data_processing

May 15, 2022

```
import pandas as pd
import numpy as np
import pathlib
import os
import warnings
warnings.filterwarnings('ignore')
```

```
# TODO: change the structure of the codes with __main__ and other functions
def main():

    pd.set_option('display.max_rows', None) # to display all rows

    # get the current working directory
    cw = pathlib.Path().absolute()

    # read the listing/review data
    df = pd.read_csv(os.path.join(cw, 'Data', 'Airbnb_Listings.csv'))
    # review = pd.read_csv(os.path.join(cw, 'Data', 'reviews.csv'))
    df.columns

    # drop the columns that are definitely unnecessary
    # remove the variables that are definitely not needed
    df.drop(columns=["listing_url", "experiences_offered", "thumbnail_url",
    "medium_url","picture_url", "xl_picture_url", "host_url", "host_name",
    "host_thumbnail_url", "host_picture_url", "neighbourhood_group_cleansed",
    "square_feet", "has_availability", "requires_license", "license",
    "jurisdiction_names", "is_business_travel_ready","scrape_id", "street",
    "city", "state", "smart_location", "country_code", "country",
    "calendar_updated", "calendar_last_scraped"], inplace=True)

    # call some data processing functions
    df = clean_variable(df)

    # check the pattern of the null variables
    df.isnull().sum()

    df = missing_var_manipulation(df)
```

```
    df = create_new_var(df)

    # df.isnull().sum()

    # keep the listings with valid host_since date
    df = df[~df.host_since.isna()]

    # TODO: decide what are the variables I want to keep and make sure they are␣
 ↪all clean
    df.to_csv(os.path.join(cw, 'list_processed.csv'), index=False)

def clean_variable(df):
    '''this function mainly cleans the existing values for the variables,
    such as converting data types, recoding variables
    '''

    # tranform all the currency from text to numeric (using apply)
    var_currency = ["price", "weekly_price", "monthly_price",
    "security_deposit", "cleaning_fee", "extra_people"]
    df[var_currency] = df[var_currency].apply(lambda x: x.str.replace('[$,]',␣
 ↪'').astype(float))
    print(df[var_currency].head())

    # transform all the date variables from onject to date type
    var_date = ["host_since", "first_review", "last_review", "last_scraped"]
    df[var_date] = df[var_date].apply(lambda x: pd.to_datetime(x))

    # transform all the rate variables from object to float
    var_rate = ["host_response_rate", "host_acceptance_rate"]
    df[var_rate] = df[var_rate].apply(lambda x: x.str.replace('%', '').
 ↪astype(float)/100)
    print(df[var_rate].head())

    # clean the long string/text - e.g. remove some unneeded characters
    text_var = ["amenities", "host_verifications"]
    # remove the characters {, }, ", '
    df[text_var] = df[text_var].apply(lambda x: x.str.replace("[{\"'}]", ""))
    # for the host_verification variable, also remove the brackets - []
    df["host_verifications"] = df.host_verifications.str.replace("[\[\]]", "")

    # extract/process the zipcodes
    # special record
    df.loc[df.zipcode=="9503\n9503\n95035", "zipcode"] = "95035"
    # modify the values starting with 'CA '
    df['zipcode'] = df.zipcode.str.replace("CA ", "")
    # remove the zip-extension
    df["zipcode"] = df.zipcode.str.split("-", expand=True)[0]
```

```
    # change the zipcodes with length < 5 to NA
    df.loc[df.zipcode.str.len()!=5, 'zipcode'] = np.nan
    print(df.zipcode.unique()) # there are some missing zipcodes

    # make all the indicator/boolean variable from t/f to Y/N
    boolean_var = ["host_is_superhost", "host_has_profile_pic",
    "host_identity_verified", "is_location_exact", "instant_bookable",
    "require_guest_profile_picture", "require_guest_phone_verification"]
    df[boolean_var] = df[boolean_var].apply(lambda x: np.where(x=="t", "Y",␣
 ↪"N"))

    return df

def missing_var_manipulation(df):
    '''impute some missing values'''

    # deal with some missing (numeric) data
    numric_missing_var = ["reviews_per_month", "bedrooms", "bathrooms"]
    for var in numric_missing_var:
        df.loc[df[var].isna(), var] = 0

    df.loc[df.host_response_time.isna(), "host_response_time"] = "Unknown"

    return df

def create_new_var(df):
    '''create some new variables that may potentially be useful for the␣
 ↪analysis'''

    # bathroom per capita #NOTE: these two variables with most valid data
    df["bath_per_cap"] = df.bathrooms/df.accommodates

    # not sure how the security deposit and cleaning fee is measured (per night?
 ↪), so create
    # indicator indicating whether requires security deposit and cleaning fee
    df["require_security_deposit"] = np.where(df.security_deposit>0, "Y", "N")
    df["require_cleaning_fee"] = np.where(df.cleaning_fee>0, "Y", "N")

    # some variables regarding review score
    # whether have overall score rating
    df["has_review_rating"] = np.where(df.review_scores_rating, "Y", "N")

    # whether got the highest score rating for each aspect (10)
    for var in ["review_scores_accuracy", "review_scores_cleanliness",
    "review_scores_checkin", "review_scores_communication",
    "review_scores_location", "review_scores_value"]:
```

```
        df[var.replace("review_scores_", "")+"_review10"] = np.
 ↪where(df[var]==10, "Y", "N")

    # list$price.per.guest = list$price/list$guests_included

    return df

if __name__ == "__main__":
    main()
```

data_analysis

2022-05-17

The whole analysis refers a lot of the knowledge from the Book "An Introduction to Statistical Learning", but codes are my own.

Other important references include:

http://www.sthda.com/english/articles/24-ggpubr-publication-ready-plots/76-add-p-values-and-significance-levels-to-ggplots/

https://www.datanovia.com/en/blog/beautiful-radar-chart-in-r-using-fmsb-and-ggplot-packages/

```
options(repos = c(CRAN = "http://cran.rstudio.com/")) # avoid error "error in contrib url repos source"
knitr::opts_chunk$set(
    echo = TRUE,
    message = FALSE,
    warning = FALSE
)
# First specify the packages of interest
packages <- c("data.table", "tidyverse", "ggplot2", "gridExtra",
              "ggcorrplot", "ggpubr", "tm", "SnowballC", "wordcloud",
              "stringr", "leaps", "glmnet", "randomForest", "gbm",
              "cluster", "fmsb", "reshape2", "purrr", "maps", "dplyr") # library(fmsb)-radar chart

## Now load or install&load all
package.check <- lapply(
  packages,
  FUN = function(x) {
    if (!require(x, character.only = TRUE)) {
      install.packages(x, dependencies = TRUE)
      library(x, character.only = TRUE)
    }
  }
)
```

```
## Loading required package: data.table
```

```
## Loading required package: tidyverse
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.6
## v forcats   1.0.1     v stringr   1.6.0
## v ggplot2   4.0.1     v tibble    3.3.1
## v lubridate 1.9.4     v tidyr     1.3.2
## v purrr     1.2.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::between()     masks data.table::between()
```

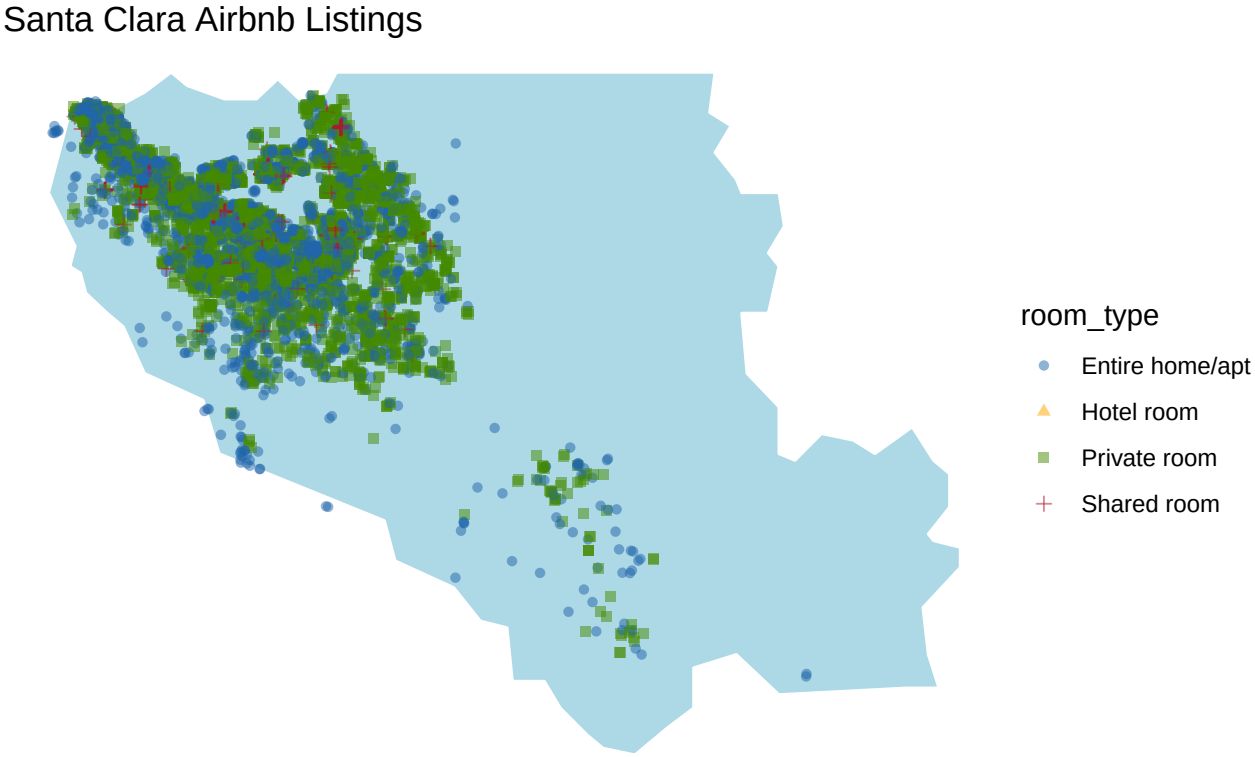
```
## x dplyr::filter()      masks stats::filter()
## x dplyr::first()       masks data.table::first()
## x lubridate::hour()    masks data.table::hour()
## x lubridate::isoweek() masks data.table::isoweek()
## x lubridate::isoyear() masks data.table::isoyear()
## x dplyr::lag()         masks stats::lag()
## x dplyr::last()        masks data.table::last()
## x lubridate::mday()    masks data.table::mday()
## x lubridate::minute()  masks data.table::minute()
## x lubridate::month()   masks data.table::month()
## x lubridate::quarter() masks data.table::quarter()
## x lubridate::second()  masks data.table::second()
## x purrr::transpose()   masks data.table::transpose()
## x lubridate::wday()    masks data.table::wday()
## x lubridate::week()    masks data.table::week()
## x lubridate::yday()    masks data.table::yday()
## x lubridate::year()    masks data.table::year()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
## Loading required package: gridExtra
##
##
## Attaching package: 'gridExtra'
##
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
##
## Loading required package: ggcorrplot
##
## Loading required package: ggpubr
##
## Loading required package: tm
##
## Loading required package: NLP
##
##
## Attaching package: 'NLP'
##
##
## The following object is masked from 'package:ggplot2':
##
##     annotate
##
##
## Loading required package: SnowballC
##
## Loading required package: wordcloud
##
## Loading required package: RColorBrewer
##
## Loading required package: leaps
##
```

```
## Loading required package: glmnet
##
## Loading required package: Matrix
##
##
## Attaching package: 'Matrix'
##
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
##
## Loaded glmnet 4.1-10
##
## Loading required package: randomForest
##
## randomForest 4.7-1.2
##
## Type rfNews() to see new features/changes/bug fixes.
##
##
## Attaching package: 'randomForest'
##
##
## The following object is masked from 'package:gridExtra':
##
##     combine
##
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
##
## The following object is masked from 'package:ggplot2':
##
##     margin
##
##
## Loading required package: gbm
##
## Loaded gbm 2.2.2
##
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.co
##
## Loading required package: cluster
##
## Loading required package: fmsb
##
## Loading required package: reshape2
##
##
## Attaching package: 'reshape2'
```

```
##
##
## The following object is masked from 'package:tidyr':
##
##     smiths
##
##
## The following objects are masked from 'package:data.table':
##
##     dcast, melt
##
##
## Loading required package: maps
##
##
## Attaching package: 'maps'
##
##
## The following object is masked from 'package:cluster':
##
##     votes.repub
##
##
## The following object is masked from 'package:purrr':
##
##     map
```

Part 1: Read the processed data and make a simple map with the density of listings

```
# read the data
# library(maps)
# library(dplyr)
df <- fread("list_processed.csv", header=TRUE, sep=",", quote="\"")

######### Exploratory analysis ##############
# 1) How the numeric data looks like among different neighbourhood/zip codes
# the most listings, the average other measurement (for clustering)

# make a general map showing the density
county <- map_data("county") %>%
  filter(region=="california" & subregion=="santa clara")

ggplot() +
  geom_polygon(data = county, aes(x = long, y = lat, group = group),
               col="white", fill="lightblue") +
  coord_quickmap()+
  geom_point(data = df, aes(x=longitude, y=latitude, color=room_type,
                            shape=room_type), alpha=1/2) +
  scale_color_manual(values = c("#2166ac", "orange", "chartreuse4", "#b2182b"))+
  theme_void()+ #"#fddbc7", "#d1e5f0"
  ggtitle("Santa Clara Airbnb Listings")
```



Part 2: Exploratory Data Analysis (EDA)

```
## a) world cloud to identify the potentially good feature that contributes to
# high price
# text: summary, amenities - word cloud
word.cloud <- function(input, exclude_word=",") {

  dt <- eval(input)
  names(dt) <- c("text.evaluate")

  text <- Corpus(VectorSource(tolower(dt$text.evaluate)))
  #inspect(text)

  exclude <- c(exclude_word, stopwords('english'))
  text <- tm_map(text, removeWords, exclude)
  # stopwords('english')
  text <- tm_map(text, removePunctuation)

  textTDM <- TermDocumentMatrix(text)
  # inspect(reasonsTDM)

  textMT <- as.matrix(textTDM)

  textMT <- sort(rowSums(textMT), decreasing = TRUE)
```

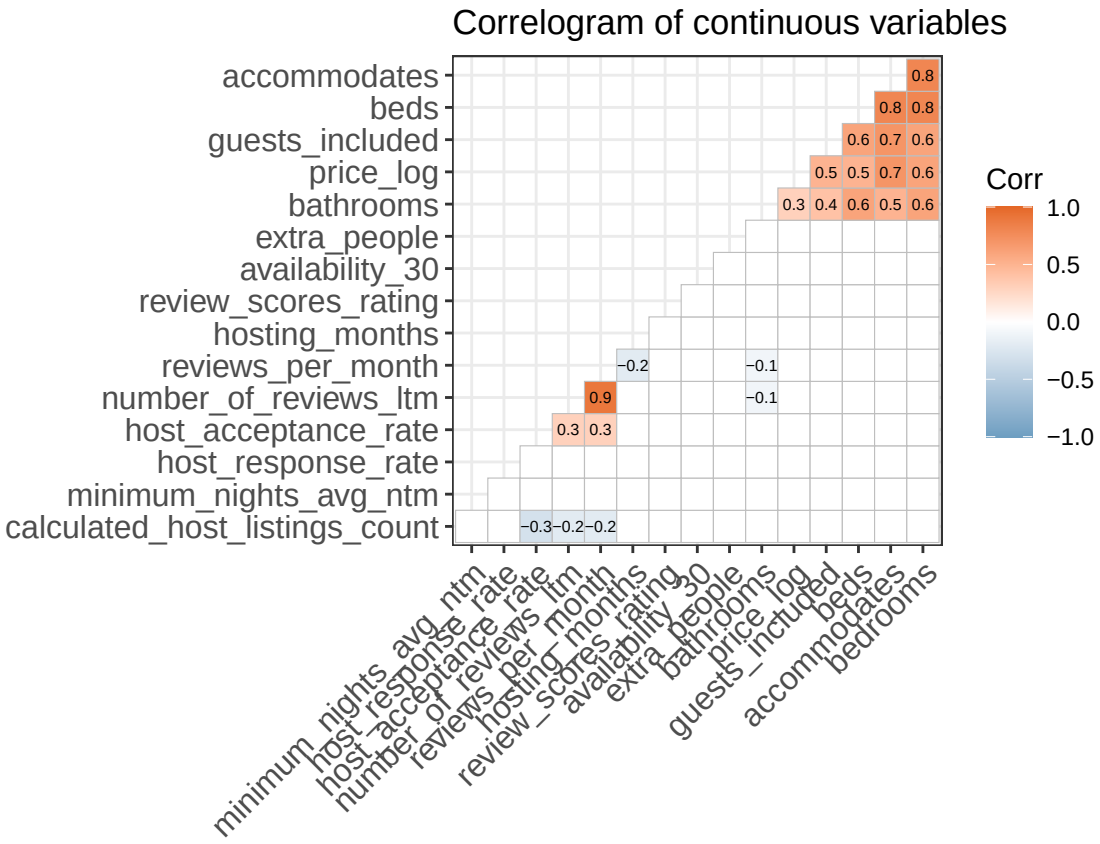
```
  # head(textMT)

  textDF <- data.frame(word = names(textMT), freq = textMT)
  # head(textDF)

  # Making the wordcloud
  pal <- brewer.pal(6,"Dark2")

  wordcloud(textDF$word, freq= textDF$freq, max.words = 100,
            random.order = FALSE, colors = pal)
}

# how the world cloud look like for price > median
input = df[df$price_log>median(df$price_log), "access"]
word.cloud(input, exclude_word = c(",", "access", "available","•"))
```


```
# how the world cloud look like for price <= median
input = df[df$price_log<=median(df$price_log), "access"]
word.cloud(input, exclude_word = c(",", "access", "available"))
```

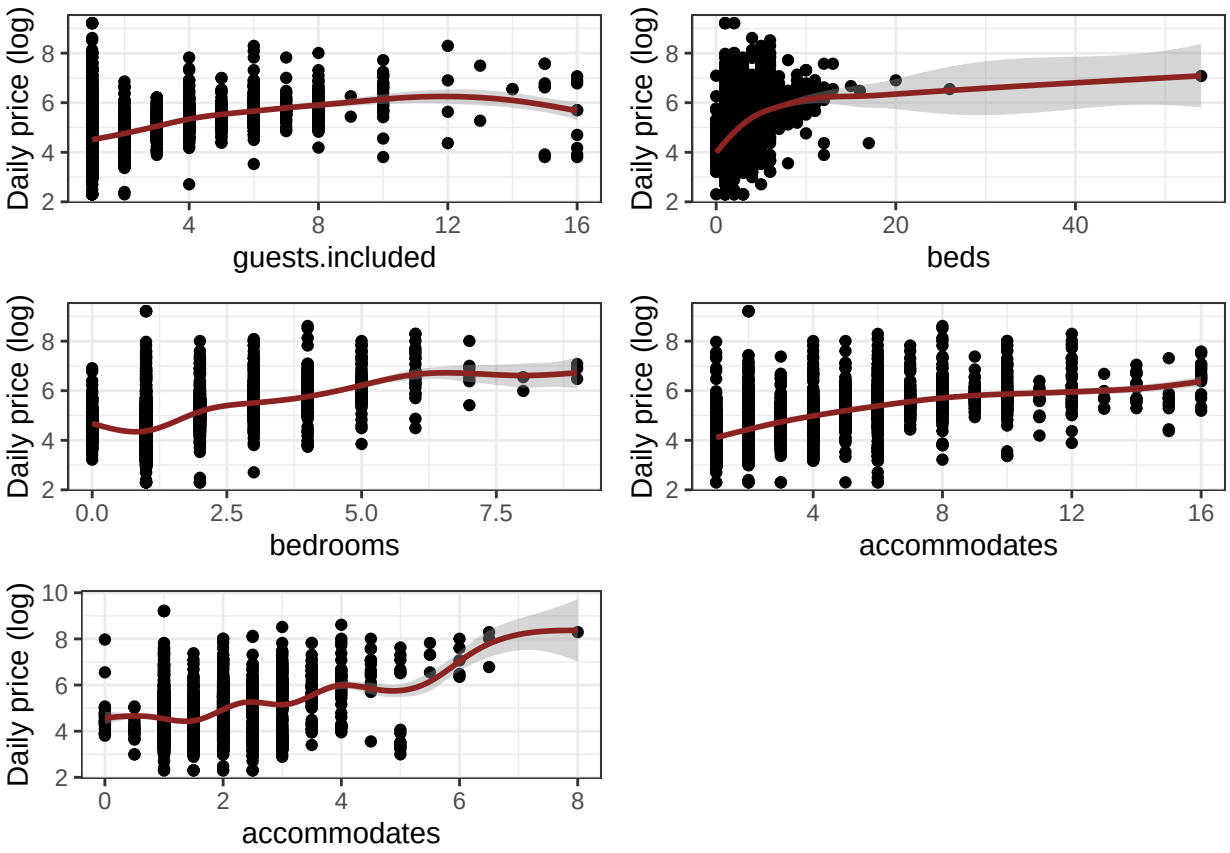

```
# create new var indicating whether access pool/garage
df <- df %>%
  mutate(pool_garage_access = ifelse(
    grepl("pool|garage",tolower(access)), "Y", "N"))
```

```
## b) The correlation matrix plot for the numeric variables
# visualization
df_con <- df %>%
  select(c(price_log, guests_included, host_response_rate, host_acceptance_rate,
           calculated_host_listings_count, accommodates, bathrooms,
           bedrooms, beds, extra_people, minimum_nights_avg_ntm,
           availability_30, number_of_reviews_ltm, review_scores_rating,
           reviews_per_month, hosting_months))

corr <- round(cor(df_con, use="complete.obs"), 1)
p.mat <- cor_pmat(corr) # used as parameter in the ggcorrplot
# Plot
ggcorrplot(corr, hc.order = TRUE,
           type = "lower",
           lab = TRUE,
           lab_size = 2,
           colors = c("#6D9EC1", "white", "#E46726"),
           insig = "blank", # this would ignore the cells with unsignificant
           p.mat = p.mat,
           sig.level = 0.1,
```

```
          title="Correlogram of continuous variables",
          ggtheme=theme_bw)
```

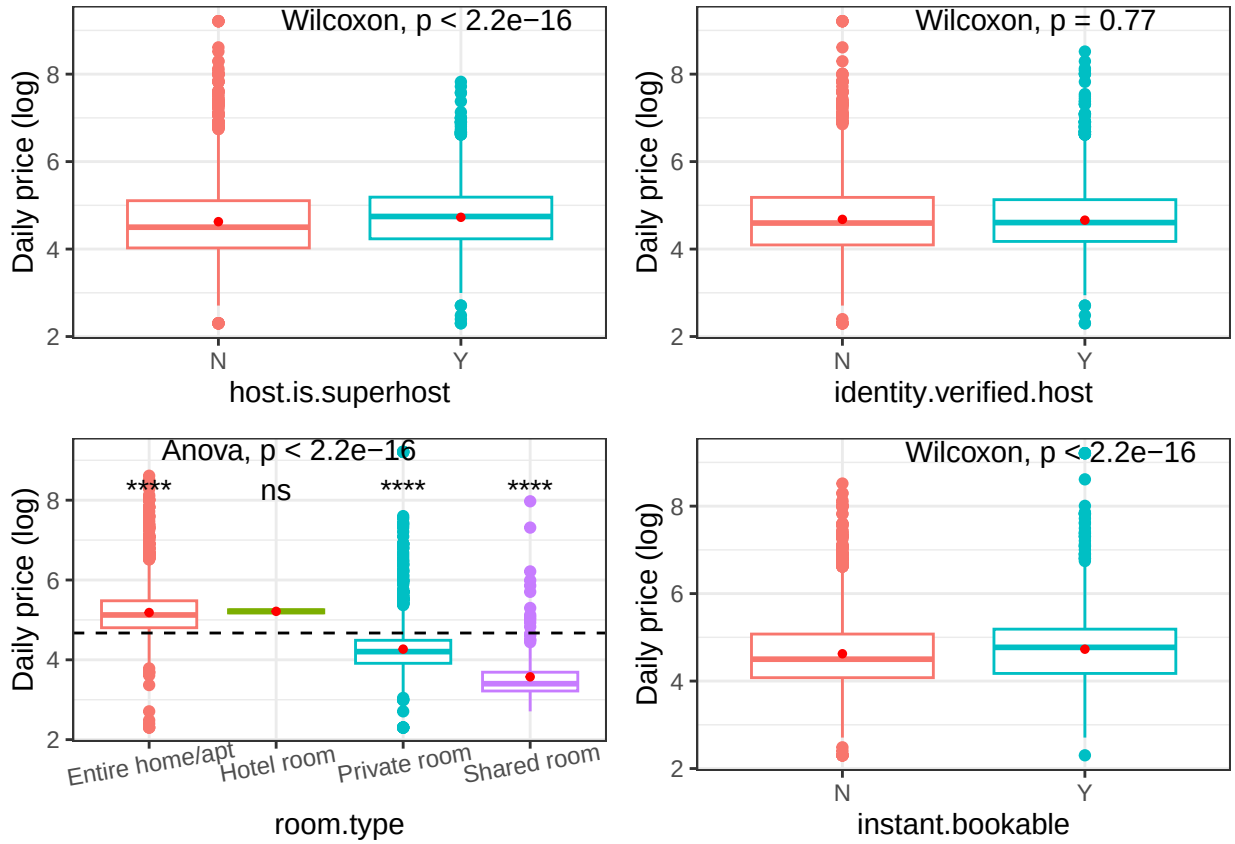



```
scatter.plot <- function(df, x, xlabel) {
  # some scatter plot

  df$x <- df[[x]]
  ggplot(df, aes(x=x, y=price_log)) +
    geom_point() + geom_smooth(color="brown4") + theme_bw()+
    labs(x=xlabel, y="Daily price (log)")
}

p1 <- scatter.plot(df, "guests_included", "guests.included")
p2 <- scatter.plot(df, "beds", "beds")
p3 <- scatter.plot(df, "bedrooms", "bedrooms")
p4 <- scatter.plot(df, "accommodates", "accommodates")
p5 <- scatter.plot(df, "bathrooms", "accommodates")

grid.arrange(p1, p2, p3, p4, p5, nrow = 3, ncol=2)
```

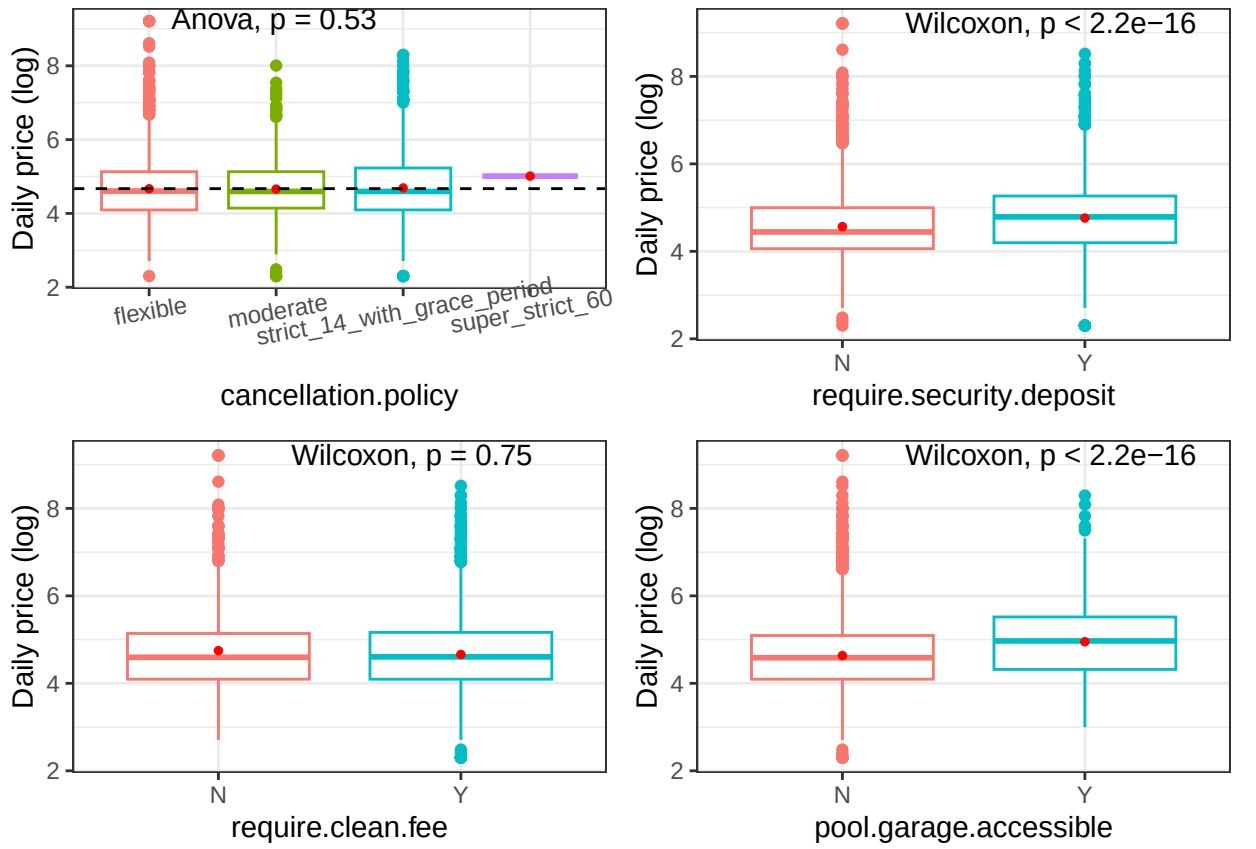



```
## c) the boxplot/ANOVA & T test for comparison among categorical variables
# https://data.library.virginia.edu/the-wilcoxon-rank-sum-test/

# categorical: host_is_superhost, host_identity_verified(ns), neighbourhood_cleansed, zipcode, property_
box.plot <- function(df, x, test.method="wilcox.test", xlabel,
                     xlabel.angle=0, test.label.position=1.5) {
  # wilcox.test/t.test
  df$x <- df[[x]]
  levels <- length(table(df$x))

  p <- df %>%
    ggplot(aes(x, price_log, color = x)) +
    geom_boxplot() + theme_bw()+
    # add the (group) mean point to the boxplot
    stat_summary(fun=mean, geom="point", color="red", size=1,
                 position = position_dodge(0.75), show.legend = FALSE) +
    stat_compare_means(method = test.method, label.y = 9,
                       label.x = test.label.position) +
    labs(y="Daily price (log)", x=xlabel) +
    theme(legend.position="none", # remove legend
          axis.text.x = element_text(angle = xlabel.angle))

  if (levels > 2) {
    p <- p +
      # add horizontal line of the overall average/mean
      geom_hline(yintercept = mean(df$price_log), linetype = 2) +
```

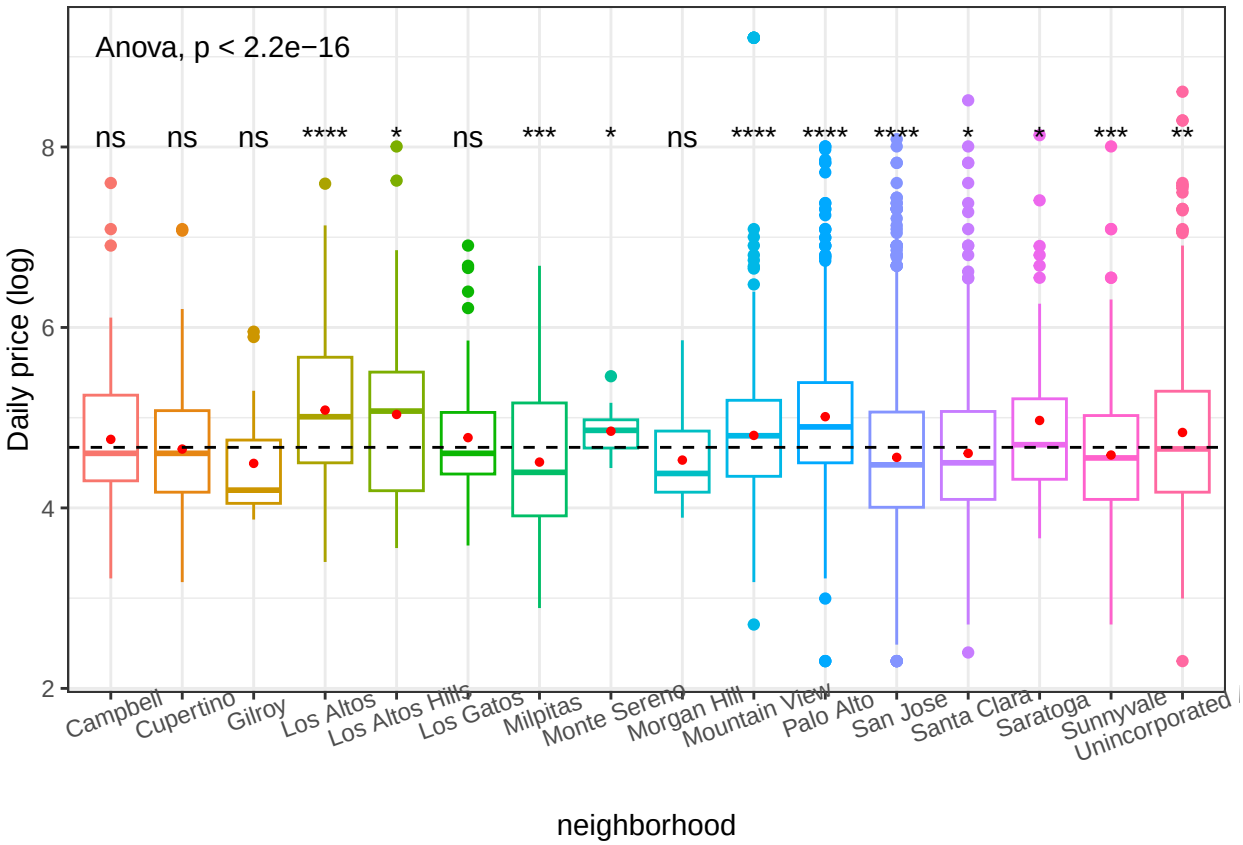
```
      # add signigicance level (comparing with overall mean)
      stat_compare_means(label = "p.signif", method = "t.test",
                         ref.group = ".all.", label.y=8)
  }

  return (p)
}

# make all the boxplots
p1 <- box.plot(df, "host_is_superhost", xlabel="host.is.superhost")
p2 <- box.plot(df, "host_identity_verified", xlabel="identity.verified.host")
p3 <- box.plot(df, "room_type", xlabel="room.type",
         test.method = 'anova', xlabel.angle = 10)
p4 <- box.plot(df, "instant_bookable", xlabel="instant.bookable")
p5 <- box.plot(df, "cancellation_policy", xlabel="cancellation.policy",
         test.method = 'anova', xlabel.angle = 10)
p6 <- box.plot(df, "require_security_deposit", xlabel="require.security.deposit")
p7 <- box.plot(df, "require_cleaning_fee", xlabel="require.clean.fee")
p8 <- box.plot(df, "pool_garage_access", xlabel="pool.garage.accessible")
grid.arrange(p1, p2, p3, p4, nrow = 2, ncol=2)
```



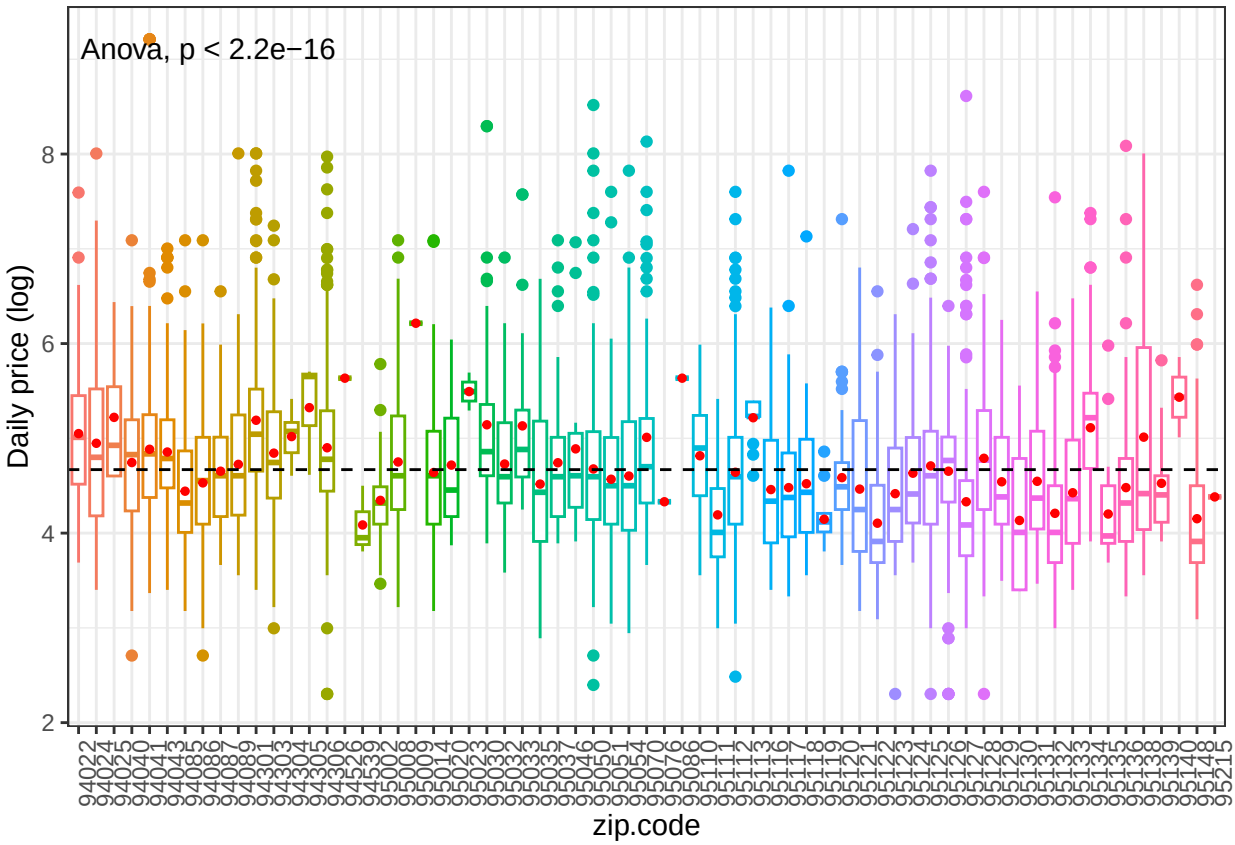
```
grid.arrange(p5, p6, p7, p8, nrow = 2, ncol=2)
```



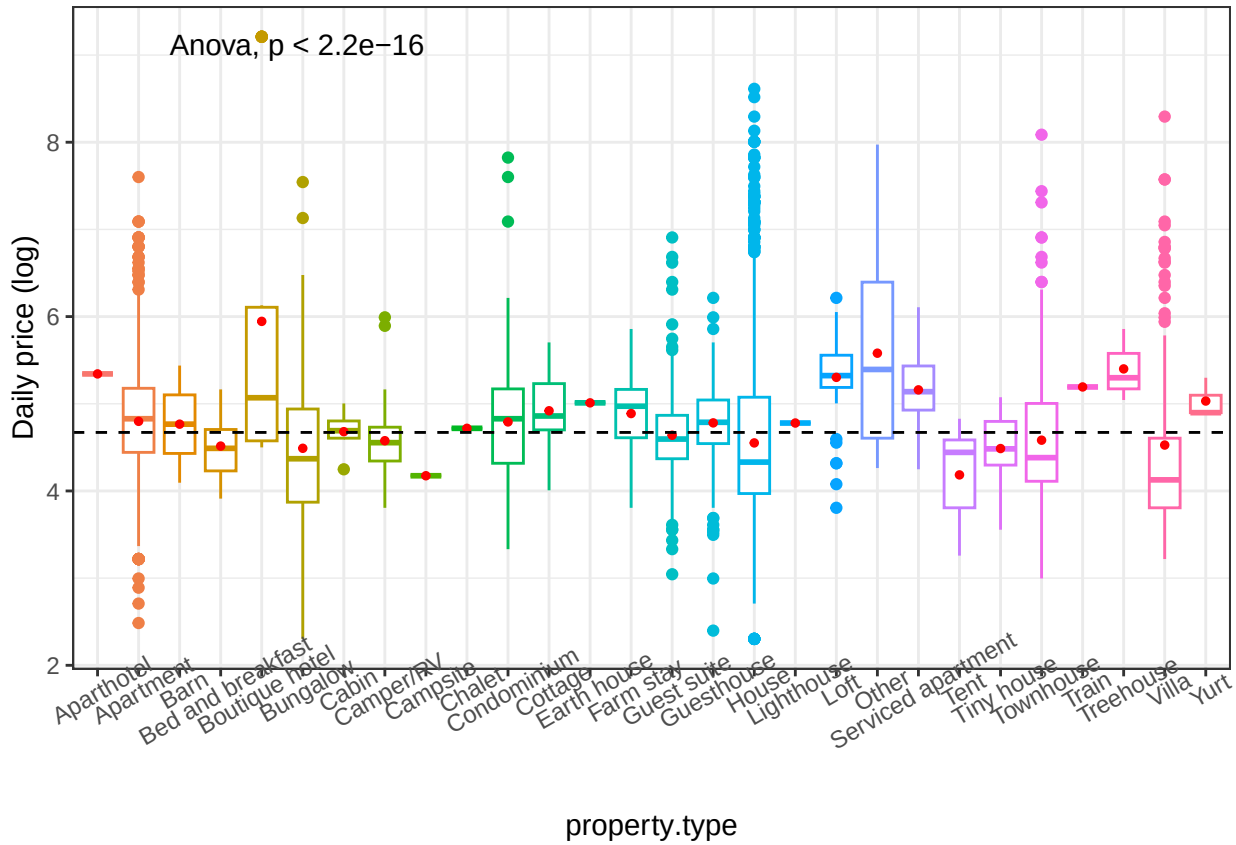

```
box.plot(df, "neighbourhood_cleansed", xlabel="neighborhood",
         test.method = 'anova', xlabel.angle=20)
```




```
df$zipcode <- as.factor(df$zipcode) # TODO: move somewhere else?
box.plot(df[df$zipcode!="NA",], "zipcode", xlabel="zip.code",
         test.method = 'anova', xlabel.angle=90,
         test.label.position=4)
```

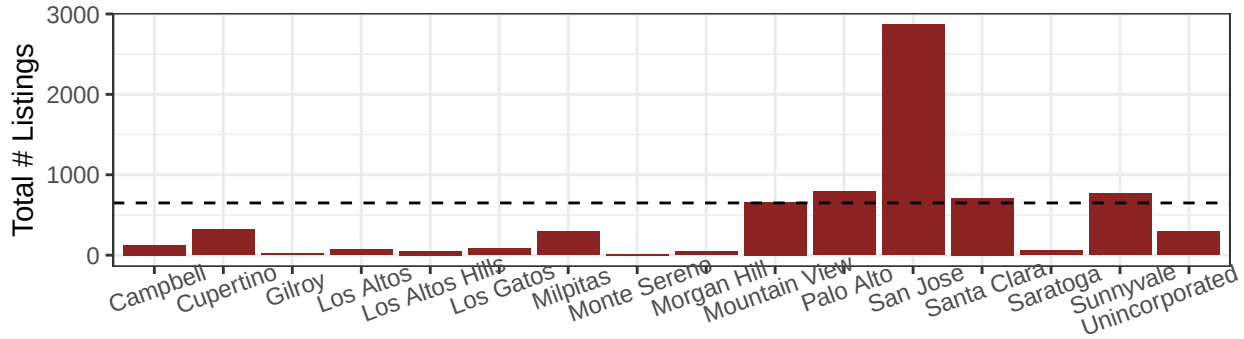



```
box.plot(df, "property_type", xlabel="property.type",
         test.method = 'anova', xlabel.angle=30,
         test.label.position=4)
```

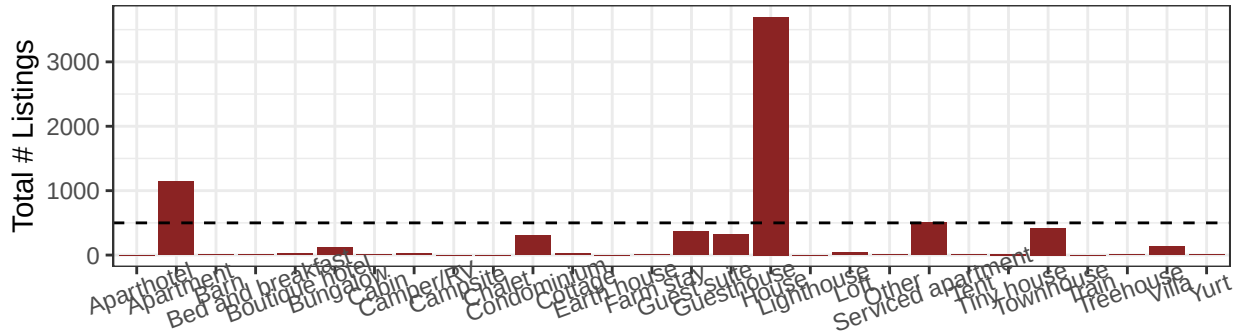



```
## d) make some bar plots of the number of listings within each
# neighbourhood and of each property type
bar.plot.freq <- function(data, groupvar, xlabel, ylabel,
                          yline=0, xlabel.angle=20) {
  data$group = data[[groupvar]]
  p <- data %>%
    group_by(group) %>%
    summarise(agg=n()) %>%
    ggplot(aes(y=agg, x=group)) +
    geom_bar(stat="identity", fill="brown4") + #position="stack",
    theme_bw()+
    geom_hline(yintercept = yline, linetype = 2)+
    labs(y=ylabel, x=xlabel)
  if (groupvar=="zipcode") {
    p <- p + theme(axis.text.x = element_blank(),
                   axis.title.x = element_blank(),
                   axis.title.y = element_blank()) +
      labs(subtitle = "Total # listings by zipcode")
  } else {
    p <- p+theme(axis.text.x = element_text(angle =xlabel.angle))
  }
  return (p)
}

p1 <- bar.plot.freq(df, groupvar="neighbourhood_cleansed", yline=650,
              xlabel="neighborhood", ylabel="Total # Listings")
```

```
p2 <- bar.plot.freq(df, groupvar="property_type", yline=500,
              xlabel="property.type", ylabel="Total # Listings")
grid.arrange(p1, p2, nrow = 2, ncol=1)
```



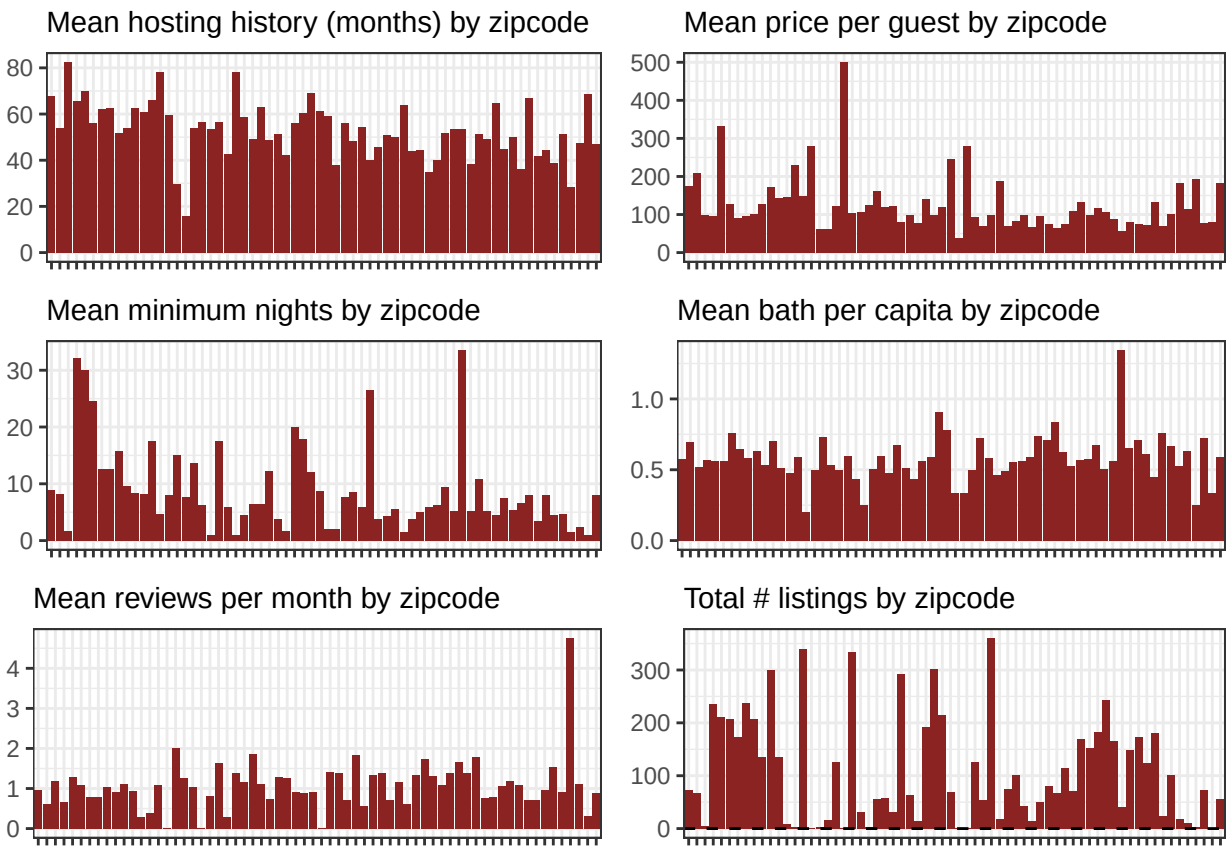
neighborhood



property.type

```
## e) make some bar plots of the average features within each zip
bar.plot.stat <- function(data, y, groupvar, stat="median", title) {
  data$group = data[[groupvar]]
  data$y = data[[y]]
  if (stat=="median") {
    data = data %>%
      group_by(group) %>%
      summarise(agg=median(y))
  } else if (stat=="mean") {
    data = data %>%
      group_by(group) %>%
      summarise(agg=mean(y))
  }

  data %>%
    ggplot(aes(y=agg, x=group)) +
    geom_bar(stat="identity", fill="brown4") +
    theme_bw()+
    labs(subtitle = title)+
    theme(axis.text.x = element_blank(),
          axis.title.x = element_blank(),
```

```
          axis.title.y = element_blank())
}

p1 <- bar.plot.stat(
  df, y="hosting_months", groupvar="zipcode", stat="mean",
  title= "Mean hosting history (months) by zipcode" )
p2 <- bar.plot.stat(
  df, y="price_per_guest", groupvar="zipcode", stat="mean",
  title= "Mean price per guest by zipcode" )
p3 <- bar.plot.stat(stat="mean",
  df, y="minimum_nights_avg_ntm", groupvar="zipcode",
  title= "Mean minimum nights by zipcode" )
p4 <- bar.plot.stat(
  df, y="bath_per_cap", stat="mean", groupvar="zipcode",
  title= "Mean bath per capita by zipcode" )
p5 <- bar.plot.stat(
  df, y="reviews_per_month", stat="mean", groupvar="zipcode",
  title= "Mean reviews per month by zipcode" )

# number of total listings by zip codes
p6 <- bar.plot.freq(df, groupvar="zipcode",
              xlabel="zipcode", xlabel.angle=90,
              ylabel="Total # Listings")
grid.arrange(p1, p2, p3, p4, p5, p6, nrow = 3, ncol=2)
```



Part 3: Regression Modeling

```
# categorical: host_is_superhost, host_identity_verified(ns), neighbourhood_cleansed, zipcode, property

# Change some variables--combine some categories
df$property.type <- df$property_type
df[!df$property_type %in% c("Apartment", "House"), "property.type"] <- "Other"


df$neighbourhood <- df$neighbourhood_cleansed
dense.area <- c("Santa Clara", "Palo Alto", "Mountain View",
                "San Jose", "Sunnyvale")
df[!df$neighbourhood_cleansed %in% dense.area, "neighbourhood"] <- "Other"
df[df$room_type %in% c("Hotel room", "Shared room"), "room_type"] <- "Other"

# create data for modeling
df.model <- df %>%
  select(c(price_log, guests_included, beds, accommodates, bedrooms, bathrooms,
           host_is_superhost, property.type, room_type, instant_bookable,
           require_security_deposit, pool_garage_access,
           neighbourhood)) %>%
  na.omit()
```

```
# a. linear regression
model <- lm(price_log~., data=df.model)

summary(model)
```

```
##
## Call:
## lm(formula = price_log ~ ., data = df.model)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9741 -0.2805 -0.0518  0.2097  4.7300
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  4.645587   0.028386 163.657  < 2e-16 ***
## guests_included              0.011365   0.004774   2.381 0.017308 *
## beds                        -0.014709   0.006292  -2.338 0.019424 *
## accommodates                 0.077112   0.005621  13.718  < 2e-16 ***
## bedrooms                     0.172248   0.011867  14.515  < 2e-16 ***
## bathrooms                    0.026600   0.011949   2.226 0.026036 *
## host_is_superhostY          -0.048086   0.012465  -3.858 0.000115 ***
## property.typeHouse          -0.065574   0.019443  -3.373 0.000748 ***
## property.typeOther           0.025799   0.019060   1.354 0.175910
## room_typeOther              -1.241812   0.032177 -38.593  < 2e-16 ***
## room_typePrivate room       -0.525607   0.017227 -30.510  < 2e-16 ***
## instant_bookableY           -0.000451   0.012177  -0.037 0.970456
## require_security_depositY -0.073112   0.012565  -5.819 6.18e-09 ***
## pool_garage_accessY          0.072508   0.019086   3.799 0.000146 ***
```

```
## neighbourhoodOther        -0.085828   0.024102  -3.561 0.000372 ***
## neighbourhoodPalo Alto     0.140745   0.026647   5.282 1.32e-07 ***
## neighbourhoodSan Jose     -0.197903   0.022055  -8.973  < 2e-16 ***
## neighbourhoodSanta Clara  -0.079441   0.027512  -2.887 0.003896 **
## neighbourhoodSunnyvale    -0.107054   0.026854  -3.986 6.77e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5003 on 7094 degrees of freedom
## Multiple R-squared:  0.5831, Adjusted R-squared:  0.582
## F-statistic: 551.2 on 18 and 7094 DF,  p-value: < 2.2e-16
```

```
k.fold <- 10

# randomly assign fold number
set.seed(10)
folds <- sample(1: k.fold, nrow(df.model), replace=T)

# initialize the cv error data table
cv.lm <- function(data, fold.k=k.fold) {
  test.error = 0
  test.error.org = 0
  for (k in 1:fold.k) {
    lm.fit = lm(price_log~., data=data[folds!=k, ])
    pred = predict(lm.fit, data[folds==k, ])
    test.error[k] = mean((data$price_log[folds==k]-pred)^2)
    test.error.org[k] = mean((exp(data$price_log[folds==k])-exp(pred))^2)
  }
  # return the mean test error
  return(c(mean(test.error), mean(test.error.org)))
}

cv.lm(df.model) # 0.2511718
```

```
## [1] 2.511718e-01 1.102059e+05
```

```
# b. ridge & lasso
# initialize the cv error data table
x <- model.matrix(price_log~., df.model)[,-1]
y <- df.model$price_log

cv.ridge.lasso <- function(x, y, alpha, seed, fold.k=10) {

  cv.out <- data.frame(k=1:fold.k, bestlam=0, error=0, error.org=0)

  for (k in 1:fold.k) {
    mod = glmnet(x[folds!=k, ], y[folds!=k], alpha=alpha)

    # use cv to find the best lambda
    set.seed(seed)
    cv.mod = cv.glmnet(x[folds!=k, ], y[folds!=k], alpha=alpha)
    cv.out[k, "bestlam"] = cv.mod$lambda.min
```

```
    pred = predict(mod, s=cv.mod$lambda.min, newx = x[folds==k,])
    cv.out[k, "error"] = mean((y[folds==k]-pred)^2)
    cv.out[k, "error.org"] = mean((exp(y[folds==k])-exp(pred))^2)
  }
  # return the mean test error
  return(cv.out)
}
cv.ridge <- cv.ridge.lasso (x, y, alpha=0, seed=1)
write.csv(cv.ridge,"cv_ridge.csv", row.names = FALSE)
c(mean(cv.ridge$error), mean(cv.ridge$error.org)) # 0.251999
```

```
## [1] 2.519990e-01 1.105573e+05
```

```
cv.lasso <- cv.ridge.lasso (x, y, alpha=1, seed=2)
write.csv(cv.lasso,"cv_lasso.csv", row.names = FALSE)
c(mean(cv.lasso$error), mean(cv.lasso$error.org)) # 0.2511531
```

```
## [1] 2.511531e-01 1.102583e+05
```

```
# check the coefficients from the lasso model
lasso.mod = glmnet(x, y, alpha=1)
lasso.coef <- predict(lasso.mod, type="coefficients",
                      s=mean(cv.lasso$bestlam))
lasso.coef
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                              s=0.0006169143
## (Intercept)                     4.63711967
## guests_included                 0.01111772
## beds                           -0.01242610
## accommodates                    0.07641555
## bedrooms                        0.17099742
## bathrooms                       0.02543213
## host_is_superhostY             -0.04634464
## property.typeHouse             -0.06733584
## property.typeOther              0.02278370
## room_typeOther                 -1.23925429
## room_typePrivate room          -0.52451087
## instant_bookableY               .
## require_security_depositY      -0.07097435
## pool_garage_accessY             0.07088259
## neighbourhoodOther             -0.07445506
## neighbourhoodPalo Alto          0.14883473
## neighbourhoodSan Jose          -0.18731243
## neighbourhoodSanta Clara       -0.06818132
## neighbourhoodSunnyvale         -0.09596037
```

```
########Decision Tree###########
for (var in c("host_is_superhost", "property.type", "room_type",
              "instant_bookable", "require_security_deposit",
              "pool_garage_access", "neighbourhood")){
```

```
  df.model[[var]] <- as.factor(df.model[[var]])
}

# c. random forest (& bagging)
# Tuning parameters
varnum <- length(colnames(df.model))-1
rf.tuning = expand.grid('mtry'=c(varnum/3, varnum),
                        n.trees=c(10,50,100,200,300,400,500),
                        'error_mse'=0, 'error_mae'=0, 'error_mse.org'=0, 'error_mae.org'=0,'error_medae

# Define the function to do cross validation
cv.rf = function(data, mtry., ntree., fold.k=10){
  test.error_mse = 0
  test.error_mae = 0
  test.error_mse.org = 0
  test.error_mae.org = 0 # mean absolute error in the original scale
  test.error_mape.org = 0
  test.error_medae.org = 0 # median absolute error in the original scale
  test.error_medape.org = 0

  for(k in 1:fold.k){
    cv.train = data[folds!=k,]
    cv.test = data[folds==k,]
    set.seed(100)
    rf = randomForest(price_log~., mtry=mtry., ntree=ntree., data=cv.train)
    pred = predict(rf, cv.test) # in log space
    ## transform back to the original scale - need some correction
    # adjustment: average of the exponentiated residuals - (1/n) × sum(exp(residual_i)
    adjustment = mean(exp(pred-cv.test$price_log))
    pred_original = exp(pred)*adjustment

    # calculate all different errors
    test.error_mse[k]= mean((pred-cv.test$price_log)^2)
    test.error_mae[k]= mean(abs((pred-cv.test$price_log)))
    test.error_mse.org[k] = mean((pred_original-exp(cv.test$price_log))^2)
    test.error_mae.org[k] = mean(abs(pred_original-exp(cv.test$price_log)))
    test.error_mape.org[k] = mean(abs((pred_original-exp(cv.test$price_log))/exp(cv.test$price_log)))
    test.error_medae.org[k] = median(abs(pred_original-exp(cv.test$price_log)))
    test.error_medape.org[k] = median(abs((pred_original-exp(cv.test$price_log))/exp(cv.test$price_log)

  }
  return(c(mean(test.error_mse), mean(test.error_mae), mean(test.error_mse.org), mean(test.error_mae.org
}

# # Try different tuning parameters
# for(i in 1:nrow(rf.tuning)){
#   result = cv.rf(df.model, mtry.=rf.tuning$mtry[i],
#                              ntree.=rf.tuning$n.trees[i])
#   rf.tuning$error_mse[i] = result[1]
#   rf.tuning$error_mae[i] = result[2]
#   rf.tuning$error_mse.org[i] = result[3]
#   rf.tuning$error_mae.org[i] = result[4]
#   rf.tuning$error_mape.org[i] = result[5]
```

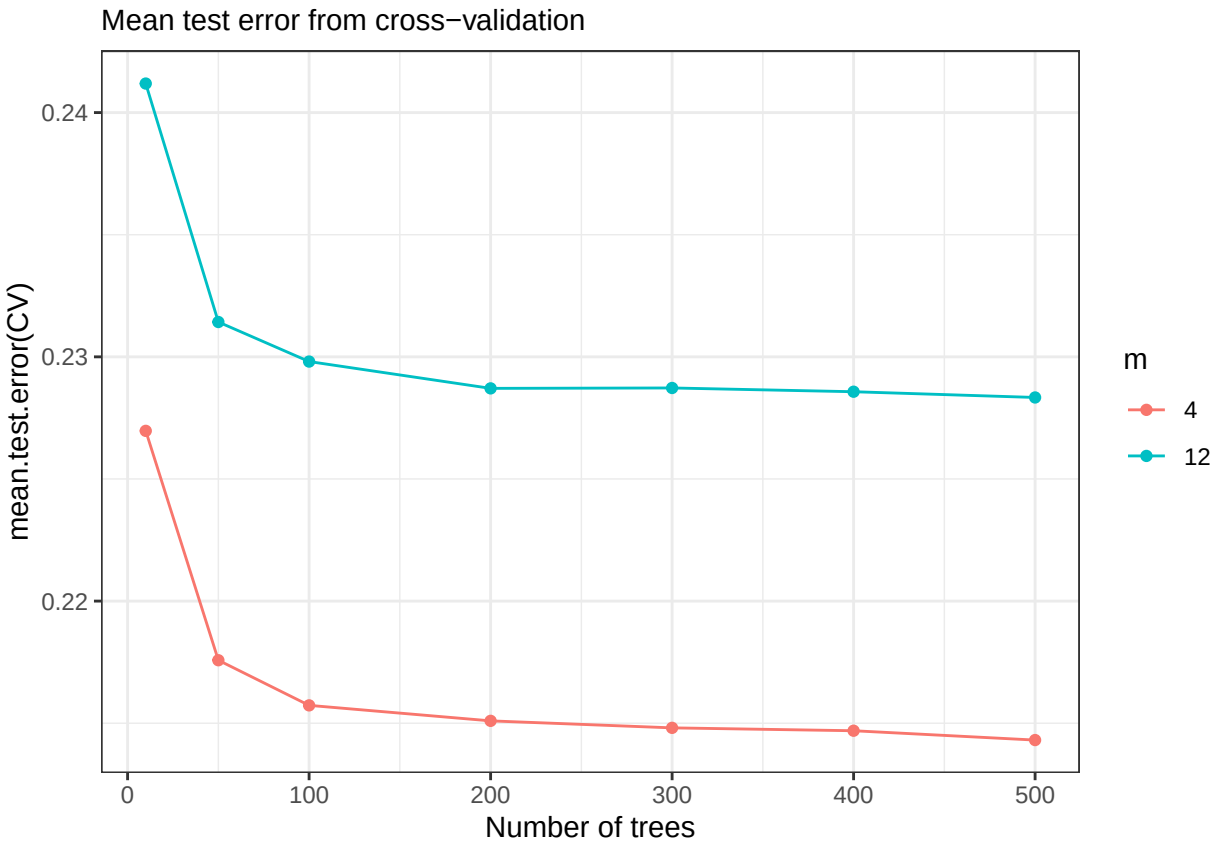
```
#   rf.tuning$error_medae.org[i] = result[6]
#   rf.tuning$error_medape.org[i] = result[7]
# }
# # User should uncomment out the for-loop (it takes some time to run)
# save(rf.tuning, file="rf_cv_result.rda")
# save(rf.tuning, file="rf_cv_result.RData")

load("rf_cv_result.rda")
# print the smallest error
rbind(
  rf.tuning[which.min(rf.tuning$error_mse),] %>% mutate(min='error_mse'),
  rf.tuning[which.min(rf.tuning$error_mae),] %>% mutate(min='error_mae'),
  rf.tuning[which.min(rf.tuning$error_mse.org),] %>% mutate(min='error_mse.org'),
  rf.tuning[which.min(rf.tuning$error_mae.org),] %>% mutate(min='error_mae.org'),
  rf.tuning[which.min(rf.tuning$error_mape.org),] %>% mutate(min='error_mape.org'),
  rf.tuning[which.min(rf.tuning$error_medae.org),] %>% mutate(min='error_medae.org'),
  rf.tuning[which.min(rf.tuning$error_medape.org),] %>% mutate(min='error_medape.org')
)
```

```
##     mtry n.trees error_mse error_mae error_mse.org error_mae.org
## 13     4     500 0.2143098 0.3066106      98356.18      64.80426
## 131    4     500 0.2143098 0.3066106      98356.18      64.80426
## 8     12     200 0.2287089 0.3174500      87915.46      68.26144
## 132    4     500 0.2143098 0.3066106      98356.18      64.80426
## 133    4     500 0.2143098 0.3066106      98356.18      64.80426
## 11     4     400 0.2146923 0.3068995      98696.42      64.91868
## 134    4     500 0.2143098 0.3066106      98356.18      64.80426
##     error_medae.org error_mape.org error_medape.org             min
## 13         24.59694      0.3748446        0.2516911       error_mse
## 131        24.59694      0.3748446        0.2516911       error_mae
## 8          26.52392      0.4138358        0.2616879   error_mse.org
## 132        24.59694      0.3748446        0.2516911   error_mae.org
## 133        24.59694      0.3748446        0.2516911  error_mape.org
## 11         24.58726      0.3755436        0.2524868 error_medae.org
## 134        24.59694      0.3748446        0.2516911 error_medape.org
```

```
print(summary(exp(df.model$price_log)))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    10.0    60.0   100.0   161.7   175.0 10000.0
```

```
###Make the plot of the test error under different pairs of parameters
rf.tuning %>%
  ggplot(aes(colour=factor(mtry), y=error_mse, x=n.trees)) +
  theme_bw()+
  # scale_x_continuous(breaks=seq(1,6,1)) +
  geom_line()+geom_point() +
  labs(y="mean.test.error(CV)", x="Number of trees",
       subtitle="Mean test error from cross-validation")+
  scale_color_discrete("m")
```

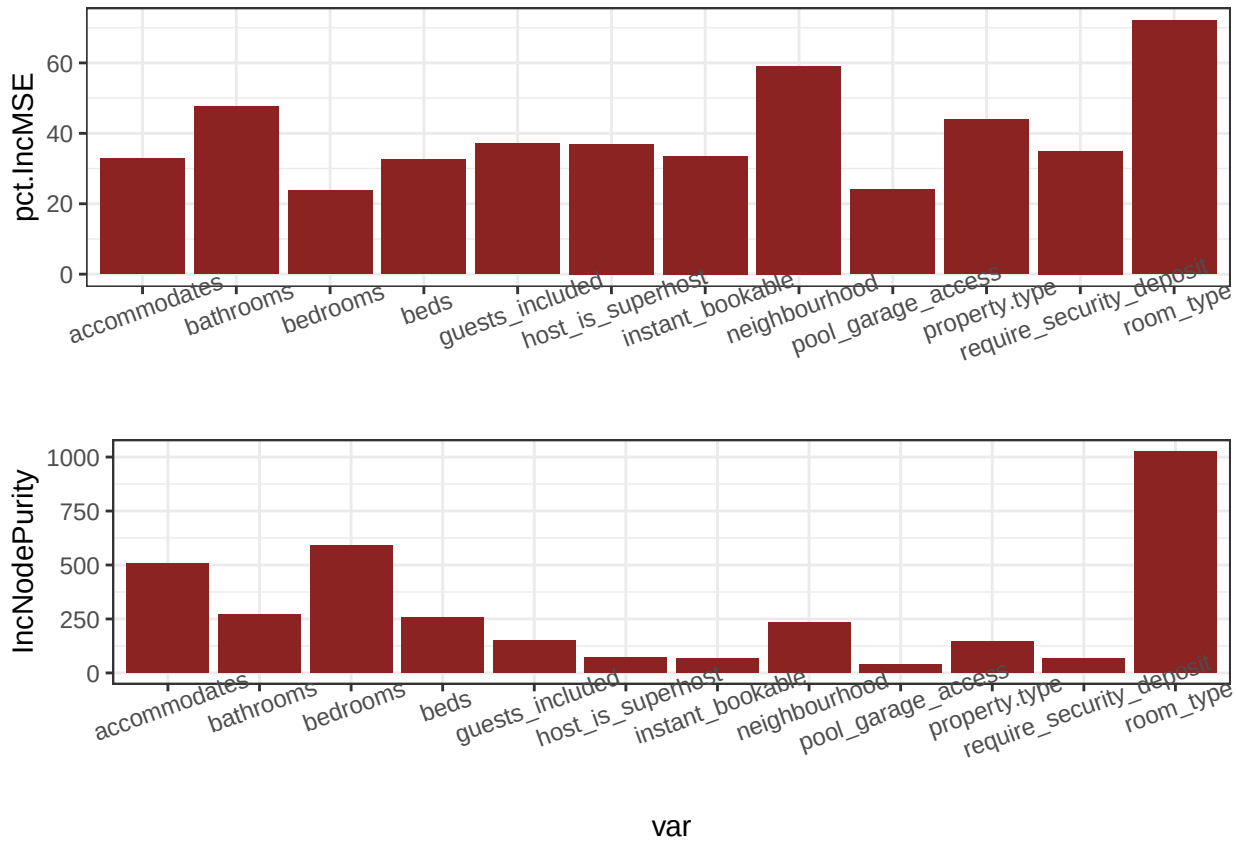



```
## Smallest error is 0.2160168 when mtry=4 and ntrees=300.

# Variable importance
set.seed(100)
rf = randomForest(price_log~., data=df.model,
                  importance=TRUE, ntree=300, mtry=4)
rf
```

```
##
## Call:
##  randomForest(formula = price_log ~ ., data = df.model, importance = TRUE,      ntree = 300, mtry = 4
##                Type of random forest: regression
##                      Number of trees: 300
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 0.2130696
##                     % Var explained: 64.41
```

```
# importance(rf)
# varImpPlot(rf)
# plot the two measurements of the variable importance
importance=data.frame(importance(rf))
importance$var = rownames(importance)
colnames(importance) <- c("pct.IncMSE", "IncNodePurity", "var")
```

```r
# dev.off()
# https://statisticsglobe.com/error-invalid-graphics-state-in-r
p1 <- importance %>%
  ggplot(aes(x=var, y=pct.IncMSE)) +
  geom_bar(stat="identity", fill="brown4") +
  theme_bw()+
  theme(axis.title.x = element_blank(),
        axis.text.x = element_text(angle = 20))
p2 <- importance %>%
  ggplot(aes(x=var, y=IncNodePurity)) +
  geom_bar(stat="identity", fill="brown4") +
  theme_bw()+
  theme(axis.text.x = element_text(angle = 20))
grid.arrange(p1, p2, nrow = 2, ncol=1)
```



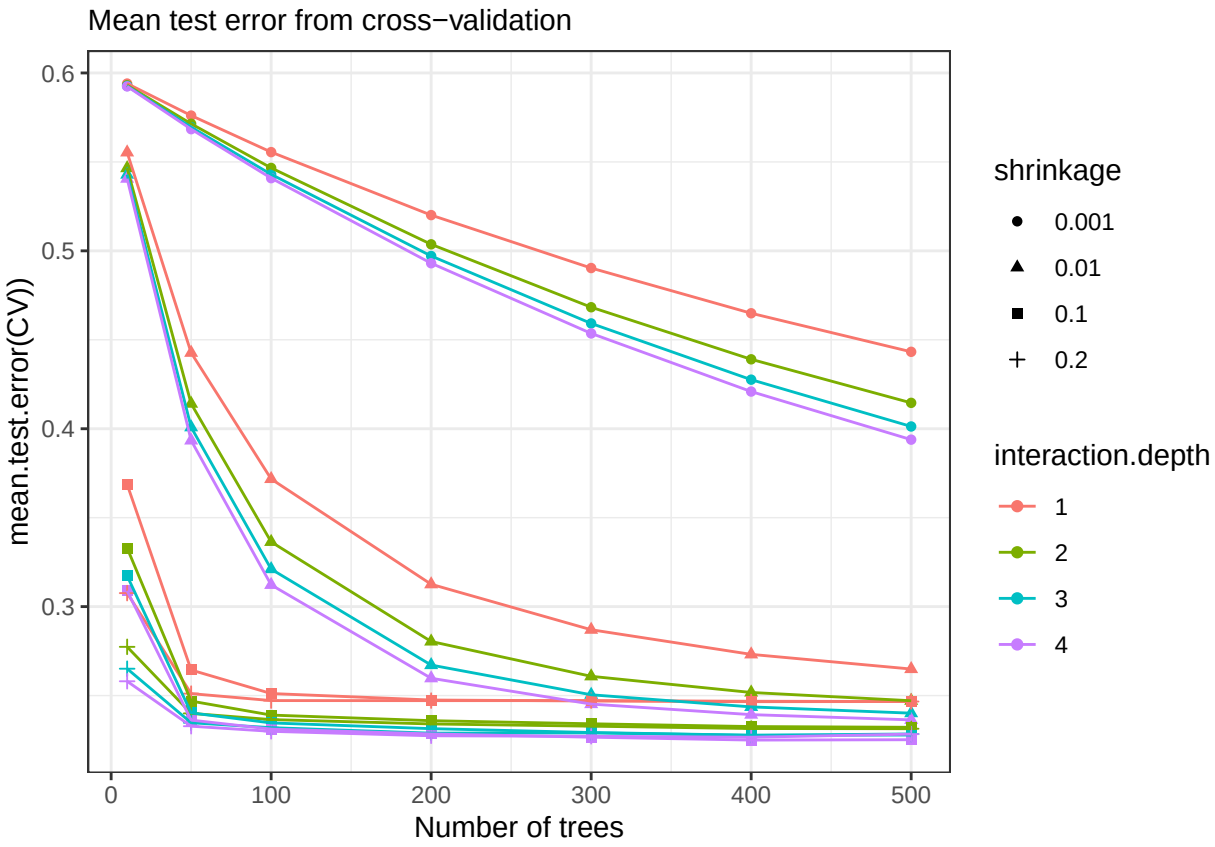

```r
# d. boosting
bst.tuning = expand.grid("n.trees"=c(10,50,100,200,300,400,500),
                         "interaction.depth"=c(1,2,3,4),
                         "shrinkage"=c(0.001,0.01,0.1,0.2),
                         'error_mse'=0, 'error_mae'=0, 'error_mse.org'=0, 'error_mae.org'=0,'error_meda

# Define a function to do crossvalidation for boosting
cv.bst = function(data, ntree, depth, lambda, fold.k=10){
  test.error_mse = 0
  test.error_mae = 0
```

```
  test.error_mse.org = 0
  test.error_mae.org = 0 # mean absolute error in the original scale
  test.error_mape.org = 0
  test.error_medae.org = 0 # median absolute error in the original scale
  test.error_medape.org = 0

  for(k in 1:fold.k){
    cv.train = data[folds!=k,]
    cv.test = data[folds==k,]
    set.seed(200)
    bst = gbm(price_log~., data=cv.train, distribution='gaussian',
              n.trees=ntree, interaction.depth=depth,shrinkage=lambda)
    pred = predict(bst, cv.test, n.trees=ntree)
    ## transform back to the original scale - need some correction
    # adjustment: average of the exponentiated residuals - (1/n) × sum(exp(residual_i)
    adjustment = mean(exp(pred-cv.test$price_log))
    pred_original = exp(pred)*adjustment

    # calculate all different errors
    test.error_mse[k]= mean((pred-cv.test$price_log)^2)
    test.error_mae[k]= mean(abs((pred-cv.test$price_log)))
    test.error_mse.org[k] = mean((pred_original-exp(cv.test$price_log))^2)
    test.error_mae.org[k] = mean(abs(pred_original-exp(cv.test$price_log)))
    test.error_mape.org[k] = mean(abs((pred_original-exp(cv.test$price_log))/exp(cv.test$price_log)))
    test.error_medae.org[k] = median(abs(pred_original-exp(cv.test$price_log)))
    test.error_medape.org[k] = median(abs((pred_original-exp(cv.test$price_log))/exp(cv.test$price_log)
  }
  return(c(mean(test.error_mse), mean(test.error_mae), mean(test.error_mse.org), mean(test.error_mae.org
}

# # Try different tuning parameters
# for(i in 1:nrow(bst.tuning)){
#   result = cv.bst(df.model, ntree=bst.tuning$n.trees[i],
#                   depth=bst.tuning$interaction.depth[i],
#                   lambda=bst.tuning$shrinkage[i])
#   bst.tuning$error_mse[i] = result[1]
#   bst.tuning$error_mae[i] = result[2]
#   bst.tuning$error_mse.org[i] = result[3]
#   bst.tuning$error_mae.org[i] = result[4]
#   bst.tuning$error_mape.org[i] = result[5]
#   bst.tuning$error_medae.org[i] = result[6]
#   bst.tuning$error_medape.org[i] = result[7]
# }
# # TODO: user needs to uncomment this out if want to run the for-loop
# save(bst.tuning, file="bst_cv_result.RData")

load("bst_cv_result.RData")
# print the smallest error
rbind(
  bst.tuning[which.min(bst.tuning$error_mse),] %>% mutate(min='error_mse'),
  bst.tuning[which.min(bst.tuning$error_mae),] %>% mutate(min='error_mae'),
  bst.tuning[which.min(bst.tuning$error_mse.org),] %>% mutate(min='error_mse.org'),
  bst.tuning[which.min(bst.tuning$error_mae.org),] %>% mutate(min='error_mae.org'),
```

```
  bst.tuning[which.min(bst.tuning$error_mape.org),] %>% mutate(min='error_mape.org'),
  bst.tuning[which.min(bst.tuning$error_medae.org),] %>% mutate(min='error_medae.org'),
  bst.tuning[which.min(bst.tuning$error_medape.org),] %>% mutate(min='error_medape.org')
)
```

```
##     n.trees interaction.depth shrinkage error_mse error_mae error_mse.org
## 83      400                 4       0.1 0.2249733 0.3206100      103600.4
## 831     400                 4       0.1 0.2249733 0.3206100      103600.4
## 111     400                 4       0.2 0.2265730 0.3238019      103462.7
## 832     400                 4       0.1 0.2249733 0.3206100      103600.4
## 84      500                 4       0.1 0.2251637 0.3209485      103689.4
## 841     500                 4       0.1 0.2251637 0.3209485      103689.4
## 842     500                 4       0.1 0.2251637 0.3209485      103689.4
##     error_mae.org error_medae.org error_mape.org error_medape.org
## 83       67.16627        26.67781      0.3904483        0.2734064
## 831      67.16627        26.67781      0.3904483        0.2734064
## 111      68.58861        27.24605      0.3983398        0.2765784
## 832      67.16627        26.67781      0.3904483        0.2734064
## 84       67.37876        26.15228      0.3897376        0.2731338
## 841      67.37876        26.15228      0.3897376        0.2731338
## 842      67.37876        26.15228      0.3897376        0.2731338
##                  min
## 83         error_mse
## 831        error_mae
## 111    error_mse.org
## 832    error_mae.org
## 84    error_mape.org
## 841  error_medae.org
## 842 error_medape.org
```

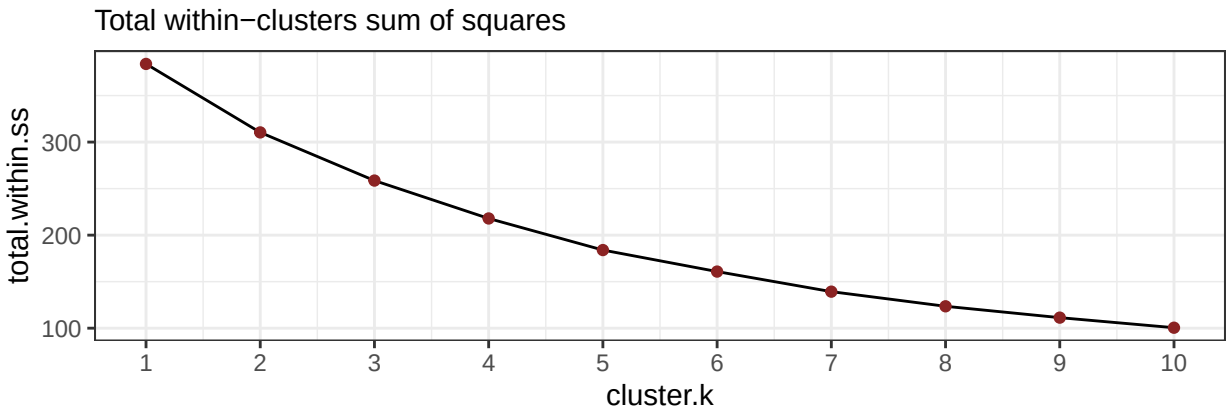
```
# plot the performance
bst.tuning %>%
  mutate(interaction.depth=factor(interaction.depth),
         shrinkage = factor(shrinkage)) %>%
  ggplot(aes(colour=interaction.depth, shape=shrinkage,
             y=error_mse, x=n.trees)) +
  theme_bw()+geom_point() + geom_line() +
  labs(y="mean.test.error(CV))", x="Number of trees",
       subtitle="Mean test error from cross-validation")
```

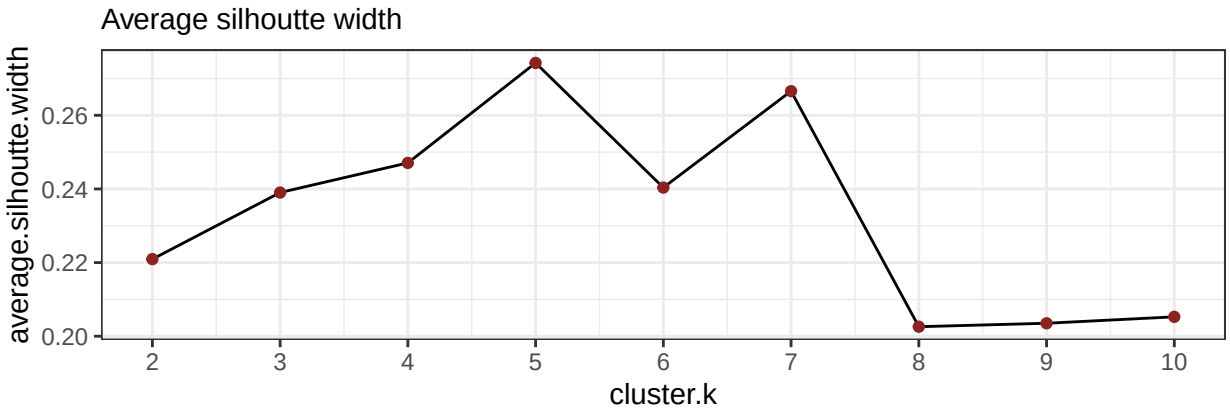


Part 4: Clustering

```
##############################
# numeric: host_response_rate, host_acceptance_rate, calculated_host_listings_count,
# accommodates, bathrooms, bedrooms, beds, extra_people, minimum_nights_avg_ntm,
# availability_30, number_of_reviews_ltm (may related to age),
# review_score_rating, reviews_per_month, bath_per_cap, hosting_months

# a) clustering the zip codes based on the average feature in that
# zip code
df.clust <- df %>%
  filter(zipcode != "") %>%
  group_by(zipcode) %>%
  summarise(minimum.nights.avg = mean(minimum_nights_avg_ntm),
         bath.per.cap.avg = mean(bath_per_cap),
         hosting.age.avg = mean(hosting_months),
         price.per.guest.avg = mean(price_per_guest),
         reviews.12.month.avg = mean(reviews_per_month),
         total.listings = n())

# scale the data
df.clust.scaled <- data.frame(scale(df.clust[, -1]))

# Elbow Method
```

```r
wss <- function(k) {
  kmeans(df.clust.scaled, k, nstart = 35 )$tot.withinss
}

# Silhouette method
avg_sil <- function(k) {
  km.res <- kmeans(df.clust.scaled, centers = k, nstart = 25)
  ss <- silhouette(km.res$cluster, dist(df.clust.scaled))
  return(mean(ss[, 3]))
}

# Compute and plot wss for k = 1 to k = 10
k.values <- 1:10
wss_values <- map_dbl(k.values, wss)
p1 <- data.frame("total.within.ss"=wss_values,
                 "cluster.k"=k.values) %>%
  ggplot(aes(y=total.within.ss, x=cluster.k)) +
  geom_line()  + geom_point(color="brown4")+
  theme_bw()+
  scale_x_continuous(breaks=seq(0,10,by=1)) +
  labs(subtitle = "Total within-clusters sum of squares")

set.seed(123) # this seed is important
k.values <- 2:10
avg_sil_values  <- map_dbl(k.values, avg_sil)
p2 <- data.frame("average.silhoutte.width"=avg_sil_values,
                 "cluster.k"=k.values) %>%
  ggplot(aes(y=average.silhoutte.width, x=cluster.k)) +
  geom_line()  + geom_point(color="brown4")+
  theme_bw()+
  scale_x_continuous(breaks=seq(0,10,by=1)) +
  labs(subtitle = "Average silhoutte width")
grid.arrange(p1, p2, nrow = 2, ncol=1)
```



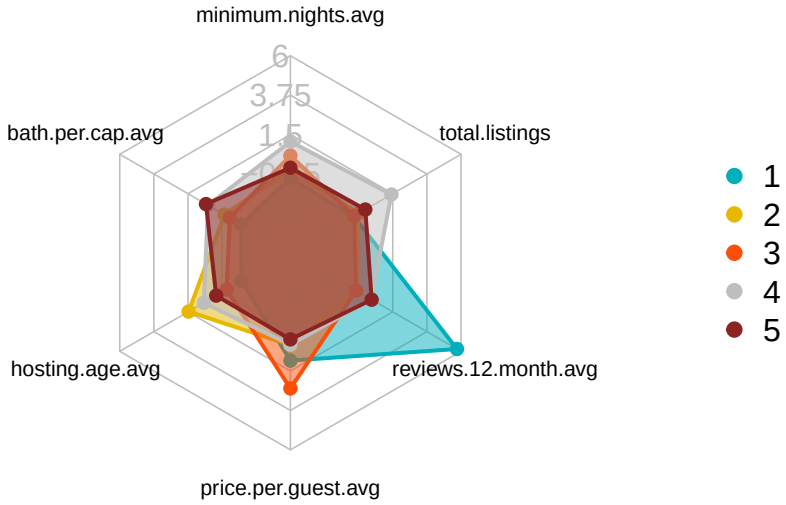



```r
# suggest 5 clusters
km.res <- kmeans(df.clust.scaled, centers = 5, nstart = 25)
table(km.res$cluster)
```

```
##
##  1  2  3  4  5
##  1  8  4 16 36
```

```r
df.clust$clusters <- km.res$cluster

# make the radar plot
df.clust.scaled$clusters <- km.res$cluster
agg <- df.clust.scaled %>%
  group_by(clusters) %>%
  summarise_all(mean)

create_beautiful_radarchart <-
  function(data, color = "#00AFBB",
           vlabels = colnames(data), vlcex = 0.7,
           caxislabels = NULL, title = NULL, ...) {
    radarchart(data, axistype = 1,
      # Customize the polygon
      pcol = color, pfcol = scales::alpha(color, 0.5), plwd = 2, plty = 1,
      # Customize the grid
      cglcol = "grey", cglty = 1, cglwd = 0.8,
```

```
      # Customize the axis
      axislabcol = "grey",
      # Variable labels
      vlcex = vlcex, vlabels = vlabels,
      caxislabels = caxislabels, title = title, ...
    )
  }

max <- ceiling(max(agg[,-1]))
min <- floor(min(agg[,-1]))

agg <- agg %>%
  mutate(clusters=as.character(clusters)) %>%
  add_row(clusters="Min", .before = 1) %>%
  replace(is.na(.), min) %>%
  add_row(clusters="Max", .before = 1) %>%
  replace(is.na(.), max)

colors <- c("#00AFBB", "#E7B800", "#FC4E07", "grey", "brown4")
create_beautiful_radarchart(
  data= agg[,-1], color = colors,
  caxislabels = seq(min, max, (max-min)/4))
# Add an horizontal legend
legend(
  x = "right",
  legend = factor(agg[-c(1,2),]$clusters), horiz = F,
  bty = "n", pch = 20, col = colors,
  text.col = "black", cex = 1, pt.cex = 1.5
)
```




```
# find the corresponding zip codes in each cluster
df.clust$clusters <- km.res$cluster
# table(df.clust$clusters)
as.character(df.clust$zipcode[df.clust$clusters==1])
```

```
## [1] "95140"
```

```
as.character(df.clust$zipcode[df.clust$clusters==2])
```

```
## [1] "94022" "94025" "94304" "94305" "95023" "95076" "95134" "95215"
```

```
as.character(df.clust$zipcode[df.clust$clusters==3])
```

```
## [1] "94526" "95009" "95086" "95113"
```

```
as.character(df.clust$zipcode[df.clust$clusters==4])
```

```
##  [1] "94040" "94041" "94043" "94085" "94086" "94087" "94301" "94306" "95014"
## [10] "95035" "95050" "95051" "95054" "95112" "95126" "95128"
```

```
as.character(df.clust$zipcode[df.clust$clusters==5])
```

```
##  [1] "94024" "94089" "94303" "94539" "95002" "95008" "95020" "95030" "95032"
## [10] "95033" "95037" "95046" "95070" "95110" "95111" "95116" "95117" "95118"
## [19] "95119" "95120" "95121" "95122" "95123" "95124" "95125" "95127" "95129"
## [28] "95130" "95131" "95132" "95133" "95135" "95136" "95138" "95139" "95148"
```