SML 201 Project 3

Tyler Campbell

May 15, 2018

Project 3 is due by 11:59p.m. on Tuesday May 15 on Blackboard. Please submit **both a .Rmd and a .pdf** file on Blackboard **and** drop off a hard copy of the pdf file at 26 Prospect Avenue (see the *Submitting Problem Sets and Projects* section under *Problem Sets and Projects* on the Syllabus for detailed instructions) by **10am** on the **next day** (Wednesday May 16) of the due date; it is fine to drop off the copy before May 16. Please do not modify your .pdf hard copy after your BB submission; otherwise, you might get points deducted.

Late **projects** will be penalized at intervals rounded up to multiples of 24 hours. For example, if you are 3 hours late, 10% off or if you are 30 hours late, 20% off.

Make sure that you have all your digital signatures along with the honor pledge in each of these documents (there should be more than one signature if you work in groups).

This project can be completed in groups of up to 3 students. It is okay to work by yourself, if this is preferable. **You may not work with a given student on more than one project.** In other words, if you work with Student_1 and Student_2 on Project 1, then you cannot work with Student_1 or Student_2 on any other projects. You must form completely new groups for every project.

When working in a group it is your responsibility to make sure that you are satisfied with all parts of the report and the submission is on time (e.g., we will not entertain arguments that deficiencies are the responsibility of other group members). We expect that the work on any given problem set or project contains approximately equal contributions from all members of the group; we expect that you each work independently first and then compare your answers with each other once you all finish or you all work together. Failing to make contributions and then putting your name on a project will be considered a violation of the honor code. Also, please do not divide work among your group mates.

In general you are not allowed to get help on projects from other people except from partners in your group. Clarification questions are always welcome. Please treat projects as take-home exams.

For all parts of this problem set, you **MUST** use R commands to print the output as part of your R Markdown file. You are not permitted to find the answer in the R console and then copy paste the answer into this document.

If you are completing this problem set in a group, please have only **one** person in your group turn in the .Rmd and .pdf files; **other people in your group should turn in the list of the people in your group in the *Text Submission* field on the submission page**.

Please type your name(s) after "Digitally signed:" below the honor pledge to serve as digital signature(s). Put the pledge and your signature(s) at the beginning of each document that you turn in.

I pledge my honor that I have not violated the honor code when completing this assignment.

Digitally signed: Tyler Campbell

In order to receive full credits, please have sensible titles and axis labels for all your graphs and adjust values for all the relevant graphical parameters so that your plots are informative. Also, all answers must be written in complete sentences.

Just a friendly reminder: Please remember to annotate your code and have answers in the write up section, not in code chunks.

Objective of this project

(Hypothetical) Congrats! You have been offered an internship position at Redfin (Redfin.com) in Seattle. Today is your first day at work and your manager would like you to improve the model Redfin uses to predict house prices for the King county area of the Washington state.

To see the prices predicted by Redfin's current model, you can see the number shown near the top of the web page of a listing; e.g., for this house (https://www.redfin.com/WA/Seattle/132-NE-95th-St-98115/unit-B108/home/2316), the Redfin estimate is $469,840.)

You will use a subset of the dataset `kc_house_data.csv` to build the model. The dataset contains sold prices for houses in King County (in Washington state), including Seattle, for transactions made between May 2014 and May 2015. We will use only a subset of the variables in `kc_house_data.csv` because we only want to include variables that have clear definitions and seem relevant for house prices. A description of the original dataset `kc_house_data.csv` and the variable definitions can be found here (https://www.kaggle.com/harlfoxem/housesalesprediction/data).

Background info and the dataset used in this project

We will use the dataset `subset_kc_house_data.csv` and the definitions for the variables in the dataset are listed below (see the website address provided above for the complete list of the variables).

- **date** The date the house was sold
- **price** Sold price of the house
- **bedrooms** Number of bedrooms in the house
- **bathrooms** Number of bathrooms in the house
- **sqft_living** Square footage of the house
- **sqft_lot** Square footage of the lot
- **floors** Total number of floors (levels) in house
- **waterfront** Does the house have a view to a waterfront (0-No; 1-Yes)
- **condition** How good the overall condition is
- **grade** Overall grade given to the housing unit, based on King County grading system
- **sqft_above** Square footage of house apart from the basement
- **sqft_basement** Square footage of the basement
- **yr_built** Year when house was built

- **yr_renovated** Year when house was renovated
- **zipcode** Zip code of the house address

Question 1

Read the dataset `subset_kc_house_data.csv` into `R` and name it `house`. `house` should have 21613 rows and 15 columns.

```
# read csv file to dataframe
house = read.csv("subset_kc_house_data.csv")
# dimension of house df
dim(house)
[1] 21613    15
```

Investigate the relationship between `price` and other variables.

Question 2

Part a Use `ggpairs()` in the `GGally` package and make matrices of scatterplots to investigate the relationships between `price` and other variables; you should make several plots instead of just one so that your scatterplots will not be too small for you to study the relationships. Make sure that the correlation figures are small enough that they fit within the margins of the plots but big enough to be legible.

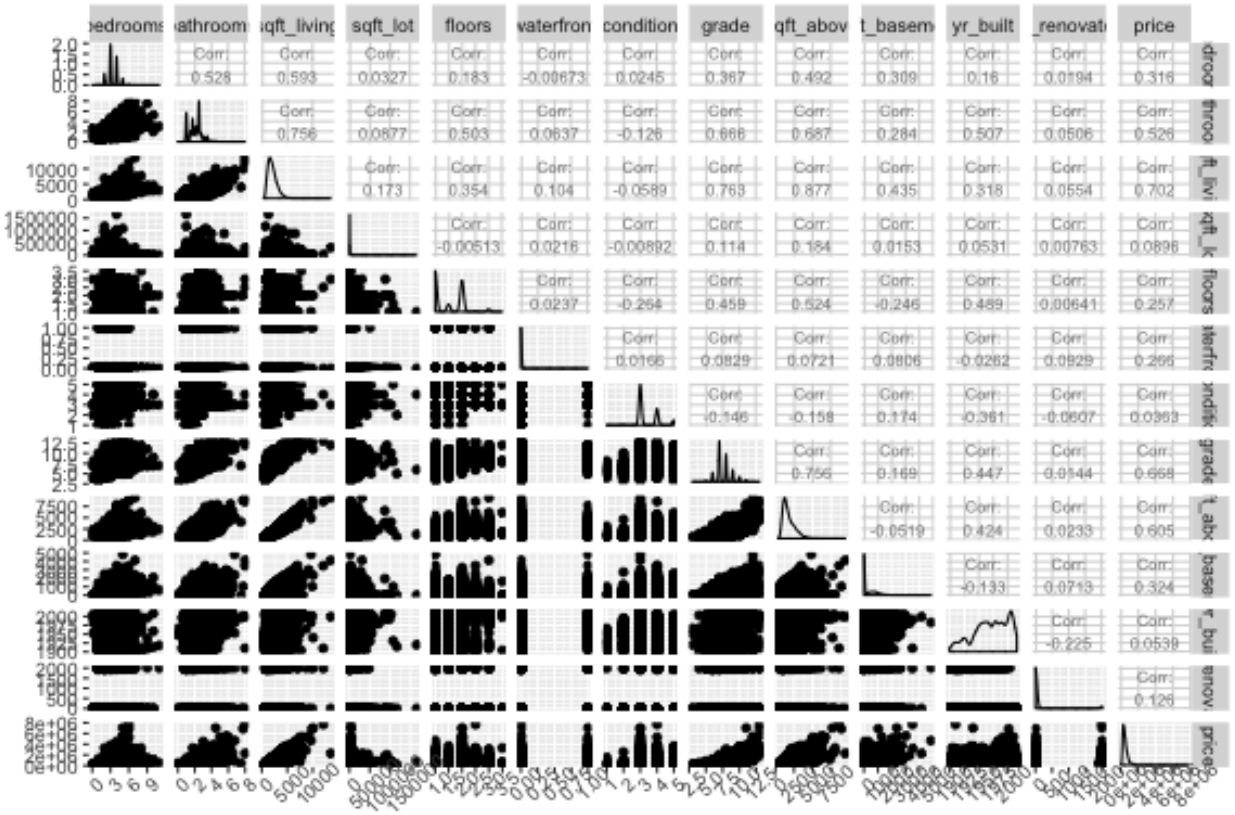
Group `sqft_living`, `sqft_lot`, `sqft_above` and `sqft_basement` in the same plot to see if any of these variables are correlated. In general, it is good to have x-variables that are highly correlated (i.e., with correlation close to -1 or 1)? Explain why yes or why not.

Note that you should not include `date` in your scatterplots since `date` has 372 levels and since the closing date of a transaction should not affect the sold price of a house; (I also investigated the relationship between the month of sale and `price` and did not see any patterns); therefore, we will not include the date-related information in our model.

Also, note that including `zipcode` in the scatterplot matrix will not give you much info as the graph will be too small to show the relationship between `price` and `zipcode`; thus, we will investigate the relationship between `price` and `zipcode` seperately later in the report.

It is bad to have x-varibles that are highly correlated. If they are highly correlated, then there is multi-collinearity and hence, the variance of the dependent variable is not fully explained by a single variable but is repeated as one variable explains another. This also overinflates the standard errors.

```
# 2b remove 8 records that are errors
house = house[!(house$bedrooms == 0 & house$bathrooms == 0) & !(house$bedrooms >
    30), ]

# ggpairs of house without date and zipcode
library(ggplot2)
library(GGally)
ggpairs(house[, c(3:14, 2)], upper = list(continuous = wrap("cor",
    size = 2.6))) + theme(axis.text.x = element_text(angle = 45))
```



Part b From the scatterplots in Part a we see that there is a house with more than 30 bedrooms; what is the living square footage of this house? Also, there are 7 houses recorded with 0 bedroom and 0 bathroom but with positive values for the square footage. All these 8 records are likely to be errors. Add a line of code in Part a before the code for making the matrix scatterplots to remove these 8 observations in `house` and assign the resulting data frame to the same name `house`; this way, your scatterplot matrices will not include these 8 observations. Check the dimension of the new `house` to make sure that the replacement was done correctly.

House with 30 bedrooms has 1620 living square footage.

```
# living square footage of house with 30 bedrooms
house[house$bedrooms > 30, ]$sqft_living
integer(0)
# dimension of new house df
dim(house)
[1] 21605    15
```

Part c From the scatterplots for `price` v.s. `sqft_living`, for `price` v.s. `sqft_above`, and for `price` v.s. `bathrooms` does the spread of `price` gets bigger or smaller as the x-value gets bigger? This indicates a possible violation of which assumption on the error terms in a linear regression model?

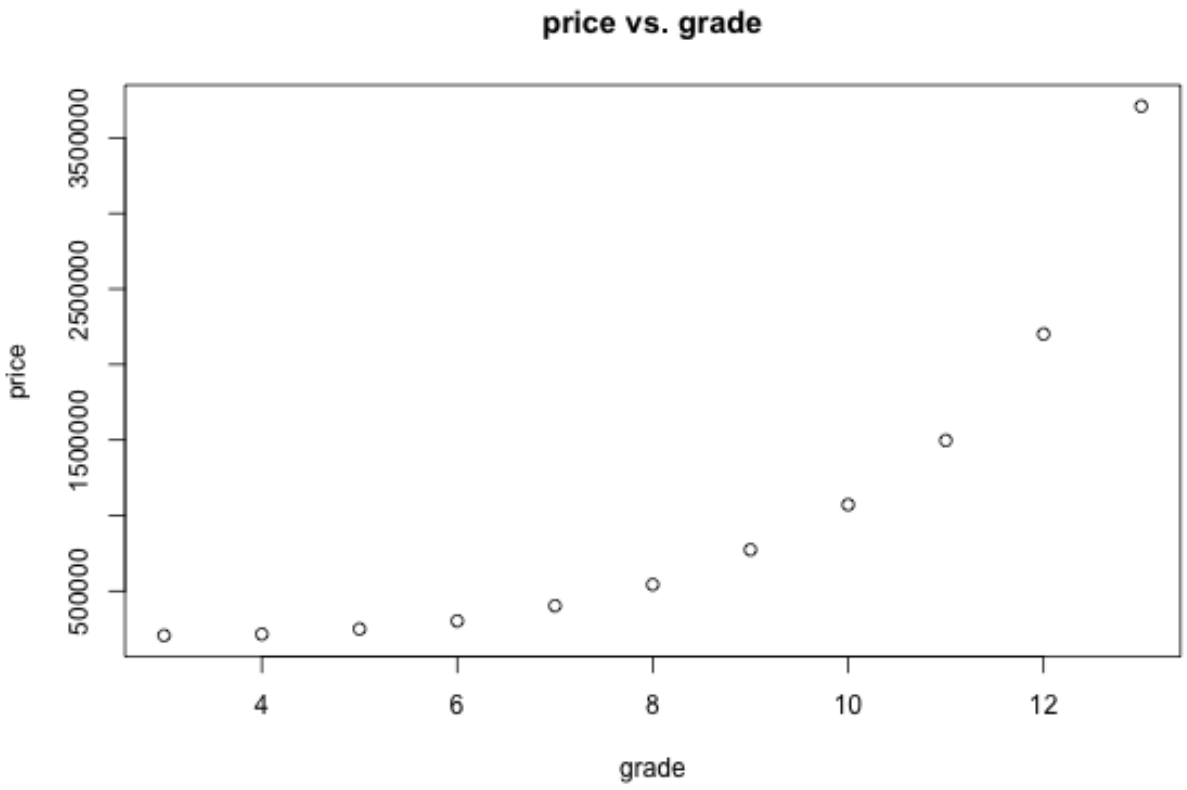
The spread gets bigger for these 3 as the x-value gets bigger. This indicates a violation of constant SD assumption of the errors.

Part d For each of the variables `grade`, `bathrooms` and `yr_built`, calculate the average house price for each unique value of the variable and make a scatterplot for the average house price v.s. the unique values of the variable; e.g., `grade` takes on the integer values 3 through 13 so your scatterplot for `grade` should have 11 points, the first point has x-coordinate 3 and the y-coordinate the average price for all the houses with `grade` = 3, the second point has x-coordinate 4 and the y-coordinate the mean price for all the houses with `grade` = 4, and so on. Do the patterns on these scatterplots look linear? Recall that in the framework for linear

models, if we divide the x-axis into multiple intervals and calculate the mean of the y-values for each vertical strip that correspond to the x-subinterval, then y-means should form a line. However, our data do not show this pattern, so we will need to transform our variable(s) to make the relationships more linear (see question 3).

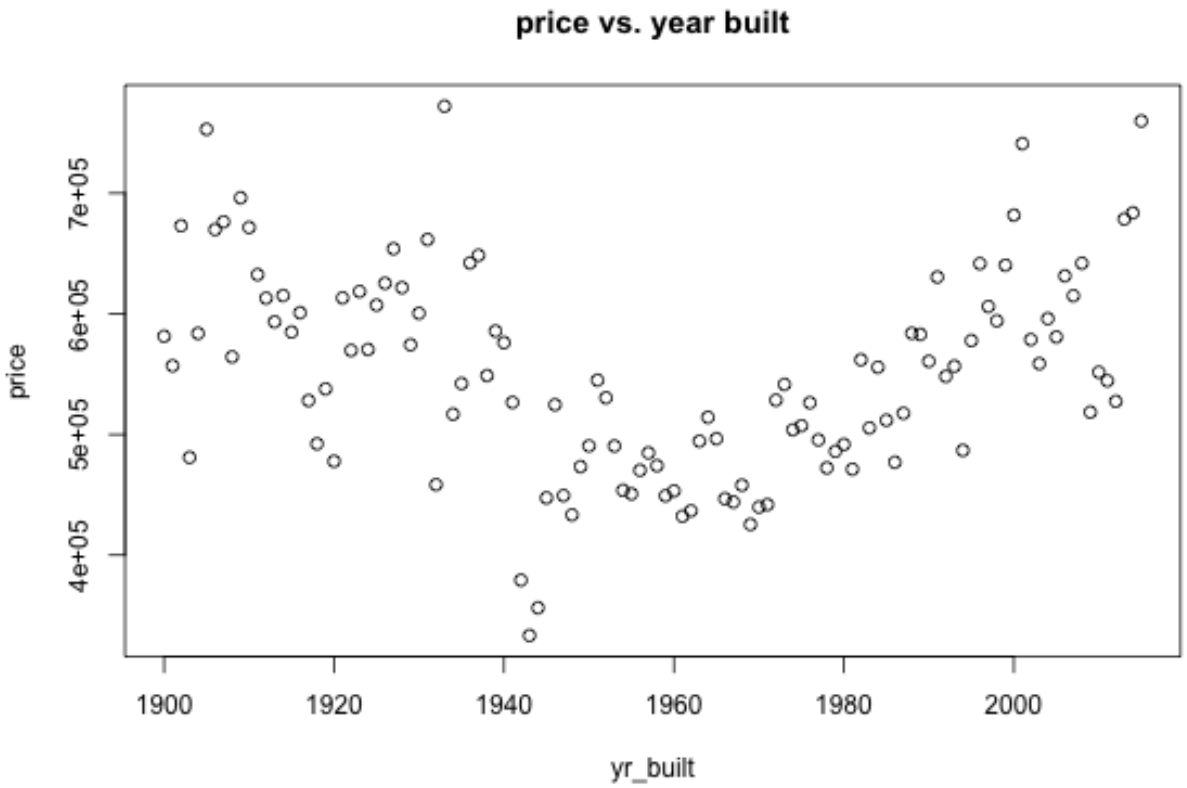
The patterns on these scatterplots do not look linear.

```
library(dplyr)
# creates df of avg price of by each unique value for each 3 of
# variables
vs.grade = house %>% group_by(grade) %>% summarise(price = mean(price))
vs.bathrooms = house %>% group_by(bathrooms) %>% summarise(price = mean(price))
vs.yr_built = house %>% group_by(yr_built) %>% summarise(price = mean(price))
# plot the avg price vs each unique variables
plot(vs.grade, main = "price vs. grade")
```




```
plot(vs.bathrooms, main = "price vs. bathrooms")
```


```
plot(vs.yr_built, main = "price vs. year built")
```



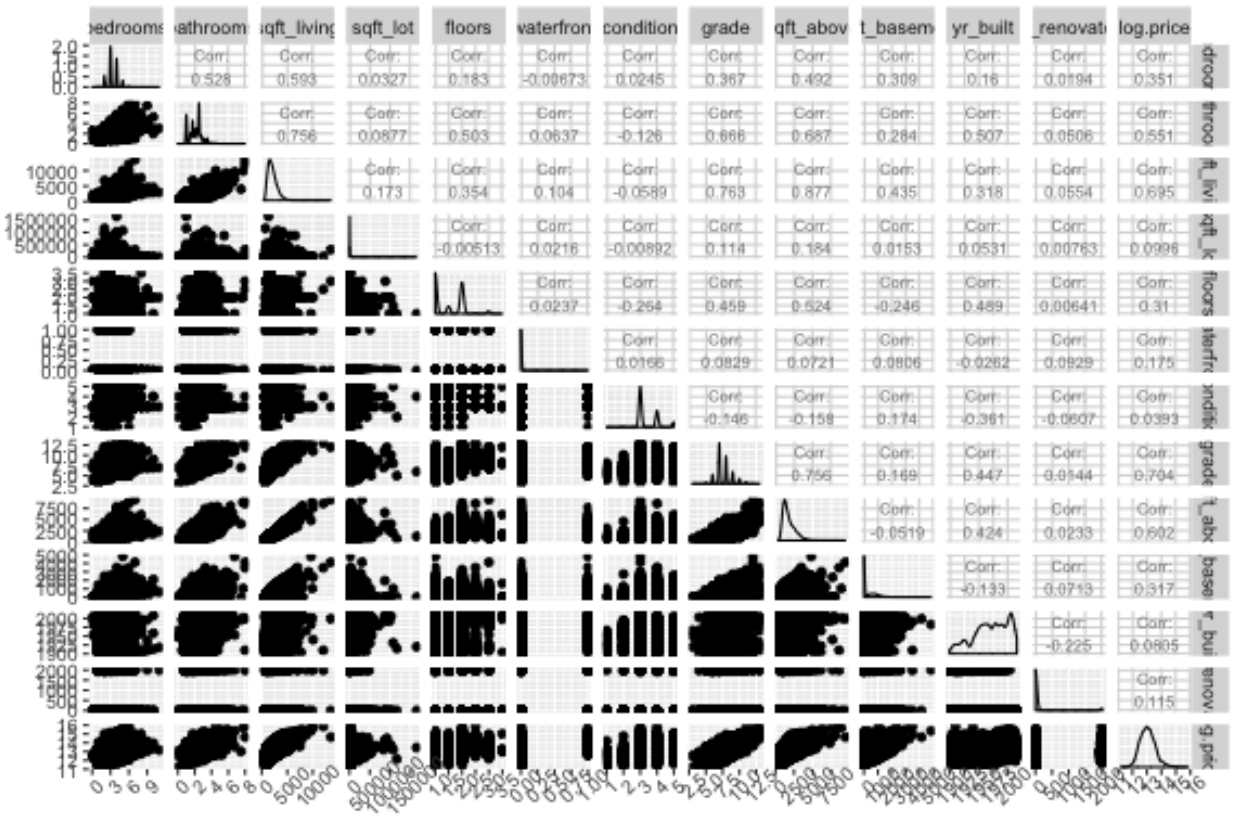
Consider transformations of the data that fit the framework of linear modeling better.

Question 3

The problems discussed in Questions 2.c and 2.d can often be fixed with a natural log transformation of the y-variable.

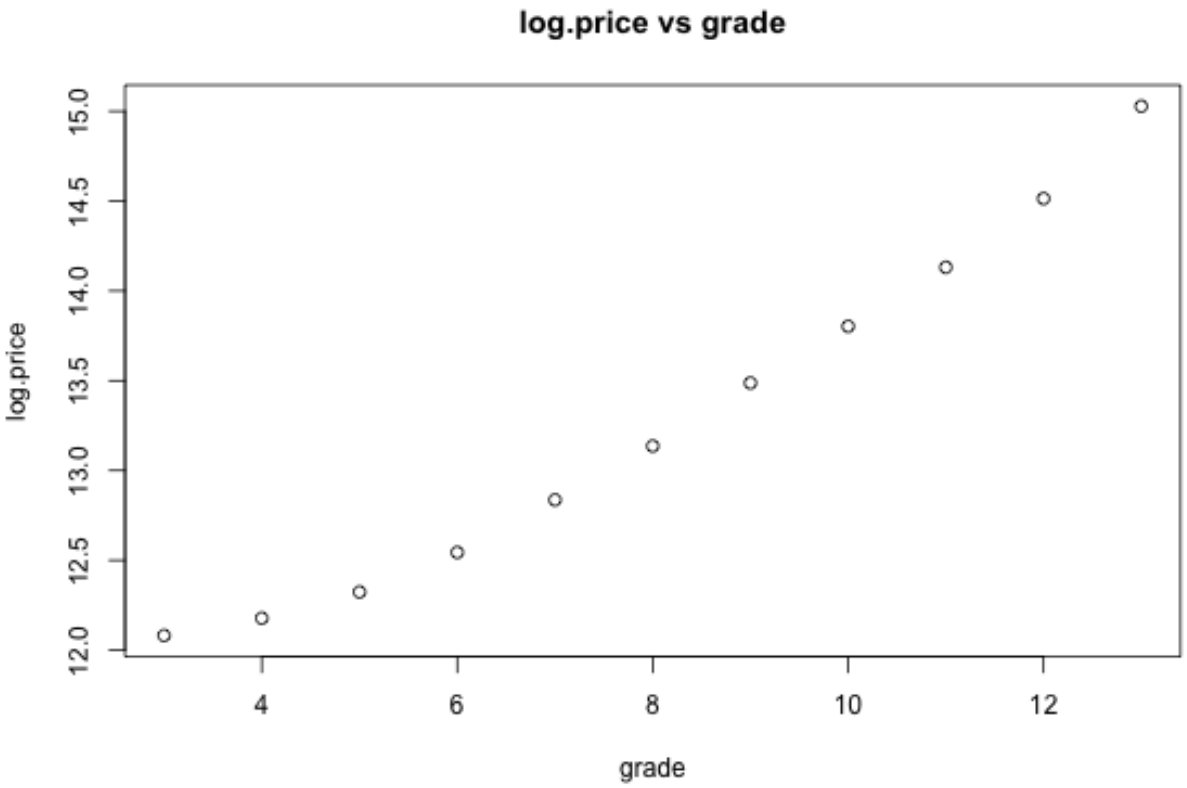
Part a Add the log(`price`) to `house` and name this column `log.price`. Make the matrices of scatterplots you made in Question 2.a again, except that you will replace `price` with `log.price`.

```
# add log(price) to house
house = mutate(house, log.price = log(price))
# ggpairs of house with log.price
ggpairs(house[, c(3:14, 16)], upper = list(continuous = wrap("cor",
    size = 2.6))) + theme(axis.text.x = element_text(angle = 45))
```



Part b Make the scatterplots that you made in Question 2.d for `grade` and `bathrooms`, except that you will replace `price` with `log.price`. Note that now the x-and-y relationships become more linear with the transformation.

```
# creates df of avg log.price of by each unique value for the 2
# variables
vs.grade2 = house %>% group_by(grade) %>% summarise(log.price = mean(log.price))
vs.bathrooms2 = house %>% group_by(bathrooms) %>% summarise(log.price = mean(log.price))
# plot the avg log.price vs each unique variables
plot(vs.grade2, main = "log.price vs grade")
```




```
plot(vs.bathrooms2, main = "log.price vs bathrooms")
```

Part c For `yr_built` transforming `price` will not be enough to make the relationship linear (you are encouraged to make the plot to check this but you are not required to turn in the plot used for checking). Let's transform the x-values too. The original x-values for the scatterplot are `1900:2015` and the mean of the vector `1900:2015` is 1957.5. Transform `yr_built` with $new.yr.built = (yr\_built - 1957.5)^2$ and make the plot that you made in Question 2.d for `yr_built` again with `price` replaced by `log.price` and `yr_built` replaced by `new.yr.built`. Notice that the x-y relationship is now more linear.

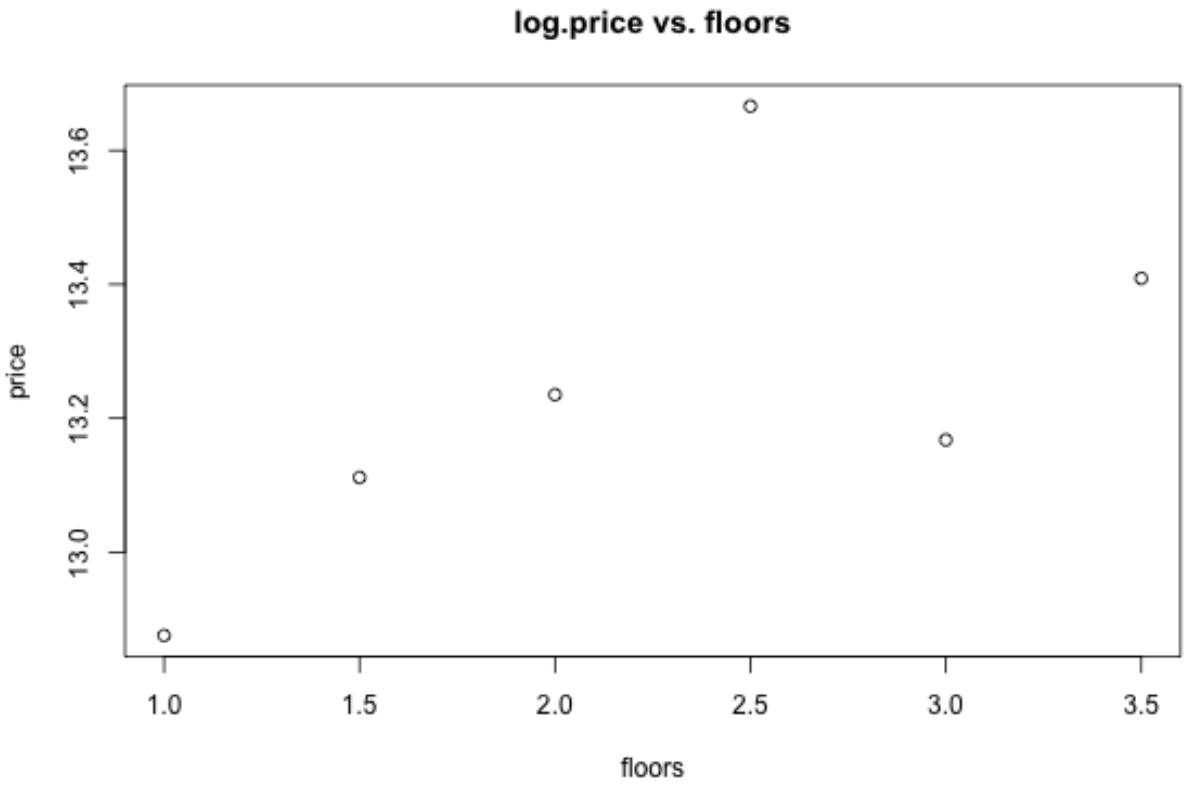
If you are curious about why this transformation will make the x-and-y relationship more linear, feel free to ask any of the instructors.

```
# transform year built
newy = (house$yr_built - 1957.5)^2
# find avg log.price for each unique value of new year built and
# plot
vs.yr_built2 = house %>% mutate(new.yr.built = (house$yr_built -
    1957.5)^2) %>% group_by(new.yr.built) %>% summarise(log.price = mean(log.price))
plot(vs.yr_built2, main = "log.price vs. new year built")
```

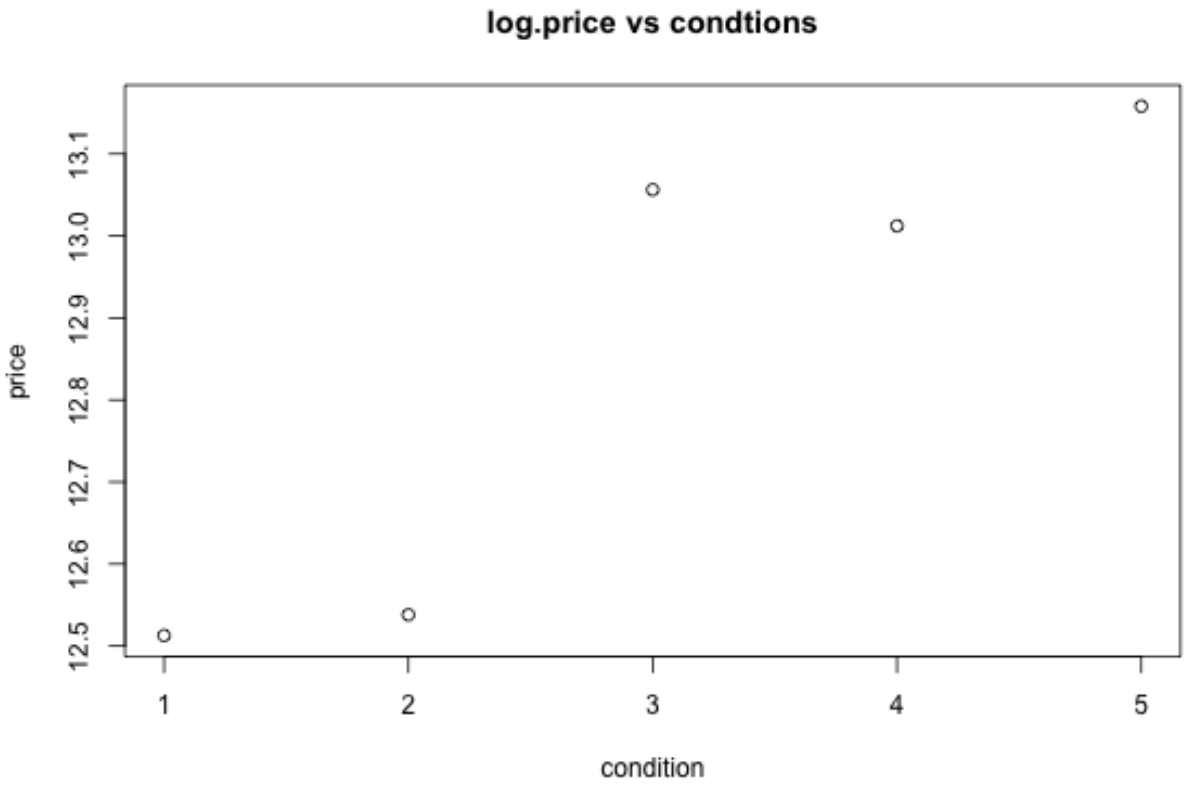
Part d (10 pts) Make similar plots as the ones that you made in Part b for `floors` and `condition`. Does the relationship between the means of `log.price` and the values of `floors` look linear? Similarly, does the relationship between the means of `log.price` and the values of `condition` look linear? Note that it is better to transform these two variables into factors for our model in this case.

Both relationships bewtween (floors and condition) and log.price do not look linear.

```
# avg log.price for floors and condtions
vs.floors = house %>% group_by(floors) %>% summarise(price = mean(log.price))
vs.condition = house %>% group_by(condition) %>% summarise(price = mean(log.price))
# plot for each unique value
plot(vs.floors, main = "log.price vs. floors")
```




```
plot(vs.condition, main = "log.price vs condtions")
```

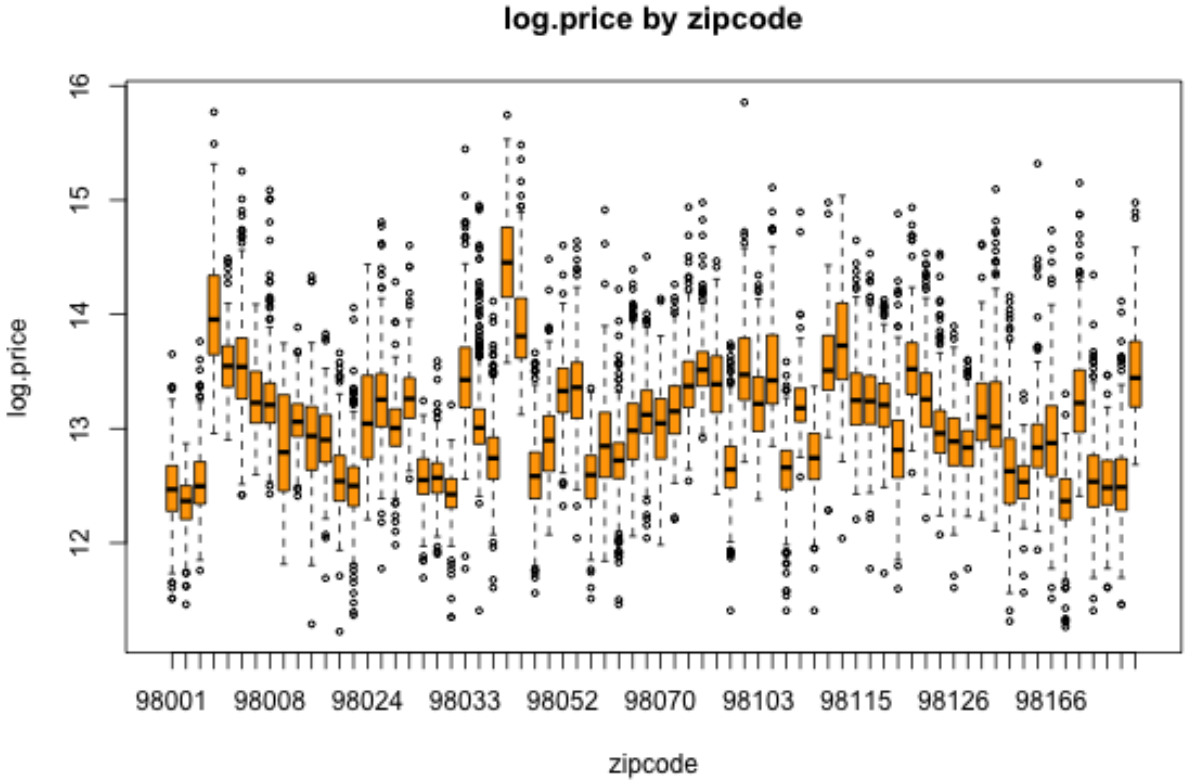


Question 4 Zipcode variable

Part a Make side by side boxplots to compare the values in `log.price` by zipcode. Make sure that your variable `zipcode` is of the correct data type for making the boxplots. Based on your boxplots, is it good to include `zipcode` in your model?

It is good to include zipcode in my model as different zipcodes have differ price ranges.

```
# side by side boxplot
boxplot(house$log.price ~ factor(house$zipcode), col = "orange",
    xlab = "zipcode", ylab = "log.price", main = "log.price by zipcode",
    outcex = 0.5)
```



Part b Consider the following model (you will need to remove `eval=F` to run the code below):

```
# assign given summary of model to a value
pmodel = summary(lm(log.price ~ factor(zipcode), data = house))
```

What is the coefficient estimate for the y-intercept? Interpret this number (hint: this is the estimated average effect on the `log.price` for a certain subset of houses. What is the coefficient estimate for the dummy variable for zipcode 98004? interpret this number too.

The coefficient estimate for the y-intercept is 12.49312. This this is the estimated average effect on the log.price for houses with the zipcode 98001. The coefficient estimate for the dummy variable for zipcode 98004 is 1.51525. This is the addtional estimated average effect on the log.price for houses with this zipcode.

Part c The p-value on the second row of the table in the `Coefficients` table is 1.14e-06. State the Null hypothesis of the test that this p-value is for.

The null hypothesis is that $\beta = 0$.

Part d Answer TRUE or FALSE only for the following statement (no explanation is required and you will only be graded based on the TRUE and FALSE part of your answer even if you provided explanations):

In hypothesis testing the p-value is the chance that the Null hypothesis is true.

FALSE

Part e For the model in Part b how many of the coefficient estimates are significant (i.e., with p-values less than $\alpha = .05$)? Use Bonferroni Correction factor and the method for controlling for False Discovery Rate (FDR) to achieve new cutoffs for the p-values. For each of the two methods, report the adjusted cutoff for the p-values and the number of coefficient estimates that are with p-values less than the new cutoff.

65 of the coefficient estimates are significant. Bonferroni Correction factor: adjusted cutoff = 0.0007142857

and 60 coefficient estimates with p-values less than the new cutoff. FDR: adjusted cutoff = 0.02398299 and 65 coefficient estimates with p-values less than the new cutoff.

```
# get p values from model
p.v = pmodel$coefficients[, 4]
alpha = 0.05
# p values less than alpha
sum(p.v < alpha)
[1] 65
# Bonferroni Correction factor
m = length(p.v)
cutoff = alpha/m
cutoff
[1] 0.0007142857
sum(p.v < cutoff)
[1] 60
# False Discovery Rate
sort.p = sort(p.v)
cutoff = max(sort.p[sort.p <= (1:m)/m * alpha])
sum(p.v <= cutoff)
[1] 65
cutoff
[1] 0.02398299
```

Creating new variables for the model

We will make a new data frame for all the variables that we will need for building the model.

Question 5

We will create the a new data frame `mod.variables` with all the transformed variables and some of the original variables. `mod.variables` should include `log.price`, `sqft_living`, `sqft_basement`, `grade`, `bedrooms`, `bathrooms` and `date` from the data frame `house` plus the following transformed variables:

- `f.waterfront`: the factor version of `house$waterfront`;
- `f.floor`: the factor version of `house$floor`;
- `f.cond`: the factor version of `house$condition`;
- `f.renov`: a factor vector and each element in `f.renov` is 1 if the corresponding element in `house$yr_renovated` does not equal to zero, and 0 otherwise;
- `trs.yr.built`: `new.yr.built` created in Question 3.c;
- `f.zipcode`: the factor version of `house$zipcode`.

We decided not to keep the variable `sqft_lot` since it does not have much linear relationship with `log.price`. `sqft_above` is also dropped since it is highly correlated with `sqft_living`.

```
# create new mod.variables dataframe from existing variables in
# house and transforming some variabled
renovated = house$yr_renovated
renovated[renovated != 0] = 1
mod.variables = house %>% select("log.price", "sqft_living", "sqft_basement",
    "grade", "bedrooms", "bathrooms", "date") %>% mutate(f.waterfront = as.factor(house$waterfront)) %>%
    mutate(f.floor = as.factor(house$floors)) %>% mutate(f.cond = as.factor(house$condition)) %>%
```

```
    mutate(trs.yr.built = (house$yr_built - 1957.5)^2) %>% mutate(f.zipcode = as.factor(house$zipcode)) %>%
    mutate(f.renov = as.factor(renovated))
```

Divide data into three subsets: training+validation set and two test sets.

For this part you just need to remove the `eval=F` argument for each of the code chunks below and run the code; no code is required from you.

`mod.variables` covers the period from May 2014 to May 2015. We will prepare a vector `date.format` that we can use to extract out the observations that correspond to transactions closed in May 2015.

```
date.format = format(as.Date(mod.variables$date, "%Y%m%dT"), "%Y%m")
length(date.format)
[1] 21605
head(date.format)
[1] "201410" "201412" "201502" "201412" "201502" "201405"
dim(mod.variables)
[1] 21605    13
```

Then, the following lines will extract out all the observations with transactions closed in May 2015. We will use these observations for our out-of-time test set `test2`. `test2` should be 646 by 12.

```
test2 = mod.variables[date.format %in% "201505", !(names(mod.variables) %in%
    c("date"))]

dim(test2)
[1] 646  12
```

For the rest of the observations run the following code chunk; this will split the remaining observations into 2 sets: 85% training plus validation set and 15% in-time test set.

```
tmp = mod.variables[!date.format %in% "201505", !(names(mod.variables) %in%
    c("date"))]

s = dim(tmp)[1]
s
[1] 20959

set.seed(2939)
permu = sample(1:s)
train.val = tmp[permu, ][1:round(s * 0.85), ]
dim(train.val)
[1] 17815    12


test1 = tmp[permu, ][(round(s * 0.85) + 1):s, ]
dim(test1)
[1] 3144   12
```

Model selection: 5-fold cross validation with the MSE criterion

We are now ready to build our model! We will use 5-fold cross validation with the MSE as our criterion for our model selection. `mod.variables` has been divided into 3 sets:

- `train.val`: The 17815 by 12 training plus validation set
- `test1`: The 3144 by 12 in-time test set
- `test2`: The 646 by 12 out-of-time test set

Question 6

For the rest of the project we will build our model with the training plus validation set, and test the performance of the model with the 2 test sets.

Part a We are trying to find the champion model that predicts `log.price` for houses in King county, Washington with 5-fold cross validation. For your model you should consider all the variables in `mod.variables`, except `date`, plus the interaction term `f.waterfront:sqft_living`. Create a 5 by `l` matrix `v_errors` of NA values. We will use this matrix to store the mean squared error estimates for the "best model" among all the candidate models with a certain number of predictors. What is the value for l? Hint: If you are not sure about this. You can apply lm() on the full model to find out the maximum number of predictors that you can have for the candidate models.

The value for l is 87, assuming it is the maximum number of predictors.

```
v.errors1 = matrix(NA, 5, 87, dimnames = list(NULL, paste(1:87)))
v.errors2 = matrix(NA, 5, 87, dimnames = list(NULL, paste(1:87)))
```

Part b Use `regsubsets()` and 5-fold cross validation (with MSE criterion) to choose the number of predictors that you want to include in your champion model. There might be too many variables for you to consider every possible subset of the potential predictors; in that case, you might want to use forward selection *and* backward selection (this can be done by using the input argument `method` in `regsubsets()`) and compare their findings.

For each selection method report the average MSE's for the best model for each given number of predictors (it is fine to just print the average MSE's out with your code). Also, plot the average MSE v.s. the number of predictors in the best model. Report the minimum estimated average MSE. Report the number of predictors for your champion model ~~and print out their names in your code~~. Justify your choice of the number of predictors for your champion model. ~~What is the mean squared error estimate for your champion model based on the validation set?~~

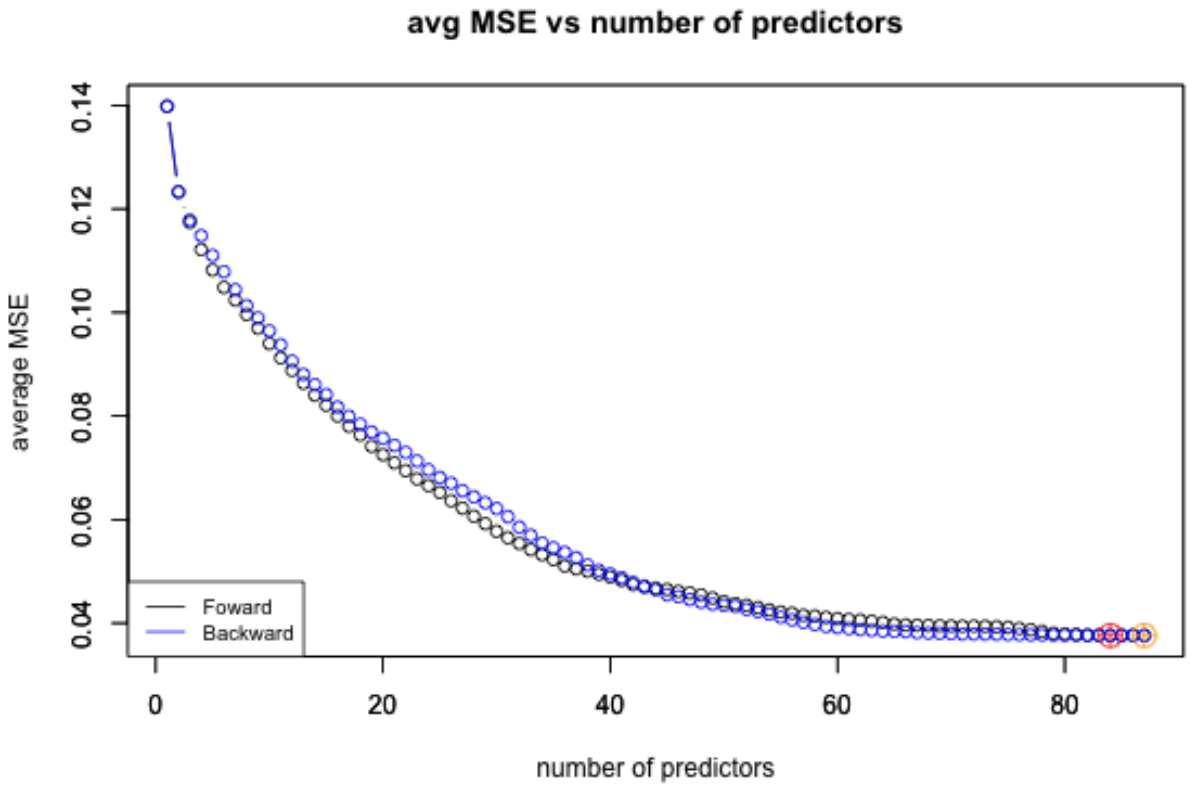
The minimum MSE for Forward selection method is 0.03761127. The minimum MSE for Backward selection method is 0.03761267. Will use 84 predictors in champion model since the min MSE for forward is less than backward and this min occurs when there are 84 predictors.

```
library(leaps)
k = 5  # number of folds
set.seed(2132)
folds = sample(1:k, nrow(train.val), replace = TRUE)

predict.reg = function(object, newdata, id) {
    form = as.formula(object$call[[2]])  # Extract the formula used when we called regsubsets()
    mat = model.matrix(form, newdata)  # Build the model matrix
    coefi = coef(object, id = id)  # Extract the coefficients of the id'th model
    xvars = names(coefi)  # Pull out the names of the predictors used in the ith model
```

```
    mat[, xvars] %*% coefi  # Make predictions using matrix multiplication
}

for (j in 1:k) {

    # Perform best subset selection on the full dataset, minus the
    # jth fold
    best_fit_f = regsubsets(log.price ~ . + f.waterfront:sqft_living,
        data = train.val[folds != j, ], nvmax = 87, method = "forward")

    best_fit_b = regsubsets(log.price ~ . + f.waterfront:sqft_living,
        data = train.val[folds != j, ], nvmax = 87, method = "backward")

    # Inner loop iterates over each number of predictors
    for (i in 1:87) {

        # Predict the values of the current fold with the 'best' model
        # for i predictors
        pred1 = predict.reg(best_fit_f, newdata = train.val[folds ==
            j, ], id = i)

        pred2 = predict.reg(best_fit_b, newdata = train.val[folds ==
            j, ], id = i)
        # Calculate the MSE for the jth fold and best i-predictor model,
        # store it in the matrix we created above
        v.errors1[j, i] = mean((train.val$log.price[folds == j] -
            pred1)^2)

        v.errors2[j, i] = mean((train.val$log.price[folds == j] -
            pred2)^2)
    }
}
# Take the mean of over all folds for each model size
mean.v.errors1 = apply(v.errors1, 2, mean)
mean.v.errors2 = apply(v.errors2, 2, mean)
# print avg MSE for all number of predictores
print("Forward: ")
[1] "Forward: "
mean.v.errors1
         1          2          3          4          5
0.13983054 0.12327666 0.11740255 0.11214940 0.10820913
         6          7          8          9         10
0.10484851 0.10241477 0.09961300 0.09696566 0.09399125
        11         12         13         14         15
0.09122288 0.08880568 0.08633753 0.08403453 0.08207406
        16         17         18         19         20
0.07999466 0.07802240 0.07633675 0.07415646 0.07252011
        21         22         23         24         25
0.07097735 0.06943243 0.06782867 0.06655072 0.06524477
        26         27         28         29         30
0.06361580 0.06219101 0.06063852 0.05922321 0.05771644
        31         32         33         34         35
0.05646744 0.05538883 0.05420286 0.05321477 0.05226912
```

```
        36         37         38         39         40
0.05101855 0.05051221 0.05006162 0.04944917 0.04891285
        41         42         43         44         45
0.04824790 0.04746233 0.04694426 0.04675255 0.04649993
        46         47         48         49         50
0.04614464 0.04582801 0.04544646 0.04483164 0.04416053
        51         52         53         54         55
0.04376132 0.04343047 0.04300466 0.04254367 0.04213397
        56         57         58         59         60
0.04189884 0.04150238 0.04136232 0.04111106 0.04084803
        61         62         63         64         65
0.04073092 0.04057398 0.04037572 0.04012487 0.03981144
        66         67         68         69         70
0.03964295 0.03959830 0.03954862 0.03949039 0.03943927
        71         72         73         74         75
0.03940222 0.03942728 0.03938077 0.03929991 0.03919112
        76         77         78         79         80
0.03895179 0.03873057 0.03840549 0.03802633 0.03790297
        81         82         83         84         85
0.03780552 0.03765451 0.03764113 0.03761127 0.03761958
        86         87
0.03762671 0.03761267
print("Backward: ")
[1] "Backward: "
mean.v.errors2
         1          2          3          4          5
0.13983054 0.12327666 0.11783139 0.11484503 0.11102159
         6          7          8          9         10
0.10787610 0.10440230 0.10130555 0.09895827 0.09645082
        11         12         13         14         15
0.09374865 0.09061192 0.08804232 0.08606196 0.08412869
        16         17         18         19         20
0.08173149 0.07994374 0.07843024 0.07690325 0.07573482
        21         22         23         24         25
0.07435455 0.07298515 0.07132273 0.06963023 0.06813811
        26         27         28         29         30
0.06702742 0.06557537 0.06439312 0.06325564 0.06214390
        31         32         33         34         35
0.06056087 0.05853996 0.05697261 0.05548578 0.05454384
        36         37         38         39         40
0.05367490 0.05262500 0.05124218 0.05019507 0.04951317
        41         42         43         44         45
0.04879031 0.04788168 0.04715902 0.04644882 0.04549043
        46         47         48         49         50
0.04512775 0.04458458 0.04404023 0.04366570 0.04345039
        51         52         53         54         55
0.04324506 0.04265283 0.04223274 0.04174999 0.04115979
        56         57         58         59         60
0.04062089 0.04014425 0.03974209 0.03934160 0.03916169
        61         62         63         64         65
0.03893768 0.03875892 0.03861642 0.03846626 0.03840043
        66         67         68         69         70
0.03838644 0.03817176 0.03807940 0.03801396 0.03798590
```

```
       71         72         73         74         75
0.03790868 0.03789615 0.03788190 0.03785509 0.03780602
       76         77         78         79         80
0.03777483 0.03772105 0.03771197 0.03768715 0.03766320
       81         82         83         84         85
0.03767916 0.03766528 0.03766005 0.03765298 0.03764735
       86         87
0.03762956 0.03761267
# find and print min and when they occur
print(c(min(mean.v.errors1), which.min(mean.v.errors1)))
                    84
 0.03761127 84.00000000
print(c(min(mean.v.errors2), which.min(mean.v.errors2)))
                    87
 0.03761267 87.00000000

# plot avg MSE vs number of predictors
min1 = which.min(mean.v.errors1)
min2 = which.min(mean.v.errors2)
plot(mean.v.errors1, type = "b", xlab = "", ylab = "")
par(new = T)
plot(mean.v.errors2, type = "b", col = "blue", main = "avg MSE vs number of predictors",
    xlab = "number of predictors", ylab = "average MSE")
points(min1, mean.v.errors1[min1][1], col = "red", cex = 2, pch = 10)
points(min2, mean.v.errors2[min2][1], col = "orange", cex = 2, pch = 10)
legend("bottomleft", legend = c("Foward", "Backward"), col = c("black",
    "blue"), lty = 1:1, cex = 0.8)
```



Part c Obtain your champion model and the estimates for the coefficients in the model by using `regsubsets()`. Since backward and forward selections could potentially give you different champion models even for the same given number of predictors, you should compare the results from both methods and pick the model with lower MSE.

Write a line of code to check whether the sets of coefficients picked by backward and forward selections are the same or not.

List the names of the variables in your champion model along with their estimates (it's okay to just print these values out in your R code; you do not need to copy and paste them into the written part of your report).

0.03761127 is the lowest MSE.

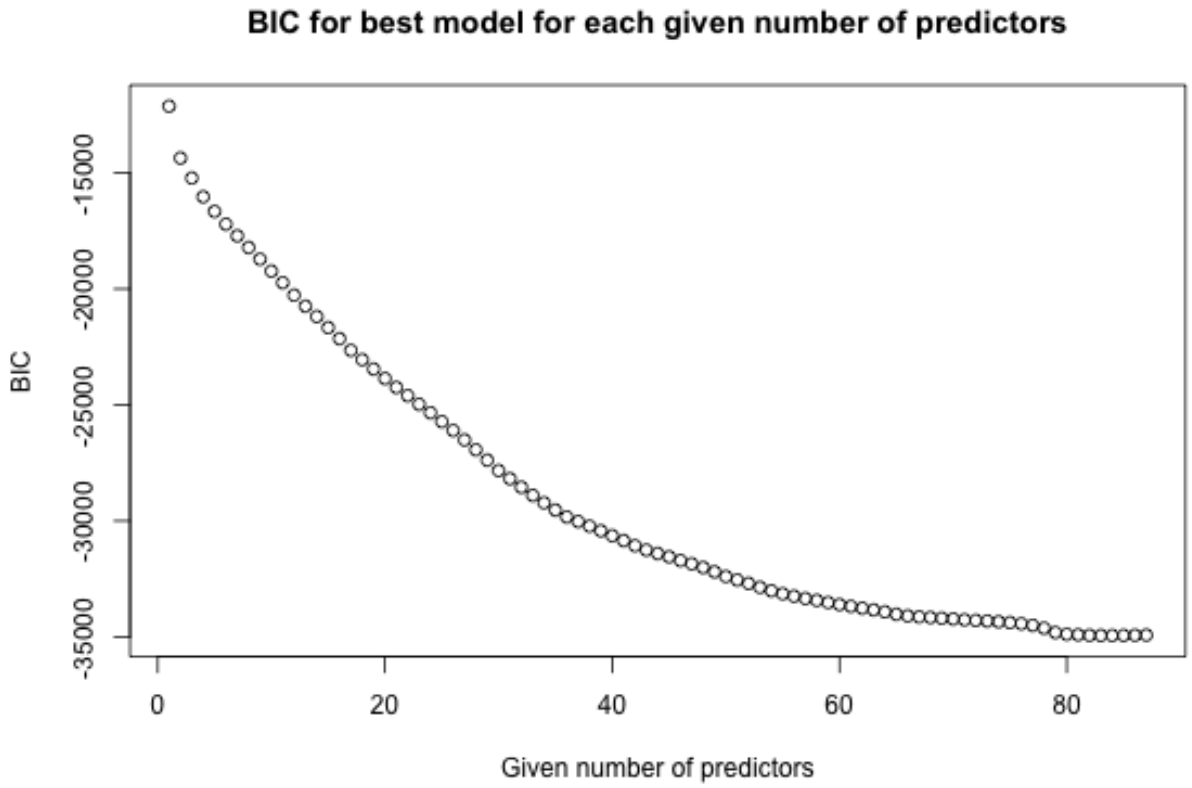
```
# best fit from regsubsets
bestf = regsubsets(log.price ~ . + f.waterfront:sqft_living, data = train.val,
    method = "forward", nvmax = 87)
bestb = regsubsets(log.price ~ . + f.waterfront:sqft_living, data = train.val,
    method = "backward", nvmax = 87)
# Build the design matrix
mat1 = model.matrix(log.price ~ . + f.waterfront:sqft_living, data = train.val)
mat2 = model.matrix(log.price ~ . + f.waterfront:sqft_living, data = train.val)
# Extract the coefficients of the id'th model
coefi1 = coef(bestf, id = 84)
coefi2 = coef(bestb, id = 84)
# check if coefficients from backward and forward selections are
# the same or not(true means they are same)
setequal((names(coefi1)), (names(coefi2)))
[1] FALSE
```

```
coefi1
                (Intercept)                  sqft_living
                1.091049e+01                 2.419096e-04
               sqft_basement                        grade
               -5.904935e-05                 1.134643e-01
                   bathrooms                f.waterfront1
                2.516166e-02                 7.998419e-01
                    f.floor2                   f.floor2.5
               -4.471558e-02                -5.855230e-02
                    f.floor3                        f.cond2
               -1.758595e-01                 9.554390e-02
                    f.cond3                        f.cond4
                2.290849e-01                 2.898702e-01
                    f.cond5                  trs.yr.built
                3.501822e-01                 1.280007e-05
              f.zipcode98002               f.zipcode98003
               -4.491910e-02                 2.482756e-02
              f.zipcode98004               f.zipcode98005
                1.115215e+00                 7.606094e-01
              f.zipcode98006               f.zipcode98007
                6.532649e-01                 6.423307e-01
              f.zipcode98008               f.zipcode98010
                6.701015e-01                 2.354203e-01
              f.zipcode98011               f.zipcode98014
                4.610902e-01                 3.556886e-01
              f.zipcode98019               f.zipcode98022
                3.545441e-01                 1.052479e-01
              f.zipcode98023               f.zipcode98024
               -3.048053e-02                 5.047615e-01
              f.zipcode98027               f.zipcode98028
                5.223122e-01                 4.365422e-01
              f.zipcode98029               f.zipcode98030
                5.893062e-01                 5.721923e-02
              f.zipcode98031               f.zipcode98032
                7.540562e-02                -3.815566e-02
              f.zipcode98033               f.zipcode98034
                7.745842e-01                 5.410016e-01
              f.zipcode98038               f.zipcode98039
                1.797824e-01                 1.202727e+00
              f.zipcode98040               f.zipcode98042
                8.914159e-01                 6.150664e-02
              f.zipcode98045               f.zipcode98052
                3.615979e-01                 6.366789e-01
              f.zipcode98053               f.zipcode98055
                5.949956e-01                 1.443036e-01
              f.zipcode98056               f.zipcode98058
                3.125391e-01                 1.620191e-01
              f.zipcode98059               f.zipcode98065
                3.380356e-01                 4.224383e-01
              f.zipcode98070               f.zipcode98072
                3.604103e-01                 5.130143e-01
              f.zipcode98074               f.zipcode98075
                5.644118e-01                 5.737100e-01
```

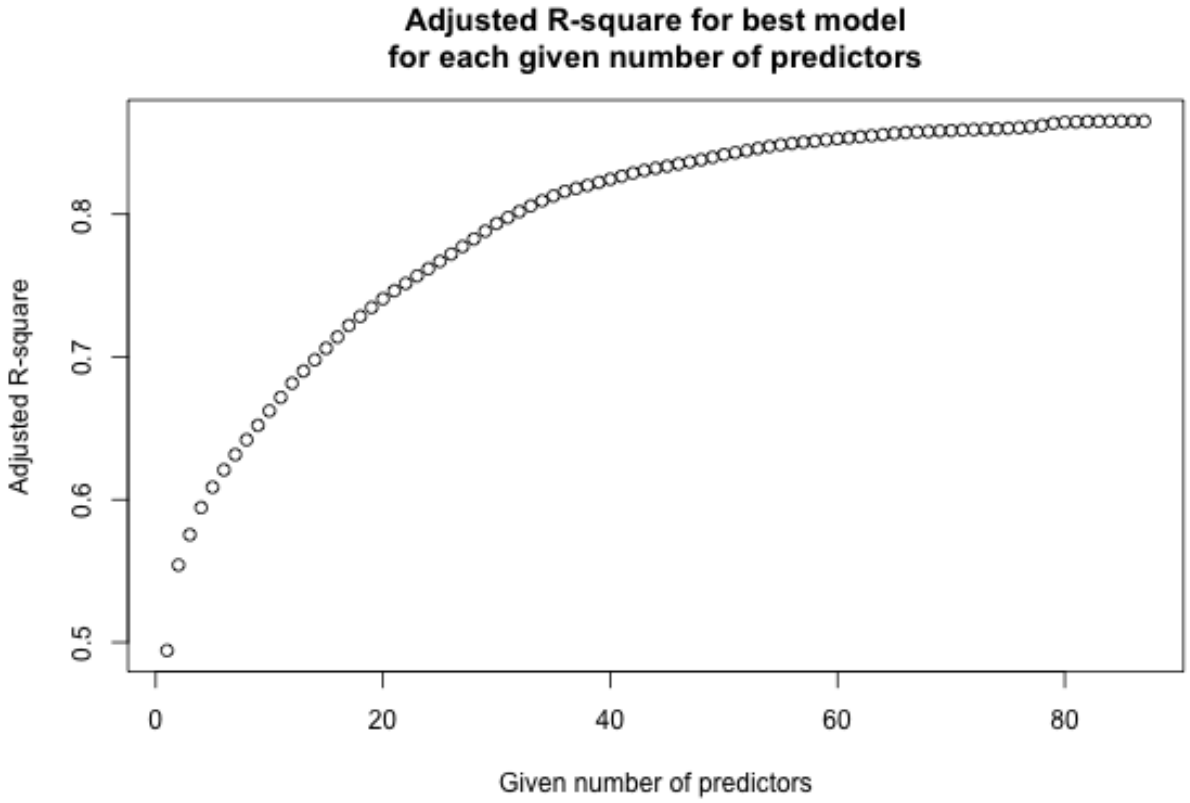
```
          f.zipcode98077           f.zipcode98092
            4.773275e-01             4.524822e-02
          f.zipcode98102           f.zipcode98103
            9.284863e-01             8.001914e-01
          f.zipcode98105           f.zipcode98106
            9.296221e-01             2.914972e-01
          f.zipcode98107           f.zipcode98108
            8.229611e-01             3.326978e-01
          f.zipcode98109           f.zipcode98112
            9.756342e-01             1.021699e+00
          f.zipcode98115           f.zipcode98116
            8.080155e-01             7.586628e-01
          f.zipcode98117           f.zipcode98118
            7.841415e-01             4.461520e-01
          f.zipcode98119           f.zipcode98122
            9.701217e-01             7.730031e-01
          f.zipcode98125           f.zipcode98126
            5.742460e-01             5.231156e-01
          f.zipcode98133           f.zipcode98136
            4.406775e-01             6.831748e-01
          f.zipcode98144           f.zipcode98146
            6.412015e-01             2.952035e-01
          f.zipcode98148           f.zipcode98155
            1.691323e-01             4.256983e-01
          f.zipcode98166           f.zipcode98168
            3.230306e-01             7.195707e-02
          f.zipcode98177           f.zipcode98178
            6.277709e-01             1.727081e-01
          f.zipcode98188           f.zipcode98198
            8.326153e-02             9.118005e-02
          f.zipcode98199               f.renov1
            8.655618e-01             1.020882e-01
sqft_living:f.waterfront1
           -4.683894e-05
```

Part d The output of `regsubsets()` in the previous part includes the best models for each given number of predictors. Plot the BIC, Adjusted R^2 and Mallow's Cp for these best models to further confirm your choice of the number of predictors for your champion model. You only need to do this for one of the selection methods; e.g., if your champion model was produced by the backward selection method, you just need to do this for the output produced by backward selection. If both methods give you the same result then feel free to pick one of the methods for this part. My choice for number of the number of predictors is consistent with the plots.

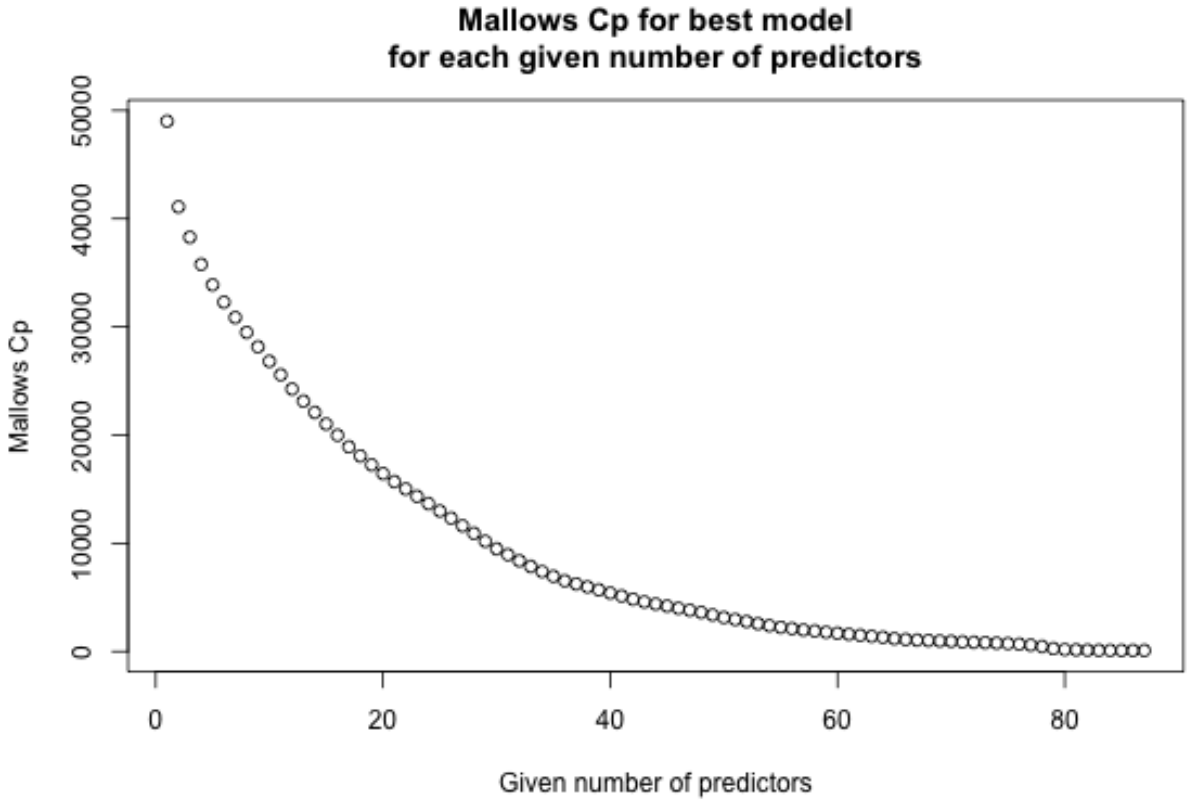
```
tmp = regsubsets(log.price ~ . + f.waterfront:sqft_living, data = train.val,
    really.big = T, method = "forward", nvmax = 87)
# plot BIC, Adjusted R^2, and Mallow's Cp
plot(summary(tmp)$bic, main = "BIC for best model for each given number of predictors",
    xlab = "Given number of predictors", ylab = "BIC")
```




```
plot(summary(tmp)$adjr2, main = "Adjusted R-square for best model
for each given number of predictors",
    xlab = "Given number of predictors", ylab = "Adjusted R-square")
```



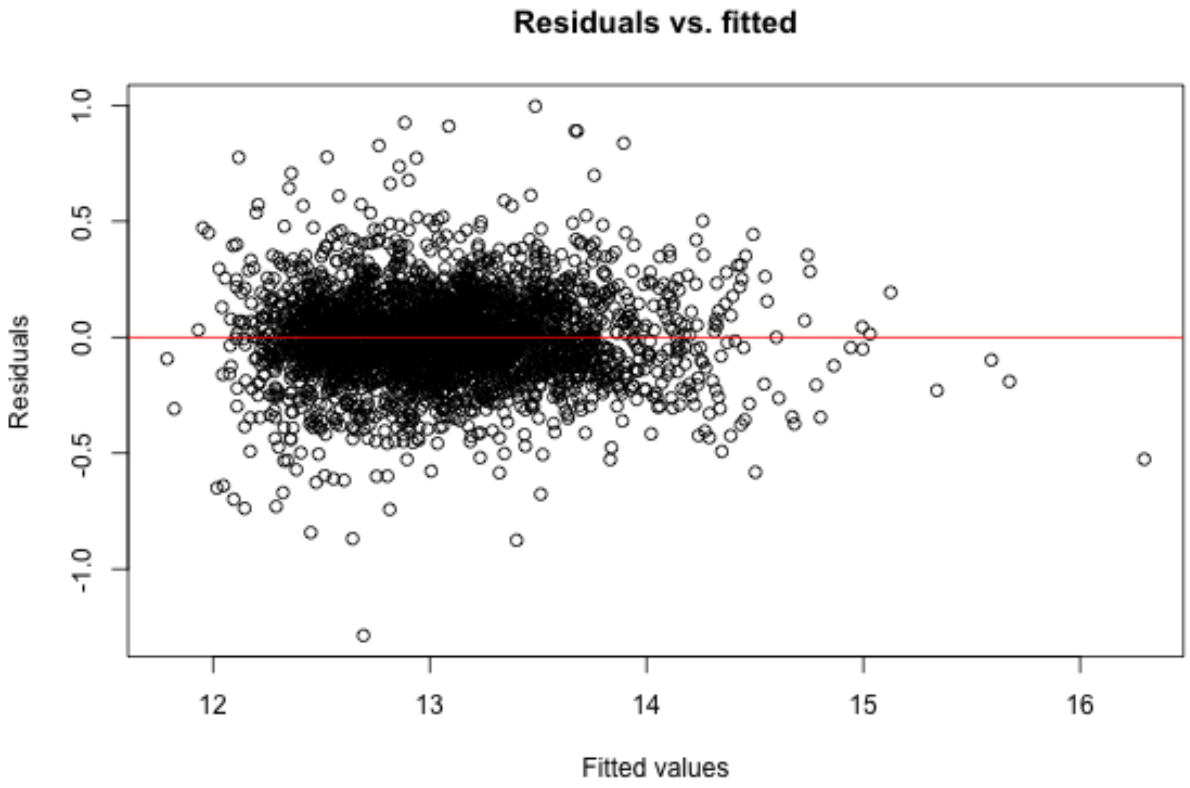

```
plot(summary(tmp)$cp, main = "Mallows Cp for best model
for each given number of predictors",
    xlab = "Given number of predictors", ylab = "Mallows Cp")
```



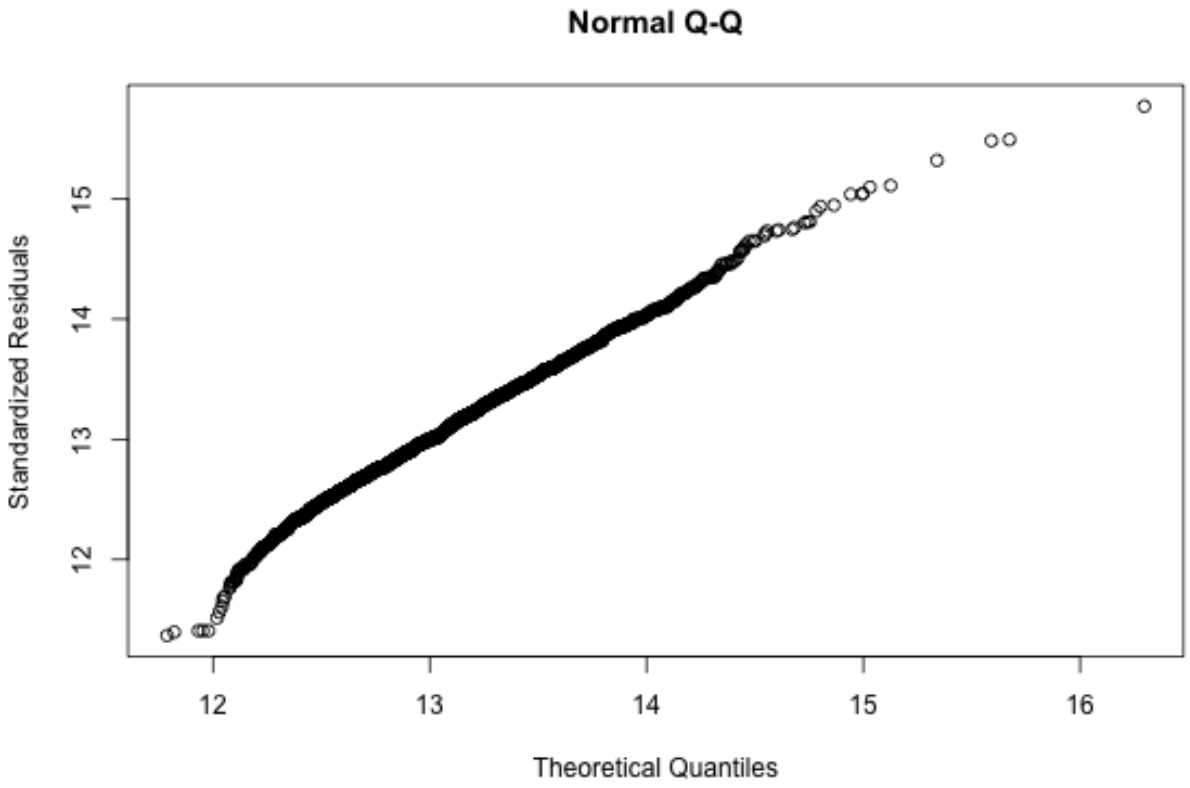
Part e For your champion model plot the residual plot, the qq plot and the histogram for the distribution of the residuals. What potential problems of the model do you see from these plots (hint: think about the assumptions for the distribution of the errors)?

There are no problems that come from the model. The assumptions for the distribution of the errors includes that the errors follow a normal distribution. All 3 plot show that it is normally distributed.

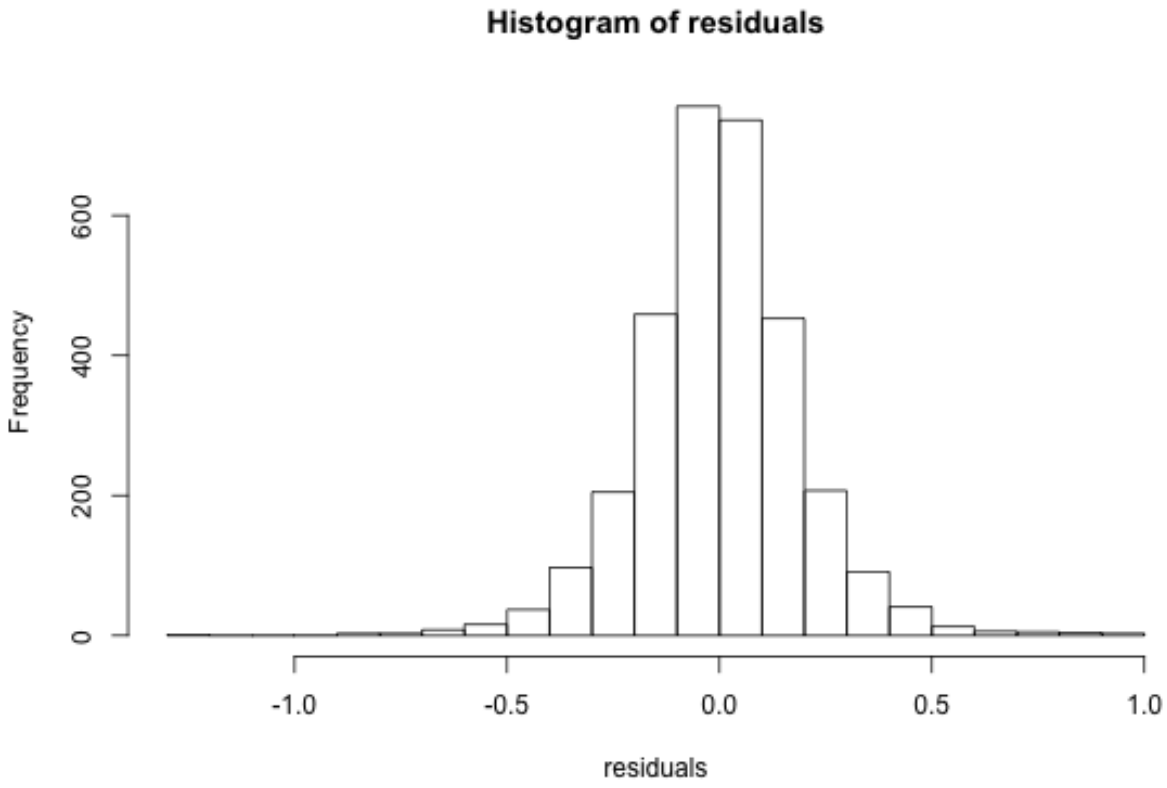
```
pred_x = predict.reg(bestf, test1, id = 84)
# residuals plot
plot(pred_x, test1$log.price - pred_x, xlab = "Fitted values", ylab = "Residuals",
    main = "Residuals vs. fitted")
abline(h = 0, col = "red")
```




```
## qq plot
qqplot(pred_x, test1$log.price, xlab = "Theoretical Quantiles",
    ylab = "Standardized Residuals", main = "Normal Q-Q")
```




```
## histogram
hist(test1$log.price - pred_x, breaks = 30, main = "Histogram of residuals",
    xlab = "residuals")
```



Part f Estimate the mean squared errors for your champion model with the in-time and out-of-time test sets. Compare these numbers with that you obtained with the validation set in part c. Is the MSE estimated with the out-of-time test set bigger or smaller than the MSE estimated with the in-time test set? Is this result expected? Explain why.

MSE estimated with the in-time test 0.03675328. MSE estimated with the out-of-time test 0.04914106. The validation MSE is greater than in-time test but is close while it is less than the out-of-time test. The MSE estimated with the out-of-time test set bigger than the MSE estimated with the in-time test set. This is expected because the model was built based on observations that are similar to the ones in the in-time test set.

```
# in-time test set
pred_x_in <- predict.reg(bestf, test1, id = 84)
mse_in = mean((test1$log.price - pred_x_in)^2)
mse_in
[1] 0.03675328
# out of time test set
pred_x_out <- predict.reg(bestf, test2, id = 84)
mse_out = mean((test2$log.price - pred_x_out)^2)
mse_out
[1] 0.04914106
```

Model interpretation

Question 7

For the following model answer TRUE or FALSE for the questions below; no explanation is required and you will only be graded based on the TRUE and FALSE part of your answer even if you provided explanations.

```
Call:
lm(formula = log.price ~ f.cond + f.waterfront:sqft_living, data = mod.variables)

Coefficients:
              (Intercept)                   f.cond2
               12.0196704                -0.0375171
                  f.cond3                   f.cond4
                0.1930924                 0.2257954
                  f.cond5  f.waterfront0:sqft_living
                0.3423396                 0.0003910
f.waterfront1:sqft_living
                0.0005391
```

(i) **TRUE or FALSE** The model above gives different y-intercepts for houses in different conditions.

TRUE

(ii) **TRUE or FALSE** The model above gives different slopes depending on if the house has a view to a waterfront or not.

TRUE

(iii) **TRUE or FALSE** To predict the `log.price` of a house in condition 3 that does not have a view to a waterfront and has 1000 sqft we can use this:

Predicted log.price $= 0.1930924 + 0.0003910 \times 1000$.

FALSE