predict the median price
# of a house in a given district to within $49,000 of the actual
# median house price. This can serve as our bechmark moving forward
# and trying other models.
train_mse = mean(as.numeric((oob_prediction - train_y)^2))
oob_rmse = sqrt(train_mse)
oob_rmse
#[1] 48742.7

# Now use the test set on the trained model. Our model scored
# roughly the same on the training and testing data, suggesting
# that it is not overfit and that it makes good predictions.
test_y = test[,'median_house_value']
test_x = test[, names(test) !='median_house_value']

y_pred = predict(rf, test_x)
test_mse = mean(((y_pred - test_y)^2))
test_rmse = sqrt(test_mse)
test_rmse
#[1] 48570.9
```