Predicting Housing Price in Melbourne

Yizhou Zhang

April 30, 2018

This analysis below answers four questions of the final project of a machine learning course that I took. The dataset is the "Melbourne Housing Market" dataset from Kaggle:

https://www.kaggle.com/anthonypino/melbourne-housing-market/data

Q1: Data Preprocessing and Analysis

The goal of this analysis is to predict the Price variable, hence any missing value of this variable can be discarded from the data (you can still use them if you want). Summarize each variable in the data set and provide simple descriptive statistics using tables/plots. You do not have to use the original scale of the variables. Any transformations, such as log, square root etc are allowed. The variables come in different forms (text, date, categorical, continuous, etc), you should provide a detailed discussion on how to transform them into a format that can be analyzed by regression models.

Answer begins:

The dataset includes the following variables:

Variable Name	Description
Suburb	Suburb
Address	Address
Rooms	Number of rooms
Type	Type of residence
Price	Sale Price
Method	The method of acquiring sales price
SellerG	Name of Realtor
Date	Date Sold
Distance	Distance from CBD
Postcode	Postcode
Bedroom2	Scraped # of Bedrooms (from different source)
Bathroom	Number of Bathrooms
Car	Number of carspots
Landsize	Land Size
BuildingArea	Building Size
YearBuilt	Year the house was built
CouncilArea	Governing council for the area
Lattitude	Latitude
Longitude	Longitude
Regionname	General Region
PropertyCount	Number of properties that exist in the suburb

After removing all observations with NAs, there are 8755 observations int the dataset.

Variable preprocessing

Removal of variables I removed the variable `Address` as it will not be used in this analysis. I further remove `Suburbs`, `CouncilArea`, `Regionname`, and `SellerG` because these string variables would create excessive amounts of dummy variables in regression. However, I will keep `Postcode`.

log-transform the price Log-transformation of the dependent variable makes its distribution closer to a normal distribution.

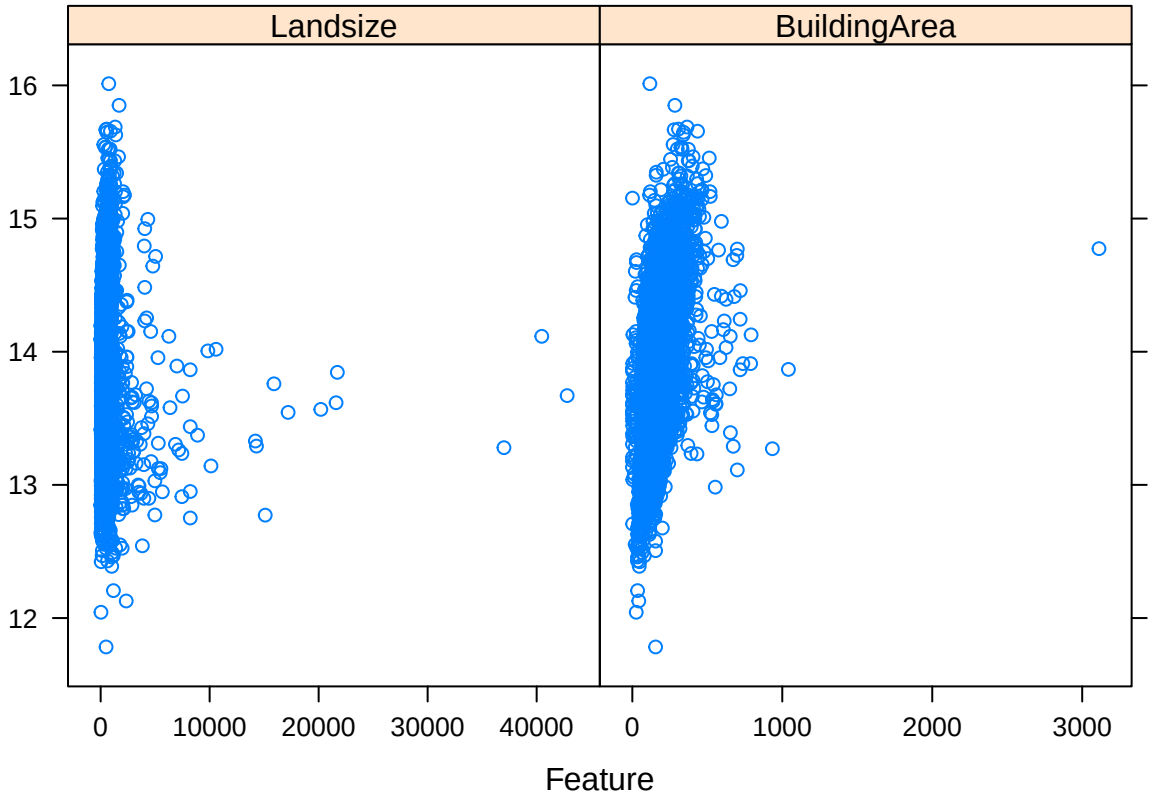
Interaction of lattitude and longitude I also added an interaction term between latitude and longitude of the house. This interaction term, together with the latitude and longitude, should account for unobserved spatial characteristics that are not covered in the dataset.

Factorization of postcode The variable `postcode` was stored as numeric but it be factorized as strings.

Removal of Bedroom2 The variable `Bedroom2` has the same meaning as `Rooms` but is scraped from a different source. The inclusion of both in a regression would create multicollinearity. Therefore, I remove all observations where the two variables are not consistent, and then remove `Bedroom2`

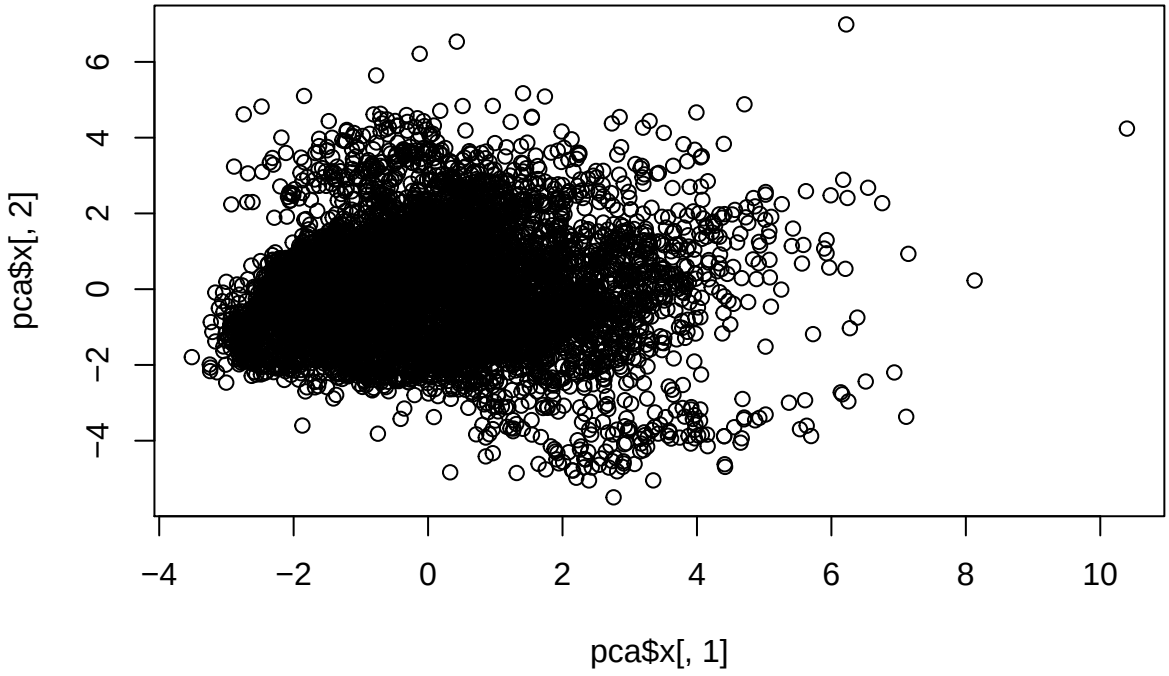
Extraction of year and month I used the substring function to extract date and month from the `Date` variable and factorize them respectively. In this way, I control for the year and month fixed effects.Then I remove the variable `Date`

Remove outliers with odd Construction Year Remove the one observation constructed in 1196, before Australia even existed.



Remove outliers with odd Building area The left panel of the figure above plots housing price against building areas. It shows that there are only a few observations with `BuildingArea` greater than or equal to 1000. Remove them as outliers.

Remove outliers with odd landsizes The right panel of the figure above plots housing price against landsizes. It shows that there are only a few observations with `landsize` greater than or equal to 10000. Given the significant impacts of outliers in regression, I remove these observations with unusually large landsizes.



Remove outliers from Principal Component Analysis The above figure plots the first and second principal components after applying PCA to all numeric variables. The observations with the 1st component greater than ten are removed as outliers.

Summary

After all the data preprocessing procedures, the table below provides the summary statistics of the numeric variables in the dataset with 7359 observations.

	Mean	Median	Std	Min	Max
Rooms	3.23	3.00	0.88	1.00	12.00
Price	13.81	13.76	0.51	11.78	16.01
Distance	11.87	11.00	6.88	0.00	47.30
Bathroom	1.69	2.00	0.72	1.00	6.00
Car	1.76	2.00	0.99	0.00	10.00
Landsize	552.53	538.00	521.88	1.00	9838.00
BuildingArea	157.17	139.00	77.73	1.00	934.00
YearBuilt	1964.51	1970.00	36.51	1830.00	2019.00
Lattitude	-37.80	-37.79	0.09	-38.17	-37.41
Longtitude	144.99	145.00	0.13	144.42	145.48
Propertycount	7376.11	6482.00	4398.62	389.00	21650.00
Longlat	-5481.01	-5479.41	15.97	-5541.26	-5420.08

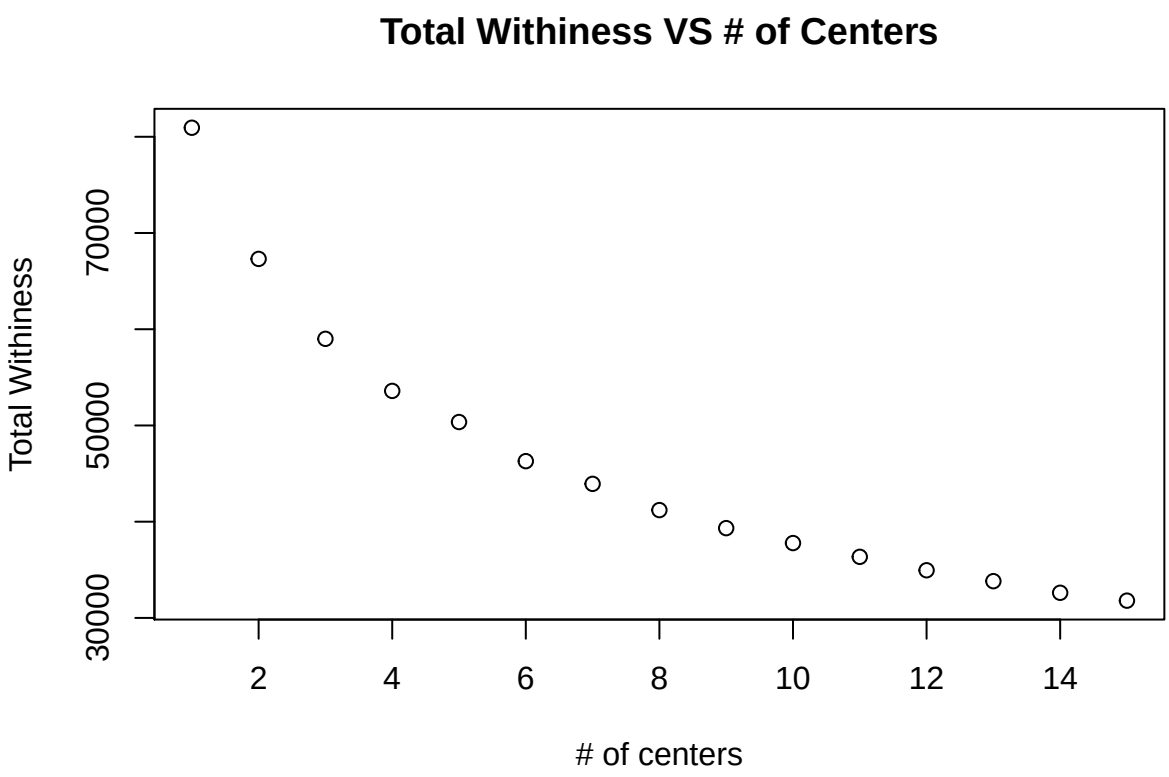
Question 2. Clustering Analysis

The goal is to cluster the houses into categories. You can use any variable except the Price variable to define the clusters. The clusters you found should have intuitive interpretations such that it might help in the predictive model for Price. You can use any clustering algorithm. Clearly describe your approach, demonstrate the findings and interpret your results.

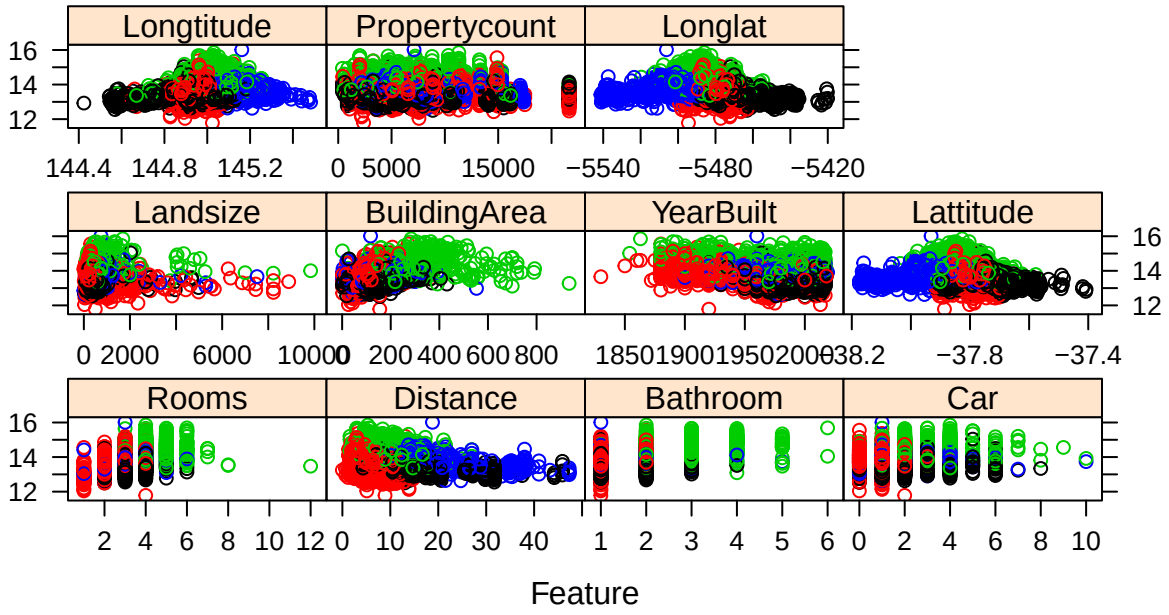
Answer begins:

K-means clustering

I am going to apply a K-mean clustering scheme on all the numeric variables of the dataset, except for `Price`. Specifically, the clustering is based on Rooms, Distance, Bathroom, Car, Landsize, BuildingArea, YearBuilt, Lattitude, Longtitude, Propertycount, Longlat, while the number of centers will be chosen based on the total withiness distance:



The figure above plots the number of clusters against the total withiness of a K-means clustering scheme. The plot does not exhibit an "elbow" where the value drops significantly. I will choose four centers for the sake of consicity. This, as shown below, is sufficient to unveil undelrying patterns relevant to price prediction.



The scatter plots above present the results of K-means clustering. It is obvious that the green group (group 3) has higher housing values than all others. The table below compares its summary statistics with the other groups:

	Group1	Group2	Group3	Group4
Rooms	3.31	2.63	4.23	3.27
Price	13.43	13.78	14.37	13.83
Distance	15.20	7.20	10.18	18.49
Bathroom	1.69	1.25	2.55	1.67
Car	2.06	1.17	2.34	1.91
Landsize	529.35	475.61	699.52	587.31
BuildingArea	149.46	112.95	258.93	150.03
YearBuilt	1987.88	1942.82	1965.45	1973.80
Lattitude	-37.72	-37.80	-37.82	-37.92
Longtitude	144.90	144.97	145.04	145.13
Propertycount	7345.56	7815.18	6873.64	7054.49
Longlat	-5465.20	-5479.58	-5484.75	-5503.48

The above table shows that group 3's (logged) sale prices are significantly higher than the other groups. Additionally, homes in this group have more rooms and bathrooms, more car spots, larger land sizes and building areas, and lower property counts in the suburb. Therefore, these variables are expected to be the most important covariates in predicting Melbourne housing values. The labels of the clustering will also be included as an additional variable.

Question 3

Perform regression models to predict the Price variable. You could consider transformations of this variable, but the prediction needs to be done on the original scale. Your analysis should at least include the following models:

- Use two penalized linear regression models with variable selection property. For example, Lasso, Ridge, elastic net, AIC, BIC, etc.
- Use two nonparametric models. For example, random forests, splines, boosting, nearest neighbors, SIR, etc.

For each model, you need to 1) discuss and tuning the parameters (if any) and select the best tuning; 2) discuss the advantages and disadvantages compared with other models you considered; 3) properly demonstrate your findings, such as estimation errors, variable selection, etc.

Answer begins:

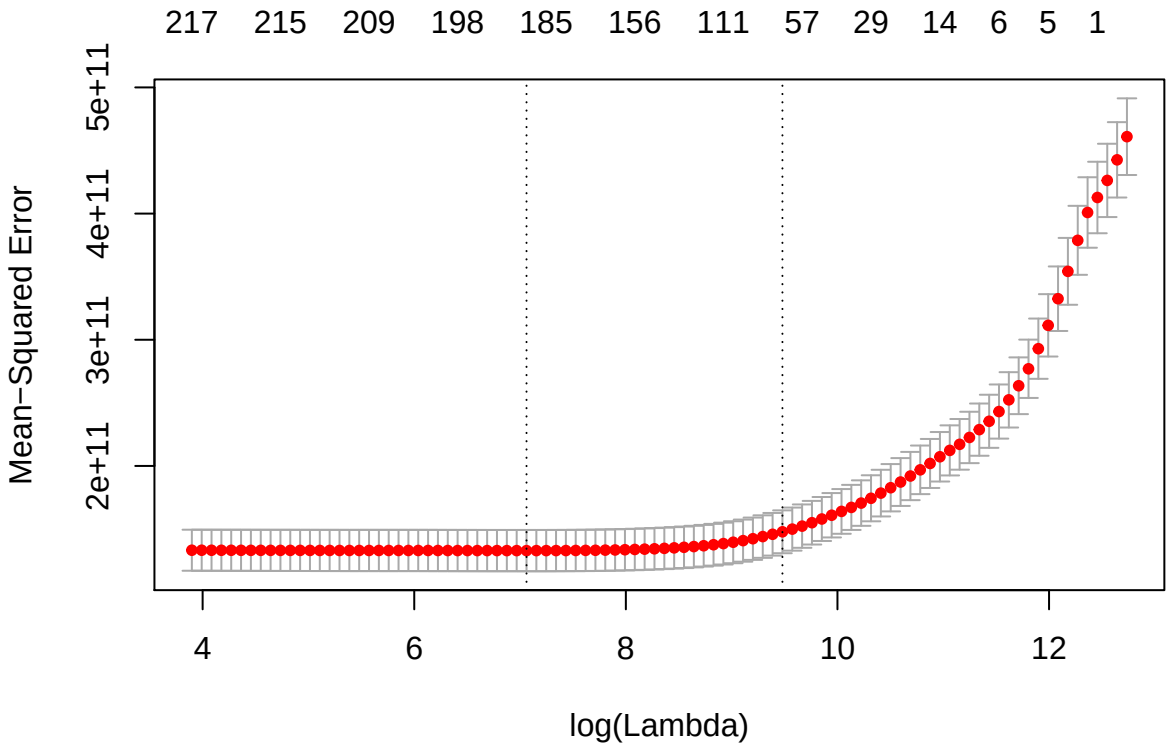
First I train-test split the data, with 75% as the training set.

```
X = model.matrix(Price ~ ., full)
y = exp(full$Price)

sim_data = data.frame(y, X)
sim_idx = createDataPartition(sim_data$y, p = 0.75, list = FALSE)
sim_trn = sim_data[sim_idx,]
sim_tst = sim_data[-sim_idx,]
```

Penalized method one: lasso

The major advantage of lasso is that it accomodates a large number of covariates, which fits our situation with a lot of factorized dummy from the `Postcode` variable. Specifically, there are 221 variables in total.

```
#first apply lasso on the dataset
fit_cv = cv.glmnet(X[sim_idx,], y[sim_idx], alpha = 1)
nonzero = which(coef(fit_cv) != 0)
plot(fit_cv)
```



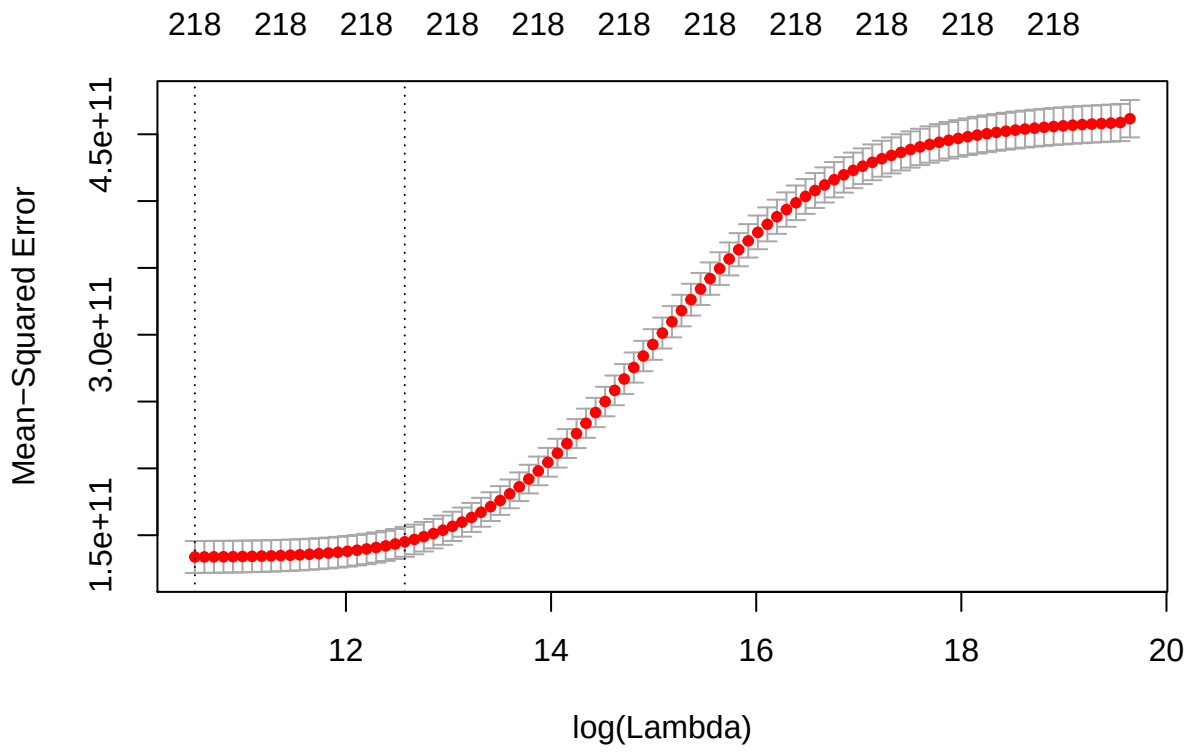
Fitting the lasso. First I use the `cv.glmnet` to tune over λ of Lasso to find the optimal λ_{min} as well the λ_{1sd} which is one standard deviation away from λ_{min}. The tuning process is displayed above. The value of λ_{min} is 1165.8, while the value of λ_{1sd} is 13095 . In total there are 71 non-zero variables.

```
cv_5 = trainControl(method = "cv", number = 5)
lasso_grid = expand.grid(alpha = 1,lambda = c(fit_cv$lambda.min, fit_cv$lambda.1se))

fit_lasso = train(
  y ~ ., data = sim_trn,
  method = "glmnet",
  trControl = cv_5,
  tuneGrid = lasso_grid
)

#Calculate the test RMSE :
calc_acc = function(actual, predicted) {
sqrt(mean((actual - predicted) ^ 2))
}
pred = predict(fit_lasso, sim_tst)
lso_acc = calc_acc(sim_tst$y, pred)
```

To further investigate ridge regression's train and test errors, I plug both two values into a `train` function,with the method specified as `glmnet` with `alpha =1`. Applying the trained model on the test set results in a testing RMSE 359168.5.

Penalized method two: Ridge regression

```
#ridge regression
rdg_cv = cv.glmnet(X[sim_idx,], y[sim_idx], alpha = 0)
plot(rdg_cv)
```




```
cv_5 = trainControl(method = "cv", number = 5)
rdg_grid = expand.grid(alpha = 0,lambda = c(rdg_cv$lambda.min, rdg_cv$lambda.1se))

fit_rdg = train(
  y ~ ., data = sim_trn,
  method = "glmnet",
  trControl = cv_5,
  tuneGrid = rdg_grid
)
```

Ridge regression Ridge regression is similar with Lasso but does not coerce coefficients to be zero. The chosen values of λ are $\lambda_{min} = 37297.1$ and $\lambda_{1sd} = 288777$ .Plugging them to the `train` model (with `alpha=0`) shows that λ_{min} has a smaller training RMSE. Applying the trained model on the test set, RMSE = 359205.

Nonparametric method one: random forest

```
#random forest
rf_grid = expand.grid(mtry = 2:5)
set.seed(676119152)
```

```
fit_rf = train(
y ~ .,
data = sim_num_trn,
method = "rf",
trControl = cv_5,
tuneGrid = rf_grid
)
save(fit_rf, file = "./models/rf.rda")
```

```
##   mtry     RMSE  Rsquared      MAE   RMSESD  RsquaredSD    MAESD
## 2    3 266284.7 0.8463444 162189.7 10633.89 0.009118734 2388.034
```

Random Forest The first chosen nonparametric model is random forest. Specifically, I tune over the parameter `mtry` from `2:5`. The parameter represent the number of variables being used in each iteration. The tuning result above shows that `mtry=4` is the optimal value. Unlike the penalized methods, I only fit numeric variables to the random forest model because the model's computational power cannot handle the excessive amount of postcode fixed effects. The training RMSE is 266284, while the testing RMSE is 194529.

Nonparametric method two: Gradient Boosting

```
gbm_grid = expand.grid(
interaction.depth = c(1, 2, 3),
n.trees = (1:30) * 100,
shrinkage = c(0.1, 0.3),
n.minobsinnode = 20
)

fit_gbm = train(
y ~ .,
data = sim_num_trn,
trControl = trainControl(method = "cv", number = 5),
method = "gbm",
tuneGrid = gbm_grid,
verbose = FALSE
)

save(fit_gbm, file = "./models/gbm.rda")
```

```
##    shrinkage interaction.depth n.minobsinnode n.trees     RMSE  Rsquared
## 80       0.1                 3             20    2000 276583.4 0.8322852
##         MAE   RMSESD RsquaredSD    MAESD
## 80 171076.8 13091.15 0.01396042 4007.947
```

Gradient boosting model The second nonparametric model is gradient boosting. Compared to random forest,gbm features a gradual learning process where the underfitted observations receive more weights in later iterations. However, gbm is likely to suffer from overfitting to which random forest is immune. In fitting the model, I tune over `interaction.depth`. `n.trees`, `shrinkage` and `n.minobsinnode`. The tuning result is shown as above. Similar to random forest, I include only numeric variables because `gbm`'s computational power is not enough to handle the fixed effects.The training RMSE is 276583.4, while the testing RMSE is 209245.

Pros and Cons of each method

The pros and cons of penalized methods The major advantage of the two penalized methods, `lasso` and `ridge`, is their handling of large number of covariates. This is helpful in our case, because the categorical variable `Postcode` creates a big number of dummy variable after factorization. However, the main disadvantage of the two models is the underlying linearity assumption: essentially the possibilities of nonlinear effects or interaction effects between variables have been omitted.

The pros and cons of tree methods The non-parametric methods , `random forest` and `gbm`, have the advantage of incorporating nonlinear effects or interaction effects. However, these two models are overwhelmed by the larges number of fixed effects in our case and not capable to generate any results. To use these models I had to remove the fixed effects of `Postcode` and fitted only numeric variables. This potentially reduces model accuracy.

Question 4

Based on all of the analyses above, can you suggest a new approach or a modification/integration of the approaches to improve the performance? Before implementing this new approach, you should particularly discuss:

- Intuitively, why the model is likely to improve upon the existing models?
- Is there any computational challenges and if there is, how to address them?

Now, implement this method, and summarize your findings. Note that your new approach does not necessarily have to improve the accuracy. If there are computational difficulties that you cannot overcome, then consider a smaller subset of the data, such as one sub-region of the city.

Answer Begins:

To keep the advantages and address the shortcomings of the four models, I will apply the `xgbboost` model . The `xgboost` method is a tree-based approach known for its strong computational power. In our case it is likely to improve model performance because it automatically incorporates any nonlinear effects, like random forest and gbm, but is not limited to including only numeric data. Therefore, it is an improvement over the four models. In the codes below I apply this model to the full set of variables after data preprocessing.

```
#xgbboost
set.seed(676119152)
fit_xgb = train(
  y ~ ., data = sim_trn,
  method = "xgbLinear",
  trControl = cv_5,
  tuneLength = 3
)
save(fit_xgb, file = "./models/xgb.rda")
```

```
fit_xgb$results[rownames(fit_xgb$bestTune), ]
```

```
##   lambda alpha nrounds eta   RMSE  Rsquared      MAE   RMSESD RsquaredSD
## 6      0 1e-04     150 0.3 301733 0.8095496 172121.7 28927.05 0.03218077
##      MAESD
## 6 4428.696
```

```
pred = predict(fit_xgb, sim_tst)
xgb_acc = calc_acc(sim_tst$y, pred)
```

The table above displays the tuning results of `xgboost`. The training RMSE is 301733, and the testing RMSE is 131208.

comparing model performances

	Tst Err
lasso	359168.5
ridge reg	359205.0
random forest	194529.3
gbm	209245.6
xgboost	131208.0

The above table compares the model performance of the five models: two penalized methods, two non-parametric methods, and `xgboost`. The results show that the penalized methods have the lowest accuracy. The two tree models outperform the penalized models, while `xgboost` further improves upon the other tree models after taking postcode fixed effects into account.