ECS 132 Winter 2019 - Term Project

Eric Li (914752020)
Benjamin Bing (914092968)
Zhiyuan Guo (914039235)

March 21, 2019

Contents

A Problem A

A.1 Eric Li (914752020, ercli)

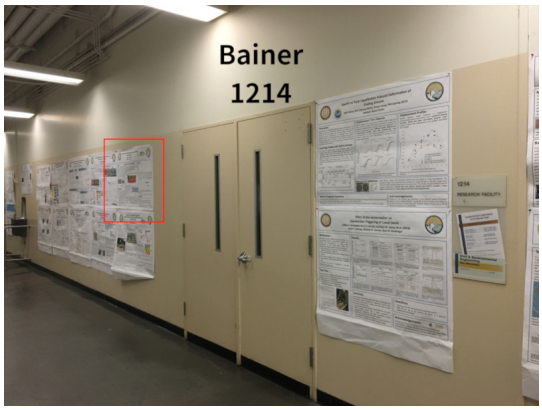

Image ercli-loc

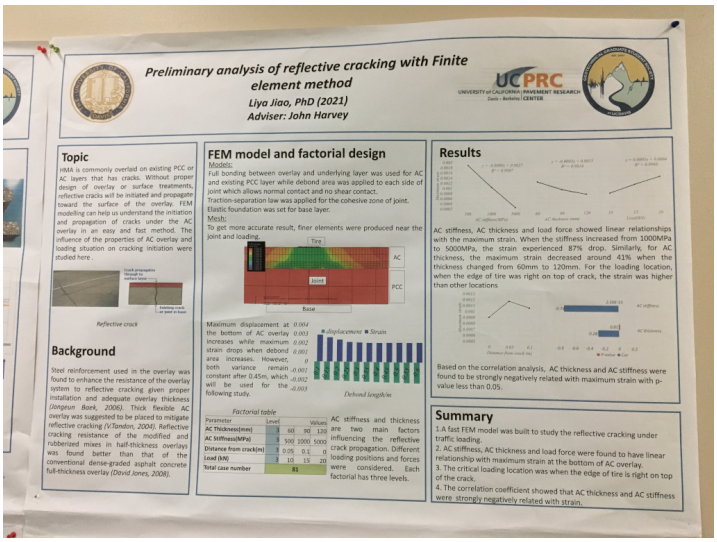

Image ercli-pos

The poster above (Image ercli-pos) is found near Bainer Hall 1214 (Image ercli-loc). This is the first poster in the about 30 posters I viewed that directly contains the word "p-value". This research is conducted under Department of Civil and Environmental Engineering. It studies some properties of something called AC layer which tries to prevent cracks.

This research uses p-value in the Results section (Image ercli-res). The student plotted the relation between Maximum strain and parameters on AC (stiffness and thickness) and used a linear model to approximate the result. She then calculated the correlation (seems to be a different definition than the correlation in chapter 10, see next paragraph) of the AC parameter and maximum strain and its p-value, then concluded emphasizing that the p-value is less than 0.05.

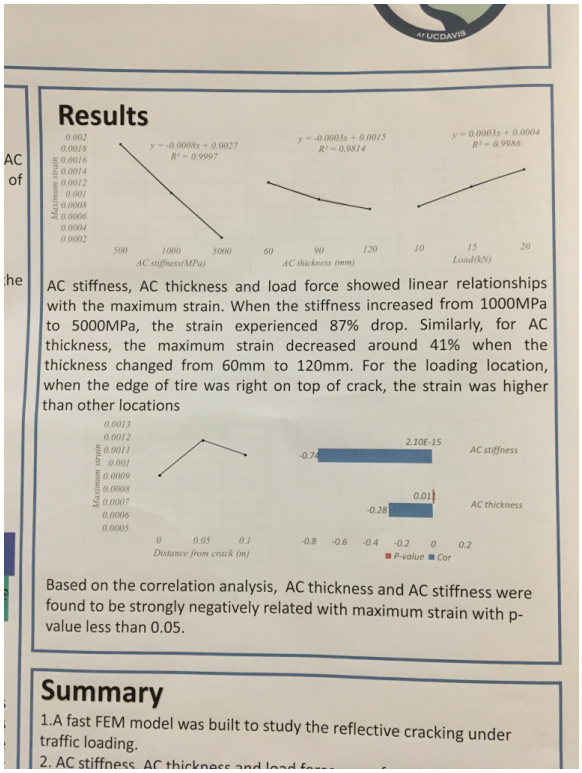

Image ercli-res

A problem with the data set is that the correlation seems to be incorrectly computed. For AC stiffness, my program (see appendix I.1) gives -0.90 but the report gives -0.74. For AC thickness, my program gives -0.99 but the report gives -0.28. The reason may be that this data cannot be analyzed linearly.

I believe this report can be made better if confidence intervals are used instead of p-values. As "From Algorithms to Z-Scores: Probabilistic and Statistical Modeling in Computer Science" by Norm Matloff says on page 255, the use of p-value here is very misleading. For example, the p-value is 2.10×10^{-15} for AC stiffness, giving readers a feeling that this approximation is extremely precise. However, considering the fact that there are only 3 data samples, the population value can be "far" off -0.74. In this case, using a confidence interval can be more informative and free-of-confusion. For example, by saying that "the 95% confidence interval is $(-0.739, -0.741)$" (I am making up the numbers, because I cannot get original data so cannot do the actual calculation).

A.2 Benjamin Bing (914092968, jbing)

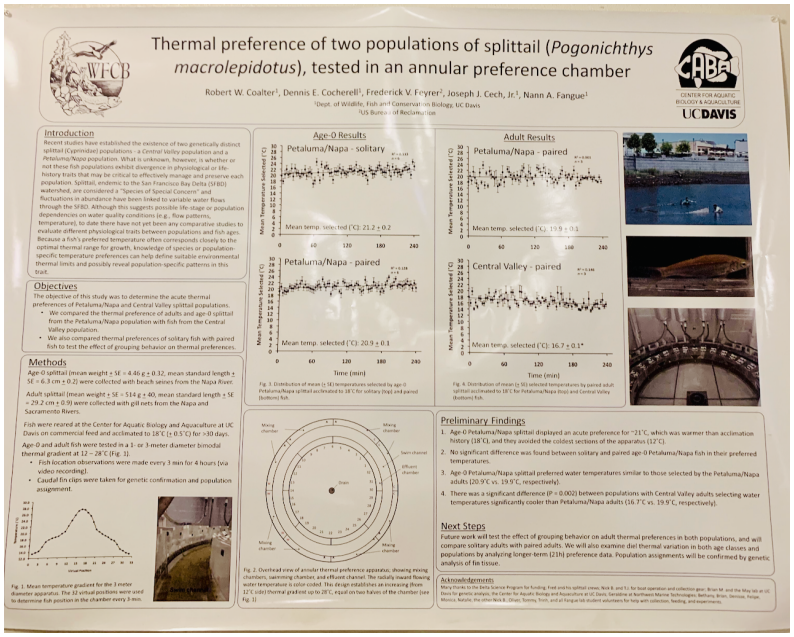

Image bing1

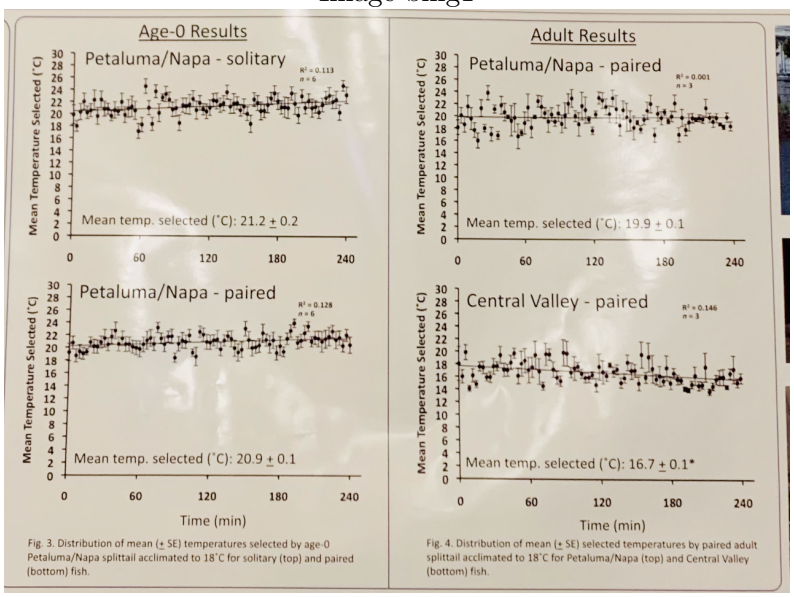

Image bing2

The research poster is at the first floor of the Academic surge. The current scientific topic being investigated is the determination of Petaluma/Napa and Central Valley splittail populations' acute thermal preferences. Comparisons were made between the adults and age-0 splittail thermal preferences. Comparisons of thermal preferences were also made between solitary fish and paired fish to determine the grouping behavior on thermal preferences. The results showed that Petaluma splittail preferred 21°C. There was no significant difference between solitary and paired age-0 Petaluma fish. The results also revealed that a significant difference existed between adult Petaluma or Napa and Central valley adults with p-value $= 0.002$. In short, all the study objectives were covered.

P-value and confidence intervals (CI) are crucial statistical concepts that play critical roles in decision making. Understanding these concepts is necessary for the assessment of scientific articles

or research. While both p-values and CIs are used in key decision making, certain instances arise when confidence intervals provide more meaningful and insightful analysis. The use of confidence interval provided insightful analysis than the p-value. CIs revealed the possible effects or values that were present in the population. Unlike p-value that only showed significance, CI presented the values that were likely to be defined in the population. In the current case, for instance, the use of CIs revealed the range of temperatures for the Petaluma Napa population and Central value paired populations. From the information provided by the CIs, one can precisely tell which populations prefer cooler temperatures, warm temperatures, or the populations that deviate from the normal expectations. The reliance on p-values brings two dangers. First, testing independent hypotheses like in the current case using p-values increases the risk of false positive. In this case, the research may incorrectly report significant differences. Lastly, given that p-value is a function of the sample (n), it might provide a misleading conclusion in case n was small to detect a population effect. In sum, the confidence interval tends to be a more robust decision-making tool than p-value.

In conclusion, there are more reasons to use CIs in statistical decisions than p-value. The current case study revealed that provides meaningful and more insightful analysis than p-value because it gives a range of possible values in the population. It also minimizes the risk of false positive.

A.3 Zhiyuan Guo (914039235, zhyguo)

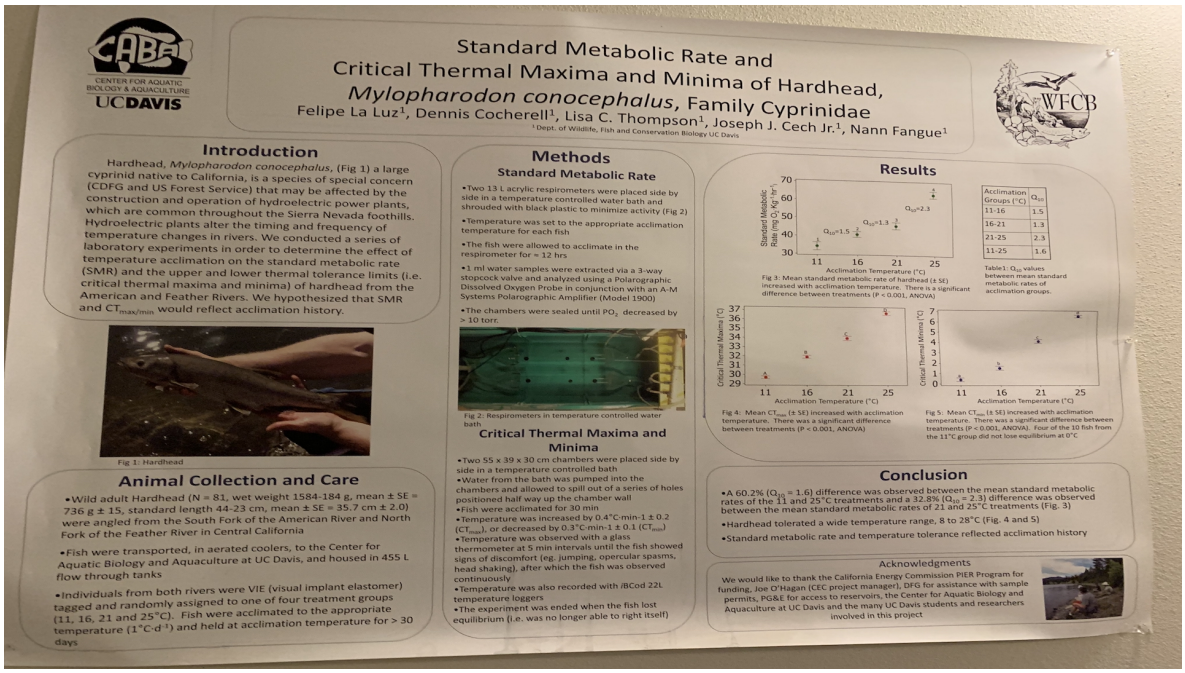

Image guo

The poster was found at the first floor of the Academic Surge.

To test whether hardhead, a large cyprinid native to California, is affected by the construction and operation of hydroelectric power plants that can alter the timing and frequency of temperature changes in rivers, a group of UC Davis Biology students conducted a series of experiments with a sample of wild adult hardhead (N=81) to determine the effect of temperature acclimation on the standard metabolic rate and the upper and lower thermal tolerance limits of hardhead.

They found that there was a significant difference ($Q_{10} = 2.3$) between the mean standard metabolic rates of 21°C and 25°C treatments at $p < 0.001$ level. However, such a reliance on significance testing is dangerous and misleading. First of all, is the difference really "significant"? It is true that 2.3 is away from zero, but is it sufficient to conclude that the 25°C group had a significantly different metabolic rates than the 21°C group? It may be the case that such variation is totally acceptable. The results of this experiment could be more convincing if a confidence interval is given instead, where experimenters, instead of significant test itself, make the decision on whether such variation is significant. Second, if the sample size were to be large, then the result could have become "significant" no matter whether it is truly significant or not. Fortunately, sample size of 81 in this case would not have such a problem, which may result in misleading conclusions in other cases.

B Problem B

B.1 Data import

We first wrote the `split_data` function in `ProbB.R` (II.11), which reads in the input file (`listings.csv.gz`) and breaks the data into training and test sets as required in the prompt. The two sets are returned inside a list (say I save the result in parameter `tt`).

B.2 Programming Languages and Libraries Used

As required, we are mostly using R to program in the term project. However, I (Eric Li) wrote some control programs in Python 3 in order to perform parallel computation (in B.6).

The project uses the following R libraries

- `reticulate` is used in B.6, which allows R to call Python functions and get the return value.
- `ggplot2` is used to plot most graphs in this project
- `plotly` is used to draw an interactive 3D-plot.
- `processx` is used to save image in plotly.
- `ggmap` is used to draw the heat map.
- `RColorBrewer` is used to get color in R.

B.3 Deal with messy data

The next thing to do is removing useless / less useful information. For example, the long descriptions and urls are not very useful (admittedly, whether the description contains "no smoking" may be a useful factor, but I am dropping this information for now).

B.3.1 Drop less useful columns

Using `names(tt$trndta)`, we can get the list of all data entries names. There are 106 of them (see appendix II.1 for results). Now we filter the ones we need out.

In field 1 to 4, `id` is the ID of the entry on AirBnB's website. This one is not important. `listing_url` can also be dropped. `scrape_id` and `last_scraped` seems to a update time stamp, and can be dropped.

Field 5 to 8 (`name`, `summary`, `space`, `description`) are too objective (contains description of the house), so they are dropped.

All entries in field 9 (`experiences_offered`) are `none`, so they are dropped.

Most of field 10 to 15 (`neighborhood_overview`, `notes`, `access`, `interaction`, `house_rules`) are also objective information, so they are dropped. However, we decide to grep some keywords in `[12] transit`, to decide whether the listing is close to public transportation resources (e.g. Bart, Muni, Bus, Shuttle). We dropped Uber and Lift because we believe most listings, even not mentioned in the transit section, have access to the service because they are in San Francisco (manually confirmed by plotting their longitude and latitude).

Field 16 to 19 are url information, and they are dropped.

Field 20 to 37 contains information about the host, and most of the information is useless. By experimenting with the data, `[29] host_is_superhost` seems to be an important factor, so is included. Other potential fields include `[23] host_since` (when the host joins), `[26] host_response_time`, `[27] host_response_rate` (indicates how responsible the host is). However, these data are still difficult to be processed, so they are dropped.

Field 38 to 51 contains address information. These information are difficult to be made useful, so are dropped. (an idea is that there may be some "hot-spots" where price is expensive, so longitude and latitude can be used to analyze. However, this data is still dropped at this stage)

Things get more interesting starting from field 52. `property_type` seems to be a category of the house, but the values are too discrete (e.g. only three entries is "Tiny house"), so we do not consider them for now.

`[53] room_type` contains information including whether the room is shared. This seems to be an important factor about the price, so we will include it. To process this kind of discrete value, we use indicator random variables. That is, we use a column of booleans `room_type_entire` to denote whether it is an entire room, and `room_type_private` to denote whether it is private. The only other case, which is shared, can be implied if the previous two columns are both FALSE.

`[54] accommodates` is also an important factor, so we will include it.

`[55] bathrooms`, `[56] bedrooms`, and `[57] beds` are important factors on price, so we will include it.

`[58] bed_type` is an important price, but we noted the fact that almost all bed types are "Real Bed". So we drop this.

`[59] amenities` records the services provided. The data is important, but unfortunately, is in string format. However, we can extract some useful information, say whether "Wifi" or "Breakfast" is contained in the string. A tricky part is that we have to use regular expression, like `grep('[\\,]Breakfast[\\,]', tt$trndta$amenities)`, or we will match unwanted entries like "Breakfast table".

`[60] square_feet` is also an important factor that should be recorded. However, there are too many NA entries, so we dropped it.

`[61] price` is the factor to be predicted. A problem is that the input is a string starting with `'$'`, but we need a numeric data. We wrote a function `priceToNum` that changes price format to numeric by replacing `'$'` and `','` using regular expression, and then cast to numerical type.

`[62] weekly_price` and `[63] monthly_price` are not allowed to use. Though by plotting we can see that these are strongly related to `[61] price`. The code for plotting is included in appendix II.3 in function `plot_relationship`. This function can plot a scatterplots while filtering out values that are too large.

`[64] security_deposit` and `[65] cleaning_fee` are money-relevant, so should be considered. However, some entries are blank on these columns, so a way to deal with the lack-of-data should be developed.

`[66] guests_included` and `[67] extra_people` are also important factors of price, so are included.

`[68] minimum_nights` and `[69] maximum_nights` are also important. However, some data are problematic. For example, a host's minimum night requirement is 100000000 [1], which is very unrealistic. We believe that after a threshold, the growing of minimum nights does not really make sense, so when minimum nights is greater than the threshold, we truncate it to the threshold

Field 70 to 75 are some more data related to long-term staying period requirement. We are dropping them for now.

[1] https://www.airbnb.com/rooms/15344978

[76] calendar_updated is dropped, because it does not seem very relevant to price.

[77] has_availability is dropped because all entries are 't'

Field 78 to 81 are about availability. These data can reflect the popularity of the house (e.g. less availability means more popular), but can also caused by the host not adding many available days. So this factor is dropped, since [84] number_of_reviews_ltm can also indicate whether the listing is popular.

[82] calendar_last_scraped seems irrelevant, and are all the same (all 2019-02-01), so this field is dropped.

Field 83 to 93 and 106 are about reviews. However, some listings does not have enough data. We decided to keep the one with full data ([84] number_of_reviews_ltm, which is relevant to whether the listing is popular), and one entry, [87] review_scores_rating, which is the overall review score of the listing. For listings with no reviews, review_scores_rating is NA, and we assume that people's attitude the listing is neutral by default, so we decided to assign the mean value to these NA fields. We dropped all other data.

Field 94 to 96 and 100 to 101 are some license / profile requirements. They are dropped.

[97] instant_bookable and [99] cancellation_policy are important factors, so they are saved. We assigned some scores based on the rule of cancellation on the offcial website page

[98] is_business_travel_ready is always 'f', so is dropped.

The meaning of field 102 to 105 is unclear, so they are dropped.

[106] reviews_per_month is dropped because we already have [84] number_of_reviews_ltm.

B.3.2 Drop less useful columns summary

In summary, the following fields are preserved or may be preserved (at this stage). There are 23 fields in total.

ID	field_name	comment
[8]	description	undecided
[12]	transit	search for "BART", ...
[23]	host_since	
[26]	host_response_time	
[27]	host_response_rate	
[29]	host_is_superhost	
[35]	host_verifications	
[37]	host_identity_verified	
[39]	neighbourhood	
[52]	property_type	
[53]	room_type	use indicator random variables
[54]	accommodates	
[55]	bathrooms	
[56]	bedrooms	
[57]	beds	
[59]	amenities	look for "Breakfast", ...
[65]	cleaning_fee	
[67]	extra_people	
[68]	minimum_nights	
[84]	number_of_reviews_ltm	
[87]	review_scores_rating	
[97]	instant_bookable	
[99]	cancellation_policy	convert to score

We wrote function drop_data to remove data decided useless in the previous section. This R function simply selects only data that we need to use. After that, the function filter_colomns

breaks data to numerical format. For example, for field `cancellation_policy`, it converts the text description to the score discussed last section. For field `room_type`, indicator random variable is used in substitute of the description string.

Note: This was only a "rough" drop of variables. Some of the variables dropped initially were included back later, and we will discuss why in the following section.

B.4 Data analysis

In order to build our model, we made use of linear regression in which the response variable is the airbnb price, $Y = P$, and the predictor variables, $X = X^{(1)}, ..., X^{(r)}$, are to be decided by ourselves.

$$m_{P;X(t)} = E(P|X = t)$$

We are interested in $m_{P;X(t)}$, the mean of airbnb price in the population of all airbnb houses for which $X = t$ [2].. We assume that $m_{P;X(t)}$ in the population is a linear function of t.

$$m_{P;X^{(1)},...,X^{(r)}}(t) = \beta_0 + \beta_1 t_1 + ... + \beta_r t_r$$

By collecting data from sample, we can get our estimates for the population regression coefficients for each predictor variable ($\hat{\beta}_i$) and thus form our model. We used cross validation to test our model by breaking data into Training and Test sets. We ran lm() on our training set to set up our linear regression model, then predicted on the test set, and finally compared the predicted prices to the real prices to get our prediction error.

Using ideas from Kaggle.com, prices that are relatively high would make our model very inaccurate if we include them. So we only include rows in training set that have price less than 700 when forming our lm model.

```
trndta = removeOutlierPriceFromtrndta(trndta)
removeOutlierPriceFromtrndta <- function(trndta)
{
  # remove rows in trndta where price is greater than 700
  pr = trndta$price
  pr = as.numeric(gsub('\\$|,', '', pr))
  # from kaggle
  trndta = trndta[pr <= 700, ]
  trndta
}
```

Notice that we can not do the same to our test data, in which we "don't know" the prices.

We started our deeper analysis with `security_deposit` and `cleaning_fee`. We first thought that these two predictors were both very important, as they may imply how much the hosts values their houses and thus affected the `price`. We also thought that they should be strongly correlated with each other so that we had to drop one of them, otherwise one would be almost a linear function of the other so that $(Q'Q)^{-1}$ would be numerically unstable [3]. However, when we did a scatter plot of `price` against `security_deposit`, there was no clear relationship as those data points were just all over the place (see Image price-sd). Then, we plotted `price` against `cleaning_fee` and saw a strong positive correlation there (see Image price-cf), with low `cleaning_fee` corresponding to low `price` and high `cleaning_fee` corresponding to high `price`. Therefore, we abandoned `security_deposit` and kept `cleaning_fee`.

[2] See chapter 14 for more details in Norm Matloff's textbook

[3] See section 14.15.3 for more details in Norm Matloff's textbook

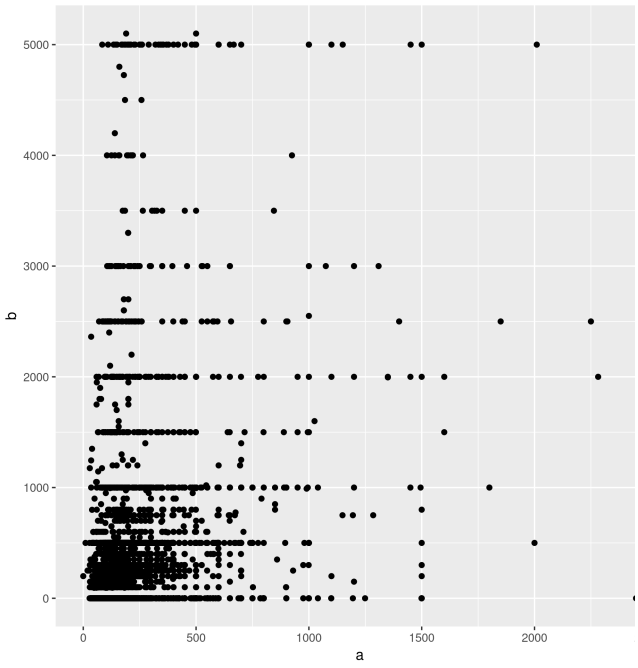

Image price-sd

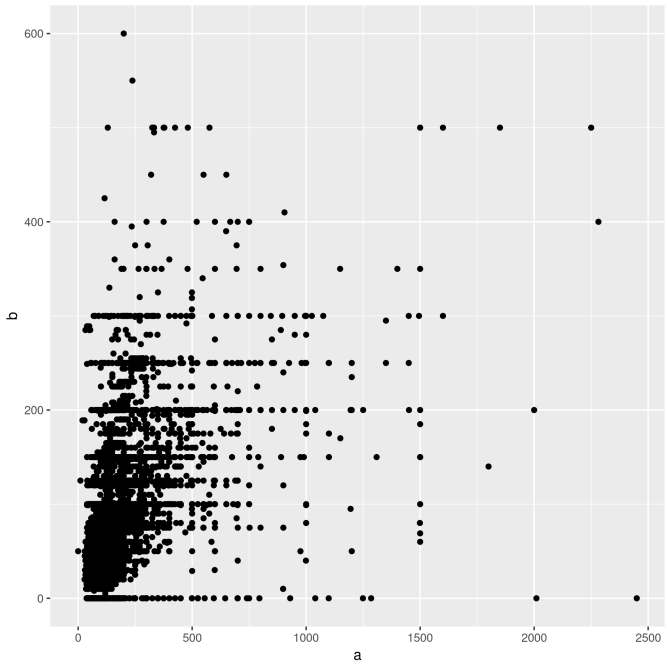

Image price-cf

We made `host_is_superhost` and `host_identity_verified` dummy variables. We found that `host_identity_verified` did make our prediction better on training and test sets, but `host_is_superhost` had some problems. If we exclude `host_is_superhost` from our model and predict training set (predict itself), we would have a smaller prediction error. But if we exclude `host_is_superhost` from our model and predict the test set, we would get a larger prediction error. The reasoning behind this scenario may be p-hacking, which we will discuss later. We decided to drop `host_is_superhost` because of this unstable behavior. Note that even though `host_identity_verified` "behaved well" in both sets, there is no guarantee that `host_identity_verified` would always do good in future(other) test sets again due to p-hacking. After we dropped `host_is_superhost`, `host_identity_verified`'s coefficient estimate still had relatively large variance, with variance $(2.195)^2$ greater than estimate 4.082 itself! The 95% CI for its coefficient estimate is $4.082 \pm 1.96(2.195) = (-0.22, 8.38)$. This interval is relatively wide. This gave us another reason to stay cautious on `host_identity_verified`.

```
Coefficients:
                               Estimate Std. Error t value Pr(>|t|)
(Intercept)                  -1.623e+01  3.963e+01  -0.410 0.682154 ***
... (other variables not included) ...         ...      ...      ...
host_is_superhostTRUE         8.121e+00  2.224e+00   3.652 0.000263
host_identity_verifiedTRUE    3.755e+00  2.033e+00   1.847 0.064752
... (other variables not included) ...         ...      ...      ...
---


Coefficients:
                               Estimate Std. Error t value Pr(>|t|)
(Intercept)                  -1.623e+01  3.963e+01  -0.410 0.682154 ***
... (other variables not included) ...         ...      ...      ...
host_identity_verifiedTRUE    4.082e+00  2.195e+00   1.859 0.063005
... (other variables not included) ...         ...      ...      ...
```

```
---
```

Then, we used two dummy variables, `room_type_entire` and `room_type_private` to denote `room_type` that can be {Entire, Private, or Shared}. For example, if `room_type_private` has value 1, then `room_type` is private. If both of them are zeros, that means `room_type` is shared.

```
Coefficients:
                              Estimate Std. Error t value Pr(>|t|)
(Intercept)                 -6.534e+00  3.958e+01  -0.165 0.868882 ***
...                                ...        ...     ...      ...
room_type_entireTRUE         9.073e+01  6.965e+00  13.027  < 2e-16 ***
room_type_privateTRUE        4.896e+01  6.840e+00   7.158 9.18e-13 ***
...                                ...        ...     ...      ...
---
```

The 95% confidence intervals for their coefficient estimates are (77.079, 104.3814) and (35.55, 62.366). The lower bounds of both both intervals were far from zero. They were behaving really well no matter on predicting training set itself or test set.

Variables `bathrooms`, `bedrooms`, and `beds` should be good predictor variables, since they imply the size of the property(house, apartment, etc.). The more rooms there are, the higher the price should be. After a few trials, we found that the prediction error was minimized when we excluded `bathrooms`. The reason might be as follows. First, `bathrooms` was closely correlated with `bedrooms`, meaning that `bathrooms` was a rough linear function of `bedrooms`. Second, unlike `bedrooms` and `beds`, the support of `bathrooms` includes decimals such as 0.5, 1.5, and 2.5.

```
> table(lsts$bathrooms)

  0  0.5    1  1.5    2  2.5    3  3.5    4  4.5    5  5.5    6    8   10
 41   17 5210  465  996  173  119   54   34   10   24    1    6   16   11
 14
  1
```

It may be true that 2 bathrooms have double the impact of 1 bathroom on price. However, 1.5 bathrooms certainly do not have 1.5 the impact of 1 bathroom, since there is no much difference between them (that extra 0.5 means a toilet and a sink). This gave us reasons to drop `bathrooms` and keep only `bedrooms` and `beds`. To see whether the effect of `beds` on `price` is greater at larger `bedrooms` values, we added an interaction term, `bedstimebedrooms`, the product of `beds` and `bedrooms`. It turned out that our prediction error did decrease a little bit after we added it.

We suspected that price would be affected by how experienced the host was. So we use current date to minus `host_since`, giving us the time length. We took the log of it, because we believed that the effect of time length on price is increasing at a smaller rate as time length goes large. We called this new variable `host_since_log`.

Variable `number_of_reviews_ltm` is negatively correlated with `price`. Its slope estimate has a 95% confidence interval of (-0.564, -0.356). It suggests that more popular hosts tend to have lower price. What this actually implies, I think, is that people tend to prefer hosts with lower price. Interestingly, after we included `number_of_reviews_ltm` in our model, it resulted in the coefficient estimate of `host_since_log` changing from -0.116 to 0.81.

```
Coefficients:
                              Estimate Std. Error t value Pr(>|t|)
(Intercept)                  8.861e+00  4.105e+01   0.216 0.829112
host_since_log              -1.158e-01  1.395e+00  -0.083 0.933865
...                                ...        ...     ...      ...
```

```
---

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                -1.251e+01  4.087e+01  -0.306 0.759601
host_since_log              8.148e-01  1.390e+00   0.586 0.557886
...                               ...        ...     ...      ...
number_of_reviews_ltm      -4.609e-01  5.294e-02  -8.706  < 2e-16 ***
...                               ...        ...     ...      ...
---
```

Again, ... means other variables not shown here.

The above implies that experienced hosts tend to have more reviews. This makes sense because the longer you have already been a host, the more transactions you have already made and thus more reviews you have received. Since `number_of_reviews_ltm` and `host_since_log` are correlated, should we even keep `host_since_log`? Earlier, we saw that `host_since_log` did play a decent role as a predictor. Now, however, we realize it may be the case that it was `number_of_reviews_ltm`, rather than `host_since_log`, that truly affected the price. By taking out `host_since_log` from our model, we found that our prediction error decreased. This confirmed our suspicion, so we would exclude `host_since_log` from our model.

Although we already have `number_of_reviews_ltm` to indicate how popular a host is, we may also need to include `review_scores_rating` because some hosts may have many reviews but with relatively low rating. After a quick check using table, we saw that almost all `review_scores_rating` values were greater than 80.

```
> table(lsts$review_scores_rating)

  20   30   40   50   56   60   64   67   68   70   72   73   74   75   76
   8    1    4    1    1   43    1    3    1   22    2    4    2    6    7
  77   78   79   80   81   82   83   84   85   86   87   88   89   90   91
   4    6    3  125    7   13   21   19   27   41   62   63   72  164  103
  92   93   94   95   96   97   98   99  100
 160  260  238  354  434  581  699  634 1656
```

We decided to introduce the square of `review_scores_rating` so that the effect of rating on price is greater at higher rating levels. We found that this indeed brought down our prediction error. Notice that when we added this interaction term, we ran a risk of overfitting, which we would address later.

Some dummy variables were really useful. For example, the distribution of `minimum_nights` was quite interesting. About half of them were smaller than 29.

```
> table(lsts$minimum_nights)

   1    2    3    4    5    6    7
1256 1471  823  285  185   35   74
   8   10   12   13   14   17   18
   1    1    1    1    3    1    1
  20   21   24   25   28   29   30
   1    1    1    2    3    1 2761
  31   32   35   40   45   50   55
 126   32    2    3    4    3    1
  58   59   60   62   75   80   85
   1    1   35    1    1    1    1
  90  100  120  140  150  170  180
  29    1    6    1    1    1   19
 183  188  200  256  298  302  360
```

```
  2         1         1         1         1         1         3
365       999      1000      1125 100000000
  6         1         1         1         1
```

We found that prices with `minimum_nights` less than 29 were quite different from those with `minimum_nights` more than 29. So we made a new dummy variable `mini_night_lessthan29`. Not surprisingly, this variable behaved well in our model with 95% CI of slope estimate (15.631, 24.029) whose lower bound was far from zero and decreased our prediction error.

We believed that we should take the square root of `accommodates`, since the effect of the number of guests on price would be smaller from 10 guests to 11 guests than from 1 guest to 2 guests.

We also thought that price should be largely affected by the location of house, so we mapped each zipcode to the median housing price in that area using data from `https://www.zillow.com/home-values/`. In total 198 rows in our data did not have zipcode and some zipcodes did not have corresponding housing price on that website, so we replaced them with the mean housing price among others. We named this new variable `zipPrice`. Notice that even though the 95% CI of `zipPrice`'s coefficient was really close to zero, it was coming from `zipPrice` values being large. The median of the support of `zipPrice` is 1367050! Thus, we still believed that `zipPrice` was worth using as a predictor.

```
Coefficients:
                                  Estimate Std. Error t value Pr(>|t|)
(Intercept)                     -1.251e+01  4.087e+01  -0.306 0.759601
...                                    ...        ...     ...      ...
zipPrice                         2.609e-05  2.795e-06   9.333  < 2e-16 ***
...                                    ...        ...     ...      ...
---
```

The law of demand states that, other things equal, the higher the price, the less quantity demanded. The demand should be a very strong predictor for price. Fortunately, we have `availability_30` and `availability_365`, telling us how many days the house is available(i.e. not yet demanded) within 30 days(short-run demand) and 365 days(long-run demand).

We also had a distance model that would be explained in B.5.

B.4.1 Dropping variables again

We found that several variables included earlier were not quite useful. We tried to make a dummy variable called `amenities_breakfast` to look for rows in amenities column that had the feature breakfast. However, only about 10% of the rows had that feature, and that variable had negligible impact on our prediction, so we decided to not use `amenities_breakfast`. If we were to make every feature listed in `amenities` a dummy variable, we would run a risk of overfitting as our number of variables would increase a lot.

As for `transit` and `cancellation_policy`, we tried to give each feature a weight instead of making them dummy variables, assuming some feature is better than others to some ratio. For example, if an entry in `transit` contains the word(string) "bus" or "shuttle", we assigned to that entry value 2. If an entry contains the word "bart" or "muni", we assigned to that entry value 3.

However, they did not fit well in our model, only to bring up prediction error a lot, and using dummies instead did not help either, so we had to give up on them.

Some of the decisions are based on advises from the result of section B.6. For instance, we removed `host_response_rate`, discussed in B.6.1.

B.4.2 Dealing with NAs

```
> sum(is.na(lsts$review_scores_rating))
[1] 1346
> sum(lsts$cleaning_fee == '')
[1] 775
> sum(is.na(lsts$beds) )
[1] 5
> sum(is.na(lsts$bedrooms) )
[1] 1
> sum(is.na(lsts$bathrooms) )
[1] 20
```

Notice that `review_scores_rating` has about 1/7 NAs and `cleaning_fee` has about 1/10 NAs . We replaced those NAs with the median of others.

B.4.3 Interaction Terms, p-hacking

We added a few interaction terms to our model. After adding all of them, our number of variables had come to 32, which was still far from 80, somewhat indicating that we are not overfitting. However, one should always be aware that even though the bias of our model decreases, the variance increases as we add more and more terms.

We introduced the square and the cube of `bedrooms`, as we believed that the effect of `bedrooms` on price would increase at a higher rate as `bedrooms` go large. The same reasoning applied to `extra_people` when we introduced `extra_people_2` and `extra_people_3`.

We introduced `clnfee_times_minnit29`, the product of `cleaning_fee` and `mini_night_lessthan29`, as Eric will explain it in B.6.2. He will also talk about why we introduce `dist2wecen_times_accom` and `dist2sncen_times_accom` in B.6.3.

After adding those terms, we did find out that our prediction error decreased. However, it does not necessarily mean that our model now is definitely better than previous version. It may be the case that this model happens to fit well in the test data due to p-hacking.

B.4.4 Result

Finally, we have our regression function ready.

$$mean\ price = \hat{\beta}_0 + \hat{\beta}_1 \texttt{host\_identity\_verified} + ...$$

$$mean\ price = -6.534 + 4.582t^{(1)} + ...$$

For a detailed summary of our lm model, check Appendix II.2.

In total, we used 31 predictor variables.

Using our regression model, our mean absolute prediction error for the test set is 68.40.

B.4.5 Code

Check II.11

Simply call `mymain()` to run our program.

B.5 Creating Hot-spot and Distance Model

We believe that the location plays an important role in predicting the `price`. Thus, we try to keep the `longitude` and the `latitude`, and visualize through graphs. Everything of the graphing the `longitude` and the `latitude` is done in the file **graph.R** (II.10)

The first thing that came to my mind is to draw a 3D scatter plot. I used the **plotly** package to draw achieve that, the reason I use it is that it is an interactive plot, in which we can zoom in and change the camera view angle. In the graph , x-axis is the `longitude`, and y-axis is the `latitude`. First, I did not filter data, and put all of the prices into the plot, which makes the range too big, and it is hard to find a tendency. Then I only picture the data whose `price` is smaller than 1500. The result is Image scatter3d.

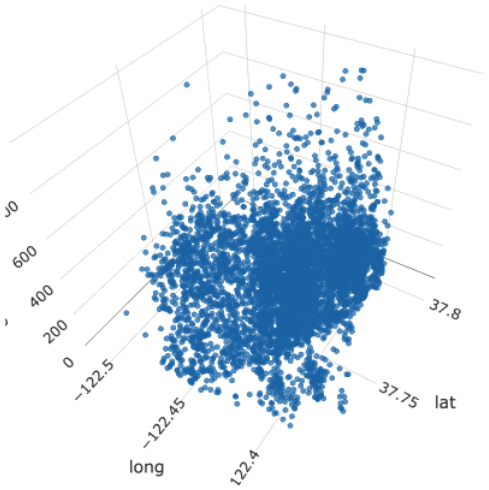

Image scatter3d

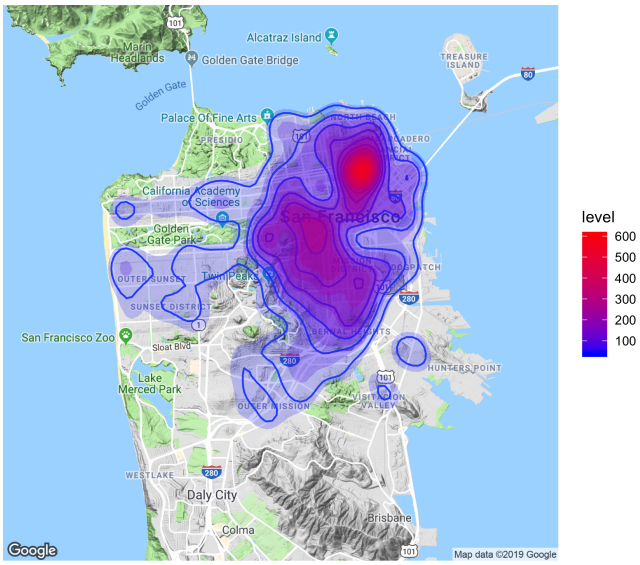

Image heatmap

It turns out that it is still hard to find a tendency in the 3D scatter.

Then I think a heat map might be helpful for us to find the relationship between `price` and geological position. I did some research online, and it is cited in the comment of **graph.R**.

I pulled the map of San Fransico from Google map, and still x-axis is the `longitude`, and y-axis is the `latitude`, visualizing the density and coloring it. The result is Image heatmap.

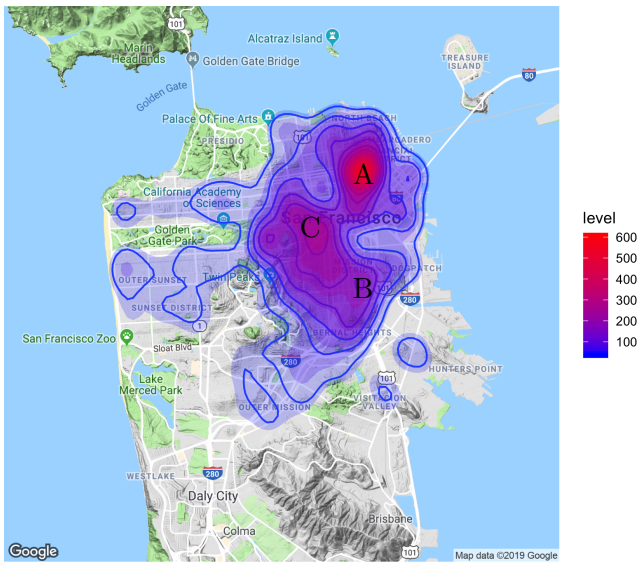

Image heatmap-labeled

B.5.1 Update the distance model

After drawing the graph, we can see that there is two major hot-spots. One is point-like; another is more diverge and and locates at the south-west of the first one. We first drew one point on the center of north-east $A = (-122.405556, 37.792123)$ one and two points on the south-west one $B = (-122.410524, 37.755024)$ and $C = (-122.427180, 37.764424)$ (see Image heatmap-labeled; note that the locations of the letters in the image are not precise). Then calculate the minimum distance to any of these points; use this distance as an estimator.

However, this model of distance is not perfect. For the point on the line segment $\overline{BC}$, its distance to the south-west hot-spot should be considered 0, but is actually not. So we used some algorithms in computational geometry, which are introduced in chapter 33 of "Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.

First, I generalized the works to do to two functions: `dist2point` calculates the listing's distance to a fixed point; `dist2line` calculates the listing's distance to any point on a given line segment. Calculating `dist2point` is trivial, but `dist2line` requires different cases. First I wrote some helper functions that calculates the distance between two points (`distance2` and `distance`) and the cross and dot product of two vectors (`cross` and `dot`). The function `same_side` uses cross-product to decide whether two points are on the same side of a line. Using `same_side`, The listings can be divided into three cases:

1. The location is closest to point B.
2. The location is closest to point C.
3. The location is closest to somewhere on the line segment.

For the first two cases, we can just use distance to point B or C. For the third case, we can use the dot product to find the actual closest point, then calculate the distance.

By using vectors to calculate, the algorithm minimizes the use of division when doing geometric computations, so it is likely to produce more precise result (e.g. round off error is eliminated).

B.6 Sub-project: Testing matching fields in parallel

I started the following "sub-project" partly because I was misled by a group member. He told me that running `lm` is relatively slow, and I thought that we should send the computation request to other computers to increase efficiency. However, it turns out that `lm` is not that slow, so sending the computation request may slow down the whole program.

Most of the program in this project is written in R, but the master program that manage all processes over all computers is written in Python 3, as making concurrent programs is more convenient in Python. A potential alternative solution is discussed below, but is unfortunately not used.

First I modified the major analysis program's interface such that it is easy to change the rows used to fit the linear model by passing a simple vector. For example, the function `makeLmModel` takes the `testCol` argument, which specifies which columns to use when creating the linear model. Then using the following program (included in `Distrib.R` in appendix II.4), I can write a program that reads a list of numbers from stdin, and convert them to a vector of numerical values.

```
f = file('stdin')
open(f)
s = strsplit(readLines(f,n=1), ' ')[[1]]
a = as.numeric(s)
```

After that, `Distrib.R` calls the function to create linear model and calculates the precision of prediction (mean absolute prediction error) by calling the function `scorePredic`.

`trntstdta.Robj` is the R objects `trndta` and `tstdta` after `filter_columns` is performed on them. Thus, we can save time reading the entire csv file and filtering the data.

In function `p`, `Distrib.R` prints the string "START" and "OK", with the answer in between. For example, if the answer is 80.1234, then the program will print the following lines near the end of its output, which allows the Python program to locate the answer easily.

```
[1] "START"
[1] 80.1234
[1] "OK"
```

After the transformation, it is easy (for most programming languages) to execute the program and use stdin and stdout as interface. (Though consulted with Professor Matloff, I decided not to use reticulate, at this stage, because I still have to deal with stdin / stdout over ssh. A solution is to write a server using R, but this can lead to security problems, so is not considered yet). The master program need to run `Distrib.R` on a remote client using ssh, and specify the list of columns to use in stdin. The slave program will do the computation and print the result. After the slave program exits, the master program only needs to search for "START" and "OK" and can get the result.

Here, I chose to use Python to send all computations to remote host. The program is `distrib.py` (in appendix II.5). It generates the combinations that we need to test using function `create_data`, and for each test create a `CSIFThread`. The thread executes the system command `ssh`, feed in the input, parse the output, and put the result into the global variable `ans`, a dictionary. This result is saved periodically to a file using `pickle`. I used a lot of locks to make sure that the program runs correctly.

I decided to make use of reticulate when analyzing the data collected by Python. I wrote a simple function `read_data` in `distrib.py`, which reads the `ans` saved by `pickle` in, do some

basic transformations (i.e. convert from map to list, which makes it easier to process the data), and return the data. `DistribAnalyze.R` (in appendix II.6) uses `reticulate` to call `read_data` and receive its return value. However, the default data type received in R is a list of lists, instead of a data frame (which is more desirable, see example below). So I changed the data structure using `sapply` and ordered the data with increasing matching error.

```
> raw_data_returned_example
[[1]]
[[1]][[1]]
[1] "host_response_time instant_bookable"

[[1]][[2]]
[1] 126.9275


[[2]]
[[2]][[1]]
[1] "host_response_time room_type_entire"

[[2]][[2]]
[1] 116.7821
...
> data_format_wanted_example
                                    key      val
2 host_response_time room_type_entire 116.7821
1 host_response_time instant_bookable 126.9275
...
>
```

The resulting data frame contains 3 columns. column `key` is the set of columns that are used in the linear model (each element separated by a space). Column `val` is the mean absolute prediction error of the linear model. To be convenient, column `kl` creates the length of `key` set.

B.6.1 Run 1: removal of column 3

This program is made useful when deciding which column to keep and which column to drop. For example, appendix II.7 records my try of the analysis during the development of the project. There were 59 columns that were ready to be used (e.g. columns of strings are converted to numerical values) (shown by printing `names(trndta)`). And I used Python to generate a power set of these columns, starting from the longest to the shortest. For each element in the power set, the program calculates the linear model and mean absolute prediction error. That is, the program first tries to use all 59 columns, then try all combinations of using 58 columns, then 57, ... until the program is stopped manually (because it takes too long to compute all possible values).

I ran the program for about 20 hours, and generated 1309463 lines of data. The program completes almost all cases when choosing more than 55 columns (since when there are errors, the data is not retried immediately, a few values are missing), and was interrupt when calculating the 54-column case. The precision of prediction is between 81.5 and 94.5. By investigating a few top-ranking choices, we can see that all of them do not choose column 3, which is `host_response_rate`. From a deeper analysis, we can see that the most accurate choice that chooses column 3 is ranked 28962th (using the `grep` function), and its prediction precision is 84.0919, which is about 2.5 dollars more than the best choice. Though this does not tell anything in certain, it gives us the hint that maybe the `host_response_rate` should be removed.

Another interesting fact about the data is that choosing less columns to fit seems to yield better result. By playing with the testing data, we can see that the best choice when choosing 55 columns

gives 81.61283, 56 gives 81.74551, 57 gives 82.19387, 58 gives 82.31867, and 59 gives 85.79139. There is a jump from 58 to 59, and the reason is that when choosing 59 columns, there is only one choice; when choosing 58 columns, the worst column can be dropped. It turns out that column 3 is dropped.

Another problem about the choose of columns is that we created dummy variables for every neighborhood (starting from "Bayview" at index 26 to the end). It seems that the dummy variables creates problems similar to over-fitting. After this experiment, all these dummy variables are dropped.

B.6.2 Run 2: Relationship between variables

Appendix II.8 records another run that tries to find relationship between existing columns. This run also lasted about a day, and resulted in 1672424 lines of data. The difference is that since it is difficult to come up with a permutation of all possible combinations of relationship between columns, I decided to go random. The python program generates 1 to 4 relations in random each time, and continues forever (or when manually interrupted). A relation is represented by a vector of variable length. When the first item is 0, this vector represents the end of list of relationships; when the first item is 1, the second and third vector is combined using relationship on the forth item (1 is multiply, 2 is divide); when the first item is 2, the second vector is raised to a power indicated by the third item. For example, the following line represents four relationships: multiply column 13 and 30, multiply column 9 and 26, raise column 14 to the third power, and raise column 16 to the second power.

```
1 13 30 1
1 9 26 1
2 14 3
2 16 2
0
```

Note that though there is a number reserved for dividing, dividing is not generated by the Python program because it is very easy to trigger zero division error.

If no relationship is specified, the mean absolute prediction error is 70.39295. After adding the random relationships, the error is between 63 to 72, which is a large improvement. From the ten lowest error in the data, we can see that two of the relationships have high frequency: `1 13 30 1` and `2 14 3`.

By grepping these relationships in high-ranking entries, we can see that these relationships are showing up a lot. The first relationship shows up 989 times in the highest 1000 entries (note that `1 13 30 1` is the same as `1 30 13 1`; the second relationship shows up 54 times in the highest 100 entries.

Now we can check the effect of these two relationships with other relationships removed. When running `1 13 30 1` only, the error goes down about 3 dollars to 66.37454. When running `2 14 3` only, the error goes down about 1.5 dollars to 68.93839. When running the two together, the error goes down about 5.5 dollars to 64.8013. So we can see that both these relationships do decrease the error a lot. (Note: When Eric ran this program, we were ignoring the NAs in our prediction error vector, which actually should not contain any NAs. So the prediction error here was inaccurately less than the "true" error. After we fixed the NAs, that error 64.8013 went up a little bit to 66.75325.)

The question of whether the relationship makes sense comes. By looking at the `names` table, we can see that `1 13 30 1` is the multiplication of `cleaning_fee` and `mini_night_lessthan29`. The latter is an indicator variable for whether the required minimum length of stay is larger than

a month. These two relationship are related. On the one hand, the host only needs to clean when the guest changes, so when the guests are staying for a long time, cleaning can be performed less frequently. On the other hand, when the guests are staying longer, they may make the room more "messy", so each time the cleaning need to be performed more thoroughly. Also, I think when staying for a long time, people tend to expect the room to be more clean.

For the `2 14 3` relationship, which means raising `extra_people` to the third power. However, this relationship can not be easily explained. The reason may be that the number of extra people should not be linearly related to price, or it may be the side-effect of P-hacking.

B.6.3 Run 3: Another Two Relationship

I tried again about one day later (result in appendix II.9). this run is a little bit shorter than the first two, but I still get more than one million data. The common entries are `2 10 3` and `1 8 25 1`.

`2 10 3` appears in more than 90% of the highest 1000 trials. This means raising the number of `bedrooms` to the third power. Again, this phenomenon is difficult to be explained. Maybe adding capacity will not linearly increase price, or maybe it is the side-effect of P-hacking. This column let the error go down from 77.57968 to 75.8323 (about 1.75 dollars)

`1 8 25 1` appears in more than 25% of the highest 1000 trials. This means multiplying `distance_tocenter` and `accommodates`. Fortunately, this estimator makes sense, because when the house is closer to the center of San Francisco, where houses tend to be smaller due to limited land resource, having more accommodates can increase the price to a larger extend.

B.6.4 P-Hacking

The paper "Bias, Variance, Overfitting and P-hacking" says [4], " If you try a large number of models, the 'winning' one may actually not be better than all the others." Indeed, the program I wrote is doing P-hacking. It tries all kinds of model (either by enumerating and guessing) and ranks the model by their error. Then I select the factor that minimizes the model's error. However, this minimization of error may not be effective when another training set / test set is given.

A solution to this problem is to try splitting the training and test in different ways. For example, we are currently using `set.seed(9999)` to seed the random number generator, but we can try other numbers. But this solution is left to be considered later because it requires much more computations to be performed.

B.7 Contribution

B.7.1 Eric Li (914752020, ercli)

I built the project's structure (e.g. create most files, link code using R's `source` function) and wrote the filter function on columns. I also used some geometric computation algorithms to update the distance model (B.5.1). I wrote the program that tests for good models automatically (B.6), and used this result to advise my group members of the ways to improve our result. I also debugged and made optimizations on our code.

In the report, I wrote most of "Deal with messy data" (B.3). I also wrote "Update the distance model" (B.5.1) and "Sub-project: Testing matching fields in parallel" (B.6) entirely.

[4]This paper can be downloaded from http://heather.cs.ucdavis.edu/ matloff/132/Overfitting.pdf. It should be a section of a larger book written by Norm Matloff, but I cannot find the source in order to cite it

B.7.2 Benjamin Bing (914092968, jbing)

In this program, I wrote a graph.R to visualize the longitude and latitude and the price through 3D scatter and heat map.

I wrote part of the `prepare_distance`, which was later modified by Eric into two variables. Also I wrote part of `filter_columns`, and the `drop_data`. I also get involved in the discussion of selecting variables and how to prepare them for linear model as well as doing some tests to check whether a variable should be part of the linear model.

Although I also write some functions to prepare the `host_since`, `transit`, and `property_type`, those variables turned not to have good effect on our prediction, so they were abandoned.

In the report, I am responsible for the part of the Creating Hot-spot and Distance Mode in B.5.

B.7.3 Zhiyuan Guo (914039235, zhyguo)

- Dealing with messiness (convert prices to real numbers, etc.)
- Zipcode (mapped to local housing price)
- Wrote part of `filter_columns`
- calcData
- removeOutlierPriceFromtrndta
- Test if a variable behaves well in our model and decide (with my teammates) whether to include that variable
- Discuss with my teammates on how to tackle some variables (Should we make it an indicator variable? etc.)

In the report, I am responsible for the entire Data Analysis section in B.4.

I Appendix A

I.1 Code: `ercli.R`

This program records my calculation of the correlation, which differs from the one in the poster.

```
> x1 = c(500, 1000, 5000)
> y1 = c(0.00188, 0.00102, 0.00023)
> print(cor(x1, y1))
[1] -0.9019413
> x2 = c(60, 90, 120)
> y2 = c(0.00122, 0.00092, 0.00073)
> print(cor(x2, y2))
[1] -0.9917051
```

II Appendix B

II.1 List of fields

```
 [1] "id"
 [2] "listing_url"
 [3] "scrape_id"
 [4] "last_scraped"
 [5] "name"
 [6] "summary"
 [7] "space"
 [8] "description"
 [9] "experiences_offered"
[10] "neighborhood_overview"
[11] "notes"
[12] "transit"
[13] "access"
[14] "interaction"
[15] "house_rules"
[16] "thumbnail_url"
[17] "medium_url"
[18] "picture_url"
[19] "xl_picture_url"
[20] "host_id"
[21] "host_url"
[22] "host_name"
[23] "host_since"
[24] "host_location"
[25] "host_about"
[26] "host_response_time"
[27] "host_response_rate"
[28] "host_acceptance_rate"
[29] "host_is_superhost"
[30] "host_thumbnail_url"
[31] "host_picture_url"
[32] "host_neighbourhood"
[33] "host_listings_count"
[34] "host_total_listings_count"
[35] "host_verifications"
[36] "host_has_profile_pic"
[37] "host_identity_verified"
[38] "street"
[39] "neighbourhood"
[40] "neighbourhood_cleansed"
[41] "neighbourhood_group_cleansed"
[42] "city"
[43] "state"
[44] "zipcode"
[45] "market"
[46] "smart_location"
[47] "country_code"
[48] "country"
[49] "latitude"
[50] "longitude"
[51] "is_location_exact"
[52] "property_type"
[53] "room_type"
[54] "accommodates"
```

```
 [55] "bathrooms"
 [56] "bedrooms"
 [57] "beds"
 [58] "bed_type"
 [59] "amenities"
 [60] "square_feet"
 [61] "price"
 [62] "weekly_price"
 [63] "monthly_price"
 [64] "security_deposit"
 [65] "cleaning_fee"
 [66] "guests_included"
 [67] "extra_people"
 [68] "minimum_nights"
 [69] "maximum_nights"
 [70] "minimum_minimum_nights"
 [71] "maximum_minimum_nights"
 [72] "minimum_maximum_nights"
 [73] "maximum_maximum_nights"
 [74] "minimum_nights_avg_ntm"
 [75] "maximum_nights_avg_ntm"
 [76] "calendar_updated"
 [77] "has_availability"
 [78] "availability_30"
 [79] "availability_60"
 [80] "availability_90"
 [81] "availability_365"
 [82] "calendar_last_scraped"
 [83] "number_of_reviews"
 [84] "number_of_reviews_ltm"
 [85] "first_review"
 [86] "last_review"
 [87] "review_scores_rating"
 [88] "review_scores_accuracy"
 [89] "review_scores_cleanliness"
 [90] "review_scores_checkin"
 [91] "review_scores_communication"
 [92] "review_scores_location"
 [93] "review_scores_value"
 [94] "requires_license"
 [95] "license"
 [96] "jurisdiction_names"
 [97] "instant_bookable"
 [98] "is_business_travel_ready"
 [99] "cancellation_policy"
[100] "require_guest_profile_picture"
[101] "require_guest_phone_verification"
[102] "calculated_host_listings_count"
[103] "calculated_host_listings_count_entire_homes"
[104] "calculated_host_listings_count_private_rooms"
[105] "calculated_host_listings_count_shared_rooms"
[106] "reviews_per_month"
```

II.2 Linear model result

```
> calcData(b$data, c$data)

Call:
lm(formula = trndta[, "price"] ~ ., data = trndta[, testCol])

Residuals:
    Min      1Q  Median      3Q     Max
-347.90  -39.22   -7.79   26.31  534.90

Coefficients:
                                    Estimate Std. Error t value Pr(>|t|)
(Intercept)                       -6.534e+00  3.958e+01  -0.165 0.868882
host_identity_verifiedTRUE         4.582e+00  2.022e+00   2.266 0.023510 *
room_type_entireTRUE               9.073e+01  6.965e+00  13.027  < 2e-16 ***
room_type_privateTRUE              4.896e+01  6.840e+00   7.158 9.18e-13 ***
accommodates_sqrt                  6.769e+01  5.434e+00  12.456  < 2e-16 ***
bedrooms                           1.485e+01  3.820e+00   3.887 0.000103 ***
beds                              -7.019e+00  2.883e+00  -2.435 0.014923 *
bedstimebedrooms                   3.171e+00  1.174e+00   2.701 0.006940 **
cleaning_fee                       5.753e-02  2.178e-02   2.641 0.008283 **
extra_people                      -5.495e-01  1.074e-01  -5.118 3.19e-07 ***
number_of_reviews_ltm             -4.585e-01  5.278e-02  -8.687  < 2e-16 ***
review_scores_rating_square        3.535e-02  5.790e-03   6.105 1.09e-09 ***
instant_bookableTRUE              -8.335e-01  2.063e+00  -0.404 0.686176
mini_night_lessthan29TRUE          2.672e+01  4.323e+00   6.180 6.81e-10 ***
zipPrice                           2.613e-05  2.794e-06   9.353  < 2e-16 ***
dist_to_we_center                 -8.556e+02  9.599e+01  -8.914  < 2e-16 ***
dist_to_sn_center                  5.045e+02  1.324e+02   3.810 0.000140 ***
require_guest_profile_pictureTRUE -4.281e+00  7.778e+00  -0.550 0.582046
require_guest_phone_verificationTRUE  4.124e+00  6.714e+00   0.614 0.539132
calculated_host_listings_count    -1.211e-01  2.773e-02  -4.368 1.28e-05 ***
review_scores_rating              -4.229e+00  9.369e-01  -4.514 6.47e-06 ***
availability_30                    1.234e-01  3.855e-01   0.320 0.748944
availability_30_square             1.943e-02  1.368e-02   1.421 0.155493
availability_365                  -1.797e-02  3.164e-02  -0.568 0.570105
availability_365_square            1.152e-04  8.568e-05   1.344 0.178998
clnfee_times_minnit29              5.063e-01  3.302e-02  15.333  < 2e-16 ***
extra_people_2                     1.226e-02  1.566e-03   7.829 5.76e-15 ***
extra_people_3                    -3.540e-05  4.446e-06  -7.963 1.99e-15 ***
bedrooms_2                         9.993e+00  1.665e+00   6.003 2.05e-09 ***
bedrooms_3                        -1.082e+00  9.173e-02 -11.799  < 2e-16 ***
dist2wecen_times_accom            -1.090e+02  2.755e+01  -3.956 7.72e-05 ***
dist2sncen_times_accom             1.168e+01  3.706e+01   0.315 0.752571
---
Signif. codes:  0 ?***? 0.001 ?**? 0.01 ?*? 0.05 ?.? 0.1 ? ? 1

Residual standard error: 73.67 on 6005 degrees of freedom
Multiple R-squared:  0.6236,    Adjusted R-squared:  0.6217
F-statistic:   321 on 31 and 6005 DF,  p-value: < 2.2e-16

[1] 68.40237
```

II.3 Code: `tmp.R`

```
plot_relationship = function(cola, colb, lima, limb) {
    tt = split_data()
    jkl = data.frame(a=as.numeric(gsub('[\\$,]', '', tt$trndta[,cola])),
                     b=as.numeric(gsub('[\\$,]', '', tt$trndta[,colb])))
    library(ggplot2)

    x = is.na(jkl$a) | (jkl$b > limb | jkl$a > lima)
    x[is.na(x)] = TRUE

    jkl[x,]$a = NA
    jkl[x,]$b = NA

    ggplot(jkl) + geom_point(aes(x=a, y=b))
}

plot_graph_for_report <- function(save_img=F) {
    plt = plot_relationship('price', 'security_deposit', 2500, 6000)
    if (save_img)
        ggsave("price-sd.png")
    plt = plot_relationship('price', 'cleaning_fee', 2500, 600)
    if (save_img)
        ggsave("price-cf.png")
}
```

II.4 Code: `Distrib.R`

This program creates new models and tests them following specification from stdin (which can be generated from `distrib.py`).

The current verrsion of this program only works for run 3. To run for run 2, change `a` vector in function `f` to the one above line `this is the a vector at run 2`. To run for run 1, use function `f1` instead of `f`.

Running this programs also requires using the correct version of `ProbB.R`. This becomes difficult because the git version history cannot be submitted along with the project. An alternative solution is to look at the result (e.g. appendix II.7) and modify `ProbB.R` so that the columns match.

```
f1 = function () {
  f = file('stdin')
  open(f)
  s = strsplit(readLines(f,n=1), ' ')[[1]]
  a = as.numeric(s)

  source("ProbB.R")
  load('trntstdta.Robj')
  calcData(trndta, tstdta, T, a)
}

f = function() {
  source("ProbB.R")
  load('trntstdta.Robj')

  f = file('stdin')
  open(f)
  # a = c(4, 5, 8, 9, 13, 14, 16, 18, 21, 22, 24, 26, 27, 29, 30, 31, 33)
  #  this is the a vector at run 2
  a = c(4, 8, 10, 11, 13, 14, 16, 18, 21, 22, 25, 26, 27, 29, 30, 32, 33, 34,
      35, 36)
  #  this is the a vector at run 3
  while (T) {
    s = as.numeric(strsplit(readLines(f,n=1), ' ')[[1]])
    if (s[1] == 0) break
    if (s[1] == 1) {          # relation of two lists
      if (s[4] == 1) {
        new_index = ncol(trndta) + 1
        trndta[[paste('V', new_index)]] = trndta[[s[2]]]*trndta[[s[3]]]
        tstdta[[paste('V', new_index)]] = tstdta[[s[2]]]*tstdta[[s[3]]]
        a = c(a, new_index)
      } else if (s[4] == 2) {
        new_index = ncol(trndta) + 1
        trndta[[paste('V', new_index)]] = trndta[[s[2]]]/trndta[[s[3]]]
        tstdta[[paste('V', new_index)]] = tstdta[[s[2]]]/tstdta[[s[3]]]
        a = c(a, new_index)
      }
    } else if (s[1] == 2) {   # relationship to itself
      for (i in 2:s[3]) {
        new_index = ncol(trndta) + 1
        trndta[[paste('V', new_index)]] = trndta[[s[2]]]**i
        tstdta[[paste('V', new_index)]] = tstdta[[s[2]]]**i
        a = c(a, new_index)
      }
    }
```

```
  }
  calcData(trndta, tstdta, F, a)
}

p = function(ans) {
  ans                          # print intermediate values
  print('START')
  print(ans)
  print('OK')
}

p(f())
```

II.5 Code: `distrib.py`

This program runs `Distrib.R` on different machines (currently the 60 machines on CSIF) in parallel.

This program contains code adapted from itertools documentation, which calculated power-set of a given set. The modification allows creating the power set in specific order.

The current verrsion of this program only works for run 3. To run for run 2, change `col_list` in function `create_data` to the one above line `col_list at run 2`. To run for run 1, use function `create_data1` instead of `create_data`.

There are also path names that need to be changed. Currently, all the programs need to be stored in `/home/lxy/tmp/tp/`, and the result will be generated in `/home/lxy/tmp/tp/tmp/`.

Note that this program is written in Python 3, which is installed on CSIF.

```
import re, os, sys, time, random, pickle, traceback, threading, socket
from subprocess import Popen
from threading import Thread, Lock
from itertools import chain, combinations
from collections import deque

def get_host(host_id) :
  return 'pc%d.cs.ucdavis.edu' % host_id

up_host_list = list(range(1, 61))
# deque(map(up_host_list.remove, [45, 46, 49, 50]), 0)
counter = dict(map(lambda x: (x, [get_host(x), 0, Lock()]), up_host_list))

fail = {}
fail_lck = Lock()

ans = {}
ans_dirty = False
ans_lck = Lock()

TMP = '/home/lxy/tmp/tp/tmp'
TIMEOUT = 100
CONCURRENCY_LIMIT = 2
PICKLE_NAME = TMP + '/tp.pickle'

def select_host() :
  while True :
    i = random.choice(list(counter))
    info = counter[i]
    info[2].acquire()
    if info[1] < CONCURRENCY_LIMIT :
      info[1] += 1
      ret = True
    else :
      ret = False
    info[2].release()
    if ret :
      return info
    time.sleep(0.1)

def get_result(txt) :
  splitted = txt.split('\n')
  ind = splitted.index('[1] "START"')
  assert splitted[ind + 2] == '[1] "OK"'
```

```
    return float(re.fullmatch('\[1\] (.+)', splitted[ind + 1]).groups()[0])

  def ssh(host, stdin, timeout) :
    p = Popen(['ssh', host, 'cd /home/lxy/tmp/tp/; R --file=Distrib.R'],
          stdin=-1, stdout=-1)
    o, e = p.communicate(stdin.encode(), timeout)
    #assert not e
    return o.decode()

  class CSIFThread(Thread) :
    def __init__(self, host, stdin, fail_cnt) :
      super(CSIFThread, self).__init__()
      self.stdin = stdin
      self.fail_cnt = fail_cnt
      self.host = host
    def run(self) :
      global ans, ans_lck, ans_dirty
      try :
        stdout = ssh(self.host[0], self.stdin, TIMEOUT)
        self.result = get_result(stdout)
      except Exception :
        if not 'save' :
          try :
            open(TMP + '/tb-dis/%d.info' % threading.get_ident(),
                'w').write(stdout)
          except Exception :
            print('Save failed >')
            traceback.print_exc()
            print('Save failed <')
        traceback.print_exc()
        print('Exception caught @', self.host[0])
        if self.fail_cnt < 3 :
          fail_lck.acquire()
          fail[self.stdin] = self.fail_cnt + 1
          fail_lck.release()
        else :
          print('Too many failures')
        self.host[2].acquire()
        self.host[1] -= 1
        self.host[2].release()
        return
      ans_lck.acquire()
      ans[self.stdin] = self.result
      ans_dirty = True
      ans_lck.release()
      self.host[2].acquire()
      self.host[1] -= 1
      self.host[2].release()

  class Forker(Thread) :
    def __init__(self, data, sleep) :
      super(Forker, self).__init__()
      self.data = data
      self.sleep = sleep
    def run(self) :
      for i in self.data :
        CSIFThread(select_host(), i + '\n', 0).start()
        print('s', i)
        # time.sleep(0.1)
```

```
    while True :
      fail_lck.acquire()
      if fail :
        k = next(iter(fail))
        CSIFThread(select_host(), k, fail.pop(k)).start()
      fail_lck.release()
      # time.sleep(0.1)

def powerset(iterable, st=0):
  # https://docs.python.org/3/library/itertools.html
  "powerset([1,2,3]) --> () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3)"
  s = list(iterable)
  return chain.from_iterable(combinations(s, r) for r in range(len(s), st - 1, -1))

def create_data1() :
  # git = eec6d38d725c6d8f87d85b6256529e807c0eb99c
  fields = list(range(1, 61))
  fields.remove(10)
  for i in powerset(fields, 54) :
    if len(i) > 1 :
      y = ' '.join(map(str, i))
      ans_lck.acquire()
      if y + '\n' not in ans :
        yield y
      ans_lck.release()
#  for i in range(10000) :
#    yield ' '.join(map(str, random.sample(range(100), 10)))

def create_data(nreps = 1048576) :
  # run 2: git = 62bd4e59e52aca7b3e9e5aa5b47c2db38598aa22 + 1 submission
  # run 3: git = 15073f267488cf51661fe29dc9d05dc18ec4206e + 1 submission
  # which(names(trndta) %in% getTC())
  #col_list = [4, 5, 8, 9, 13, 14, 16, 18, 21, 22, 24, 26, 27, 29, 30, 31, 33]
  # col_list at run 2
  col_list = [4, 8, 10, 11, 13, 14, 16, 18, 21, 22, 25, 26, 27, 29, 30, 32,
        33]
  # col_list at run 3
  for i in range(nreps) :
    ans = ''
    for j in range(random.randint(1, 4)) :
      case = random.random()
      if case > 0.5 :
        ans += '1 %d %d' % tuple(random.sample(col_list, 2)) \
          + ' %d\n' % random.randint(1, 1)
      else :
        ans += '2 %d %d\n' % (random.choice(col_list),
                  random.randint(2, 3))
    ans += '0\n'
    yield ans

def main() :
  # print(get_result(ssh(host_id(42), '1 2 3\n', 10)))
  # print(get_result(sys.stdin.read()))
  global ans, ans_lck, ans_dirty
  if os.path.exists(PICKLE_NAME) :
    ans = pickle.load(open(PICKLE_NAME, 'rb'))
    print('r', len(ans))
  Forker(data=create_data(), sleep=1).start()
  while True :
```

```
    time.sleep(600)
    # Not concurrency-safe
    print('a', sum(map(lambda x: x[1], counter.values())))
    # ans
    ans_lck.acquire()
    if ans_dirty :
      print('D', len(ans))
      try :
        pickle.dump(ans, open(PICKLE_NAME, 'wb'))
      except KeyboardInterrupt :
        print('Control-C ignored')
      ans_dirty = False
    else :
      print('d', len(ans))
    ans_lck.release()

def read_data(file_name=PICKLE_NAME) :
  return list(map(tuple, pickle.load(open(file_name, 'rb')).items()))
  # for k, v in list(reversed(sorted(a.items(), key=lambda x: x[1]))): print(v, len
    (k.split()), k[:50], sep='\t')

if __name__ == '__main__' :
  if sys.argv[1:] :
    main()

# while sleep 1h; do cp tp.pickle `mktemp -u ./tp.pickle.backup.XXXXXXXXX`; done
```

II.6 Code: `DistribAnalyze.R`

This program analyzes the result from `distrib.py`.

```
load_data <- function() {
    library(reticulate)
    use_python("/usr/bin/python3")
    source_python("distrib.py")
    data <- read_data()
    k = sapply(data, function(x){x[[1]]})
    v = sapply(data, function(x){x[[2]]})
    data = data.frame(key = k, val = v, stringsAsFactors=FALSE)
    data[order(data$v),]
}

distrib_analyze <- function() {
    data = load_data()
    get_num_elem = function(x) {length(strsplit(x, ' ')[[1]])}
    data$kl = unlist(lapply(data$k, get_num_elem))
    data
}

# source("DistribAnalyze.R")
# data = distrib_analyze()
```

II.7 Result: remove host_response_rate

```
> names(trndta)
 [1] "transit"                      "host_response_time"
 [3] "host_response_rate"           "host_is_superhost"
 [5] "host_identity_verified"       "accommodates"
 [7] "bathrooms"                    "bedrooms"
 [9] "beds"                         "price"
[11] "cleaning_fee"                 "extra_people"
[13] "minimum_nights"               "number_of_reviews_ltm"
[15] "instant_bookable"             "cancellation_policy"
[17] "reviews_per_month"            "host_since_log"
[19] "host_since_sqrt"              "room_type_entire"
[21] "room_type_private"            "amenities_breakfast"
[23] "review_scores_rating_square" "mini_night_lessthan29"
[25] "bedstimebedrooms"             "Bayview"
[27] "BernalHeights"                "CastroUpperMarket"
[29] "Chinatown"                    "CrockerAmazon"
[31] "DiamondHeights"               "DowntownCivic"
[33] "Excelsior"                    "FinancialDistrict"
[35] "GlenPark"                     "GoldenGatePark"
[37] "HaightAshbury"                "InnerRichmond"
[39] "InnerSunset"                  "Lakeshore"
[41] "Marina"                       "Mission"
[43] "NobHill"                      "NoeValley"
[45] "NorthBeach"                   "OceanView"
[47] "OuterMission"                 "OuterRichmond"
[49] "OuterSunset"                  "PacificHeights"
[51] "Parkside"                     "PotreroHill"
[53] "PresidioHeights"              "RussianHill"
[55] "Seacliff"                     "SouthofMarket"
[57] "TwinPeaks"                    "VisitacionValley"
[59] "WestofTwinPeaks"              "WesternAddition"
> a = distrib_analyze()
> nrow(a)
[1] 1309463
> names(a)
[1] "key" "val" "kl"
> table(a$kl)

    54     55     56     57     58     59
820071 455116  32505   1711     59      1
> a[1:10,]


387883 1 2 4 5 6 7 8 9 11 12 13 14 16 17 18 19 20 21 22 23 24 25 26 28 29 30 31 32
    33 34 35 36 37 38 39 40 41 42 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
    60
378935 1 2 4 5 6 7 8 9 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 29 30 31
    32 33 34 35 36 37 38 39 40 41 42 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58
    59
386999 1 2 4 5 6 7 8 9 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 28 29 30 31 32
    33 34 35 36 37 38 39 40 41 42 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
    60
387623 1 2 4 5 6 7 8 9 11 12 13 14 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
    32 33 34 35 36 38 39 40 41 42 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
    60
375997 1 2 4 5 6 7 8 9 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
    31 32 33 34 35 36 38 39 40 41 42 44 45 46 47 48 49 50 51 53 54 55 56 57 58 59
```

```
    60
386664 1 2 4 5 6 7 8 9 11 12 13 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
    32 33 34 35 36 37 38 39 40 41 42 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
    60
376089 1 2 4 5 6 7 8 9 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
    31 32 33 34 35 36 39 40 41 42 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
    60
379404 1 2 4 5 6 7 8 9 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 28 29 30 31
    32 33 34 35 36 37 38 39 40 41 42 44 45 46 47 48 49 50 51 52 53 54 56 57 58 59
    60
911581 1 2 4 5 6 7 8 9 11 12 13 14 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
    32 33 34 35 36 37 38 39 40 41 42 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
    60
387849 1 2 4 5 6 7 8 9 11 12 13 14 16 17 18 19 20 21 22 23 24 25 26 27 29 30 31 32
    33 34 35 36 37 38 39 40 41 42 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
    60
           val kl
387883 81.61283 55
378935 81.62619 55
386999 81.62675 55
387623 81.63013 55
375997 81.63132 55
386664 81.63474 55
376089 81.64025 55
379404 81.64259 55
911581 81.64379 55
387849 81.64405 55
> a[(nrow(a) - 9):nrow(a),2:3]
           val kl
392185 92.40963 55
242183 92.65807 55
392312 92.72904 55
323196 92.75713 55
911921 92.92552 55
322950 93.21298 55
323064 93.58251 55
397105 93.66776 55
392180 94.08171 55
325585 94.40886 55
> table(a$kl)

    54     55     56     57     58     59
820071 455116  32505   1711     59      1
> grep('\\b3\\b', a[,1])[1]
[1] 28962
> a[28962,2:3]
          val kl
490586 84.0919 54
>
> a[grep(55, a$kl)[1],2:3]
           val kl
387883 81.61283 55
> a[grep(56, a$kl)[1],2:3]
          val kl
27585 81.74551 56
> a[grep(57, a$kl)[1],2:3]
         val kl
1529 82.19387 57
> a[grep(58, a$kl)[1],2:3]
```

```
        val kl
31 82.31867 58
> a[grep(59, a$kl)[1],2:3]
        val kl
15 85.79139 59
>
> a[grep(58, a$kl)[1],]


31 1 2 4 5 6 7 8 9 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
    32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
    58 59 60
        val kl
31 82.31867 58
>
```

II.8 Result: Two Relationships

```
> names(trndta)
 [1] "transit"                          "host_response_time"
 [3] "host_response_rate"               "host_is_superhost"
 [5] "host_identity_verified"           "latitude"
 [7] "longitude"                        "accommodates"
 [9] "bathrooms"                        "bedrooms"
[11] "beds"                             "price"
[13] "cleaning_fee"                     "extra_people"
[15] "minimum_nights"                   "number_of_reviews_ltm"
[17] "review_scores_rating"             "instant_bookable"
[19] "cancellation_policy"              "reviews_per_month"
[21] "access"                           "interaction"
[23] "calculated_host_listings_count" "host_since_log"
[25] "host_since_sqrt"                  "room_type_entire"
[27] "room_type_private"                "amenities_breakfast"
[29] "review_scores_rating_square"    "mini_night_lessthan29"
[31] "bedstimebedrooms"                 "googleValue"
[33] "zipPrice"
> a = distrib_analyze()
> nrow(a)
[1] 1672424
> names(a)
[1] "key" "val" "kl"
> a[1:10,]
                                                    key      val kl
713292      1 13 30 1\n1 9 26 1\n2 14 3\n2 16 2\n0\n\n 63.05182 11
279079  1 29 30 1\n1 30 13 1\n2 14 3\n1 33 31 1\n0\n\n 63.06410 12
625153   1 24 26 1\n2 14 3\n1 9 26 1\n1 13 30 1\n0\n\n 63.17060 12
604891  1 30 13 1\n1 29 31 1\n2 14 3\n1 26 14 1\n0\n\n 63.21591 12
1482434            2 14 3\n1 31 33 1\n1 30 13 1\n0\n\n 63.23065  9
568599      1 26 9 1\n2 14 3\n2 21 3\n1 30 13 1\n0\n\n 63.26007 11
1182048 2 14 3\n1 33 26 1\n1 30 13 1\n1 31 33 1\n0\n\n 63.27259 12
526693  1 26 22 1\n2 14 3\n1 30 13 1\n1 29 31 1\n0\n\n 63.32605 12
1478982    2 31 2\n1 13 30 1\n1 29 31 1\n2 14 3\n0\n\n 63.33191 11
703410  1 30 13 1\n2 14 3\n1 33 14 1\n1 29 31 1\n0\n\n 63.36576 12
> a[(nrow(a) - 9):nrow(a),2:3]
             val kl
1469658 71.61099 10
1487943 71.61099 10
1174255 71.63588 12
1490809 71.63648 11
690104  71.65225 11
1041236 71.65225 11
87214   71.70299 11
725510  71.70978 11
921814  71.70978  9
1108677 71.70978  9
> length(grep("(\n|^)2 14 3(\n|$)", a$key[1:100]))
[1] 54
> length(grep("(\n|^)1 (13 30|30 13) 1(\n|$)", a$key[1:100]))
[1] 99
> length(grep("(\n|^)1 (13 30|30 13) 1(\n|$)", a$key[1:1000]))
[1] 989
>
```

II.9 Result: Another Two Relationships

```
> names(trndta)
 [1] "transit"                      "host_response_time"
 [3] "host_response_rate"           "host_is_superhost"
 [5] "host_identity_verified"       "latitude"
 [7] "longitude"                    "accommodates"
 [9] "bathrooms"                    "bedrooms"
[11] "beds"                         "price"
[13] "cleaning_fee"                 "extra_people"
[15] "minimum_nights"               "number_of_reviews_ltm"
[17] "review_scores_rating"         "instant_bookable"
[19] "cancellation_policy"          "reviews_per_month"
[21] "access"                       "interaction"
[23] "host_since_log"               "host_since_sqrt"
[25] "distance_tocenter"            "room_type_entire"
[27] "room_type_private"            "amenities_breakfast"
[29] "review_scores_rating_square" "mini_night_lessthan29"
[31] "googleValue"                  "zipPrice"
[33] "bedstimebedrooms"             "clnfee_times_minnit29"
[35] "extra_people_2"               "extra_people_3"
> a = distrib_analyze()
> nrow(a)
[1] 1140435
> names(a)
[1] "key" "val" "kl"
> a[1:10,]
                                                    key      val kl
774575   1 32 10 1\n1 8 25 1\n2 10 3\n1 13 4 1\n0\n\n 74.51643 12
115068            1 25 33 1\n1 32 10 1\n2 10 3\n0\n\n 74.56350  9
120372     2 22 3\n2 10 3\n1 25 8 1\n1 33 32 1\n0\n\n 74.59407 11
282033             2 10 3\n1 25 8 1\n1 33 32 1\n0\n\n 74.59407  9
475794             1 8 25 1\n1 33 32 1\n2 10 3\n0\n\n 74.59407  9
876399             1 33 32 1\n2 10 3\n1 25 8 1\n0\n\n 74.59407  9
674138  1 32 33 1\n1 10 8 1\n1 29 33 1\n2 10 3\n0\n\n 74.61166 12
498510    2 10 3\n1 13 16 1\n2 22 2\n1 32 33 1\n0\n\n 74.63608 11
856853    2 10 3\n2 26 3\n1 33 32 1\n1 13 16 1\n0\n\n 74.63608 11
1136044           2 10 3\n1 33 32 1\n1 13 16 1\n0\n\n 74.63608  9
> a[(nrow(a) - 9):nrow(a),2:3]
            val kl
725082  80.21362 10
315523  80.22554 10
904935  80.24023 11
495917  80.24341 10
711765  80.24341 10
517740  80.24767 10
702905  80.24767 10
728625  80.27577 10
98133   80.30920 11
1100009 80.31587 11
> length(grep("(\n|^)2 10 3(\n|$)", a$key[1:100]))
[1] 94
> length(grep("(\n|^)2 10 3(\n|$)", a$key[1:1000]))
[1] 917
> length(grep("(\n|^)1 (25 8|8 25) 1(\n|$)", a$key[1:100]))
[1] 25
> length(grep("(\n|^)1 (25 8|8 25) 1(\n|$)", a$key[1:1000]))
[1] 283
```

II.10 Code: `graph.R`

```
source("ProbB.R")

pkgCheck <-function(pkg){
  if(!( pkg %in% rownames(installed.packages() ))){
    install.packages(pkg)
  }
}

plot3dscatter <- function(x = TRUE,save = FALSE){
  library("plotly")
  Sys.setenv("plotly_username"="bjm1997")
  Sys.setenv("plotly_api_key"="mnmn1112")
  lst <- split_data()
  if(x == TRUE)
    {lst <- lst$trndta}
  else
    {lst <- as.data.frame(cbind(lst$trndta,lst$tstdta))}
  lst$price <- priceToNum(lst$price)
  lst <- lst[lst$price < 1500,]
  #p <-plot_ly(x = lst$longitude, y = t(lst$latitude), z = price)
  p <- plot_ly(lst, x = ~longitude, y = ~latitude, z = ~price,width = 500, height =
       500) %>%
    add_markers(size = 2) %>%
    layout(scene = list(xaxis = list(title = 'long'),
                        yaxis = list(title = 'lat'),
                        zaxis = list(title = 'price')
                        )
           )
  pkgCheck("processx")
  library("processx")
  #NOT YET DONE
  #need to install orca from github
  #need to test, saved img different from what i have in r studio
  if(save == TRUE) orca(p, "scatter-plot.png")
  p
}

#call p <- heatmap(TRUE, TRUE)
heatmap <-function(x = TRUE, save = FALSE){
  #cite https://blog.dominodatalab.com/geographic-visualization-with-rs-ggmaps/
  pkgCheck(("devtools"))
  if(!( "ggmap" %in% rownames(installed.packages() ))){
    devtools::install_github("dkahle/ggmap")}
  library(ggmap)
  library(RColorBrewer)
  #personal api, do not leak
  register_google(key = "AIzaSyCaZan-88_lIvrf1fb1p4lO2aOo5PC7vvw")
  MyMap <- get_map(location = c(-122.443050,37.754014), source = "google",
                   maptype = "terrain", crop = FALSE, zoom = 12)
  if(x == TRUE)
    {lst <- read.csv("listings.csv.gz")}
  else
    {lst <- split_data()$tstdta}
  lst$price <- priceToNum(lst$price)
  lst <- lst[lst$price < 1500,]
  pricemap <-data.frame(lon = lst$longitude, lat = lst$latitude, price = lst$price)
  p <- ggmap(MyMap, extent = "device") + geom_density2d(data = pricemap, aes(x =
```

```
      lon, y = lat), size = 0.3)+
    stat_density2d(data = pricemap,
                   aes(x = lon, y = lat, fill = ..level.., alpha = ..level..), size
                        = 0.03,
                   bins = 16, geom = "polygon") + scale_fill_gradient(low = "blue",
                       high = "red") +
    scale_alpha(range = c(0.2, 0.6), guide = FALSE)
  if(save)
    ggplot2::ggsave(filename="heatmap.png", plot=p)
  p
}
#from the heatmap one peak point is c(37.792123, -122.405556) lat/long
#c(37.764424, -122.427180) lat/long
#c(37.755024, -122.410524) lat/long
```

II.11 Code: `ProbB.R`

```
mymain <- function()
{
  tt = split_data()
  trndtas = filter_columns(tt$trndta)
  trndta = trndtas$data
  tstdtas = filter_columns(tt$tstdta, trndtas$means)
  tstdta = tstdtas$data
  print(calcData(trndta, tstdta))
}


# read the input file and breaks the data into training and test sets
split_data <- function(filename='listings.csv.gz') {
  lsts <- read.csv(filename)

  # code below is from prompt
  set.seed(9999)
  idxs <- sample(1:nrow(lsts),1000)
  tstdta <- lsts[idxs,]
  trndta <- lsts[-idxs,]
  # code above is from prompt
  trndta = trndta[trndta$price != 0,]
  list(trndta=trndta, tstdta=tstdta)
}

# specify which column to use in the linear model
getTC <- function() {
  c(
    "host_identity_verified",         # 37  no NA
    "room_type_entire",               # 53 no NA
    "room_type_private",              #    no NA
    "accommodates_sqrt",
    "bedrooms",                       # 56
    "beds",                           # 57
    "bedstimebedrooms",               # no NA
    "cleaning_fee",                   # 65 has 775 NAs, -> mean
    "extra_people",                   # 67 no NA
    "number_of_reviews_ltm",          # 84 no NA
    "review_scores_rating_square",    # 87 no NA
    "instant_bookable",               # 97 no NA
    "mini_night_lessthan29",          # no NA
    "zipPrice",                      # zipPrice better then googleValue
    "dist_to_we_center",
    "dist_to_sn_center",
    "require_guest_profile_picture",
    "require_guest_phone_verification",
    "calculated_host_listings_count",
    "review_scores_rating",
    "availability_30",
    "availability_30_square",
    "availability_365",
    "availability_365_square",

    # interaction terms
    "clnfee_times_minnit29",          # 65 * 68
    "extra_people_2",                 # 67**2
```

```
    "extra_people_3",                 # 67**3
    "bedrooms_2",                     # 56**2
    "bedrooms_3",                     # 56**3
    "dist2wecen_times_accom",         # (lon, lat) * 54
    "dist2sncen_times_accom"
    )
}

calcData <- function(trndta, tstdta, test_trn=FALSE, testCol = getTC()) {

  if (test_trn) {
    tstdta = trndta   # this is our error of prediction on traindata itself
  } else {
                      # this is our error of prediction on testdata itself
  }

  trndta = removeOutlierPriceFromtrndta(trndta)

  lmout <- makeLmModel(trndta, testCol)
  print(summary(lmout))
  predPrice <- predictPrice(lmout, tstdta, testCol)
  scorePredic(tstdta, predPrice)
}

# remove rows in trndta where price is greater than 700
removeOutlierPriceFromtrndta <- function(trndta) {
  pr = trndta$price
  # from kaggle
  trndta = trndta[pr <= 700, ]
  trndta
}

# Convert data to numerical values; perform calculations / interactions
filter_columns = function(data, means=NULL) {
  data <- drop_data(data)

  #refine kept data (can write a prepare function to shorten)
  # data <- prepare_transit(data) #no null,all set as numeric
  # data <- prepare_hostsince(data) #no null
  data <- prepare_longlat(data)

  data$room_type_entire <- (data$room_type == 'Entire home/apt')
  data$room_type_private <- (data$room_type == 'Private room')
  data$room_type <- NULL

  data$cleaning_fee = priceToNum(data$cleaning_fee)

  # remove NA with mean
  data$extra_people = priceToNum(data$extra_people)
  data$instant_bookable = replacetfwithTrueFalse(data$instant_bookable)
  data$price = priceToNum(data$price)

  data$mini_night_lessthan29 = (data$minimum_nights < 29)

  if (is.null(means)) {
    means = list(
      beds=median(ignore_na(data$beds)),
      bedrooms=median(ignore_na(data$bedrooms)),
      bathrooms=median(ignore_na(data$bathrooms)),
```

```
      cleaning_fee=median(ignore_na(data$cleaning_fee)),
      review_scores_rating=median(ignore_na(data$review_scores_rating))
    )
  }

  data$beds = replace_na(data$beds, means$beds)
  data$bedrooms = replace_na(data$bedrooms, means$bedrooms)
  data$bathrooms = replace_na(data$bathrooms, means$bathrooms)
  data$cleaning_fee = replace_na(data$cleaning_fee, means$cleaning_fee)
  data$review_scores_rating <- replace_na(data$review_scores_rating,
                                          means$review_scores_rating)
  data$review_scores_rating_square = data$review_scores_rating ^ 2

  data <- prepare_zipcode(data)
  data$host_identity_verified =
              replacetfwithTrueFalse(data$host_identity_verified)

  # interaction terms below
  data$bedstimebedrooms = data$beds * data$bedrooms
  data$clnfee_times_minnit29 = data$cleaning_fee * data$mini_night_lessthan29
  data$extra_people_2 = data$extra_people**2
  data$extra_people_3 = data$extra_people**3
  data$bedrooms_2 = data$bedrooms ** 2
  data$bedrooms_3 = data$bedrooms ** 3
  data$dist2wecen_times_accom = data$dist_to_we_center * data$accommodates
  data$dist2sncen_times_accom = data$dist_to_sn_center * data$accommodates

  data$require_guest_profile_picture =
           replacetfwithTrueFalse(data$require_guest_profile_picture)
  data$require_guest_phone_verification =
           replacetfwithTrueFalse(data$require_guest_phone_verification)
  data$accommodates_sqrt = sqrt(data$accommodates)

  data$availability_30_square = data$availability_30**2
  data$availability_365_square = data$availability_365**2

  list(data=data, means=means)
}

prepare_zipcode <- function(data)
{
  # see zipcode.txt for more details

  tempPrice = c(917900,1018300,903000,1032000,1201500,1072100,
           1263500,1542000,1161600,1129900,1994200,1569400,
           1402900,1651100,2135900,1652500,1452000,2523500,
           956400,1790400,1668600,1344000,1002000,1390100)
  names(tempPrice) = as.character(c(94014,94080,94102,94103,94107,
                                    94108,94109,94110,94111,94112,
                                    94114,94115,94116,94117,94118,
                                    94121,94122,94123,94124,94127,
                                    94131,94133,94134,94965))
  data$zipPrice = as.vector(tempPrice[as.character(data$zipcode)])
  # replace others with mean
  data$zipPrice[is.na(data$zipPrice)] = 1407283
  data$zipcode <- NULL
  data
}
```

```
# prepare_transit <- function(data){
#   weighted_tran <- replicate(nrow(data),0)
#   #USE GPA Grading, C = 2, B = 3, A+ = 5
#   #bus/shuttle = 2, bart/ muni <- 3 bart and muni = 5
#   x <- grep("\\bbart\\b", data$transit, ignore.case=TRUE)
#   weighted_tran[x] <- weighted_tran[x] + 3
#   x <- grep("\\bmuni\\b", data$transit, ignore.case=TRUE)
#   weighted_tran[x] <- weighted_tran[x] + 3
#   weighted_tran[weighted_tran == 3 * 2] <- 5
#   x <- grep("\\bbus\\b", data$transit, ignore.case=TRUE)
#   weighted_tran[x] <- weighted_tran[x] + 2
#   x <- grep("\\bshuttle\\b", data$transit, ignore.case=TRUE)
#   weighted_tran[x] <- weighted_tran[x] + 2
#   data$transit <- weighted_tran
#   data
# }
#
# prepare_hostsince <-function(data){
#   #return value of log(2019-2-1 - host_since)
#   #I think the log can be a better value, since the range of days is very big
#   days <- as.Date("2019-2-1") - as.Date(data$host_since)
#   days <- as.numeric(days, units = "days")
#   #checked, only one value = 0, rest values in (1~8)
#   data$host_since_log <- log(days)
#   data$host_since_sqrt <- sqrt(days)
#   data$host_since <- NULL
#   data
# }


# drop completely useless columns
drop_data <- function(lsts){
  x <- c(37,44,49,50,53,54,55,56,57,61,65,67,68,78,81,84,87,
         97,100,101,102)
  return(lsts[ ,x])
}

# make linear model using specified list of columns
makeLmModel <- function(trndta, testCol){
  lm(trndta[, "price"] ~., data = trndta[, testCol])
}

# predict price using the linear model
predictPrice <- function(lmout, tstdta, testCol) {
  raw = predict(lmout, newdata = tstdta[, testCol])
  ifelse(raw > 0, raw, 0)
  # maybe ifelse(raw > 10, raw, 10) ? since the smallest price is 10
}

# calculate mean absolute prediction error
scorePredic <- function(tstdta, predPrice) {
  truePrice <- tstdta[, "price"]
  d = as.vector(abs(as.vector(truePrice) - as.vector(predPrice)))
  # use abs instead of square as required by prompt.
  mean(d)
}

# Replace all NA entries in vector col by val
replace_na = function(col, val) {
```

```
  ifelse(is.na(col), val, col)
}

# Remove all NA entries in a vector
ignore_na = function(col) {
  col[!is.na(col)]
}

# "f" -> FALSE; "t" -> TRUE
replacetfwithTrueFalse <- function(vec)
{
  # Same from https://stackoverflow.com/questions/24849699/
  # replace 't' and 'f' with True and False
  ft = c(FALSE, TRUE)
  names(ft) = c("f", "t")
  as.vector(ft[vec])
}

# "$12,345.67" -> 12345.67
priceToNum <- function(vec)
{
  # change price format to numeric
  as.numeric(gsub('\\$|,', '', vec))
}

# dist2point([longitude=1, latitude=40], c(4, 44)) -> 5
dist2point <- function(data, p) {
  ((data$longitude - p[1]) ** 2 + (data$latitude - p[2]) ** 2) ** 0.5
}

# distance2(c(1, 40), c(4, 44)) -> 25
distance2 <- function(p1, p2) {
  sum((p1 - p2)**2)
}

# distance(c(1, 40), c(4, 44)) -> 5
distance <- function(p1, p2) {
  distance2(p1, p2) ** 0.5
}

# cross product of 2D vectors
cross <- function(p1, p2) {
  p1[1] * p2[2] - p2[1] * p1[2]
}

# dot product of 2D vectors
dot <- function(p1, p2) {
  sum(p1 * p2)
}

# whether p3 and p4 are on the same side of p1 and p2
# See "Introduction to Algorithms" Chapter 33
same_side <- function(p1, p2, p3, p4, s, d) {
  cross(p2 - p1, p3 - p1) * cross(p2 - p1, p4 - p1) > 0
}

# distance to any point on the line segment
# some algorithms are from "Introduction to Algorithms" Chapter 33
dist2line <- function(data, p1, p2) {
```

```
  don = p1[1] - p2[1] # diff in lon
  dat = p1[2] - p2[2] # diff in lat
  perp = c(-dat, don) # an offset perpendicular to p1p2
  q1 = p1 + perp      # q1p1 perpendicular to p1p2
  q2 = p2 + perp      # q2p2 perpendicular to p1p2
  ans = rep(NA, nrow(data))
  for (i in 1:nrow(data)) {
    d = c(data$longitude[i], data$latitude[i])
    if (!same_side(p1, q1, p2, d)) {
      ans[i] = distance(p1, d)
    } else if (!same_side(p2, q2, p1, d)) {
      ans[i] = distance(p2, d)
    } else {
      rate = dot(p2 - p1, d - p1) / distance2(p1, p2)
      ans[i] = distance(p1 + (p2 - p1) * rate, d)
    }
  }
  ans
}

#from the heatmap one peak point is c(37.792123, -122.405556) lat/long
#c(37.764424, -122.427180) lat/long
#c(37.755024, -122.410524) lat/long
prepare_longlat <- function(data){
  long <- data$longitude
  lat <- data$latitude
  y <- 0
  we <- dist2point(data, c(-122.405556, 37.792123))
  sn <- dist2line(data, c(-122.410524, 37.755024), c(-122.427180, 37.764424))
  data$dist_to_we_center = we
  data$dist_to_sn_center = sn
  data
}
```