RentalPredictions_Joshua_Taubman

2022-10-23

In this project we will be predicting rental prices for residences across India.

Data set reference: Banerjee, Sourav. "House Rent Prediction Dataset", *Kaggle,* Accessed: 23/10/2022.

Move the attached CSV file into the default directory for your Rstudio

```
dat<- read.csv("House_Rent_Dataset.csv")
```

Create Training and Test Sets

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
```

```
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## v purrr   0.3.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
```

```
test_indexes <- createDataPartition(y = dat$Rent,times = 1, 0.1, list = FALSE)
test_data <- dat[test_indexes,]
train_data <- dat[-test_indexes,]
```

RMSE function

```
rmse<- function(prediction, true){
  sqrt(mean(prediction-true)^2)
  }
```

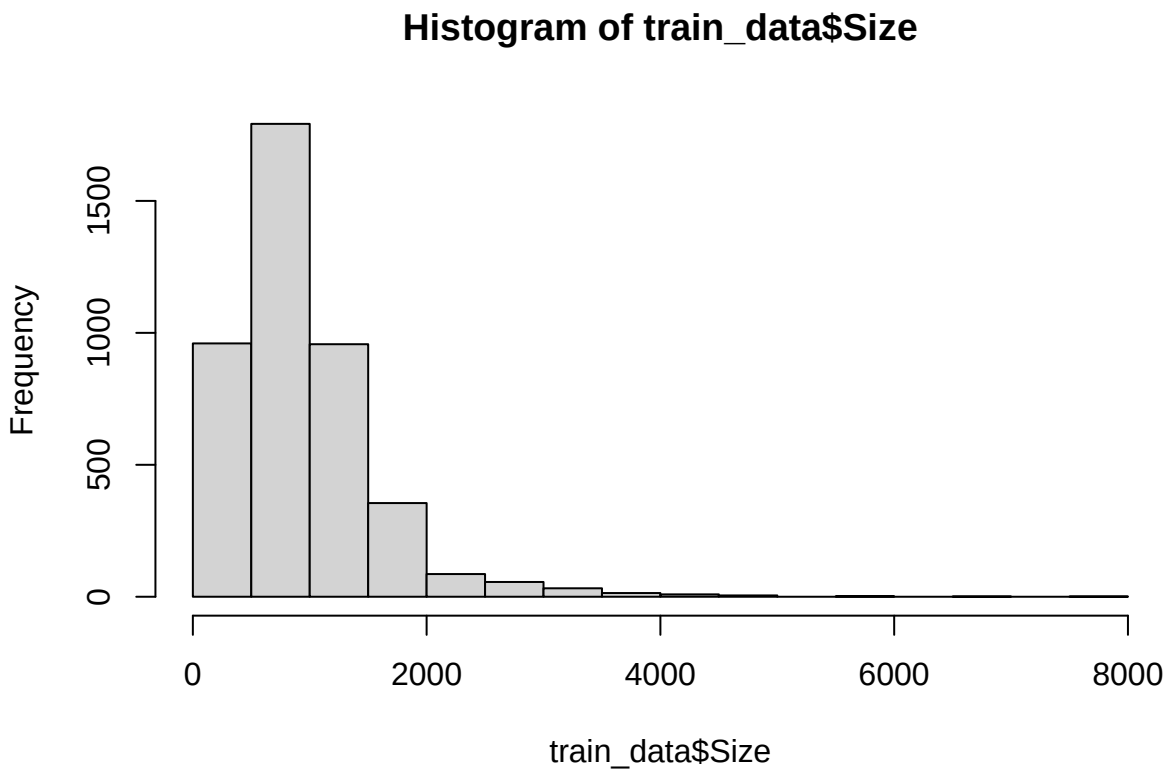
Add average across all rent for most basic predictor.

Property size brackets as predictor

To use property size as a predictor along with other things we need to transform it into a bias, which can be done by partitioning our property size into brackets.

We take the lower bound of each bracket.

```
hist(train_data$Size, breaks = 20)
```




```
level_lower_bound <- cut(train_data$Size, breaks = seq(0,8100,100),
                         labels = sapply(seq(0,8000,100), toString))
df_l <- data.frame(level_lower_bound)
colnames(df_l) <- "level_lwer_bound"
train_data <- train_data %>% cbind(df_l, .)
```

Replicate process but this time for test data

```
level_lower_bound_test <- cut(test_data$Size, breaks = seq(0,8100,100),
                              labels = sapply(seq(0,8000,100), toString))
df_l_t <- data.frame(level_lower_bound_test)
colnames(df_l_t) <- "level_lwer_bound"
test_data <- test_data %>% cbind(df_l_t, .)
```

Bind on this new classification of property size bracket

```
mu <- mean(train_data$Rent)
level_averages <- train_data %>%
  group_by(level_lwer_bound) %>%
  summarise(level_avg = mean(Rent-mu))


train_data <- train_data %>%
  left_join(level_averages, by = "level_lwer_bound")

test_data <- test_data %>%
  left_join(level_averages, by = "level_lwer_bound")
```

We have a naive linear fit just based upon property size. We will use this as a bench mark and also to fill in NAs within our data.

```
linear_fit <- lm(Rent~ Size, data = train_data)
y_hat_lm <- predict(linear_fit, newdata = test_data)
```

Replacing NAs and predicting with size brackets.

```
for (i in 1:length(test_data$level_avg)) {
  test_data$level_avg[i] <- ifelse(is.na(test_data$level_avg[i] == TRUE),
         (predict(linear_fit, newdata = test_data))[i]
         ,test_data$level_avg[i])
}

y_hat_size_strata <- mu + test_data$level_avg
```

Result RMSEs

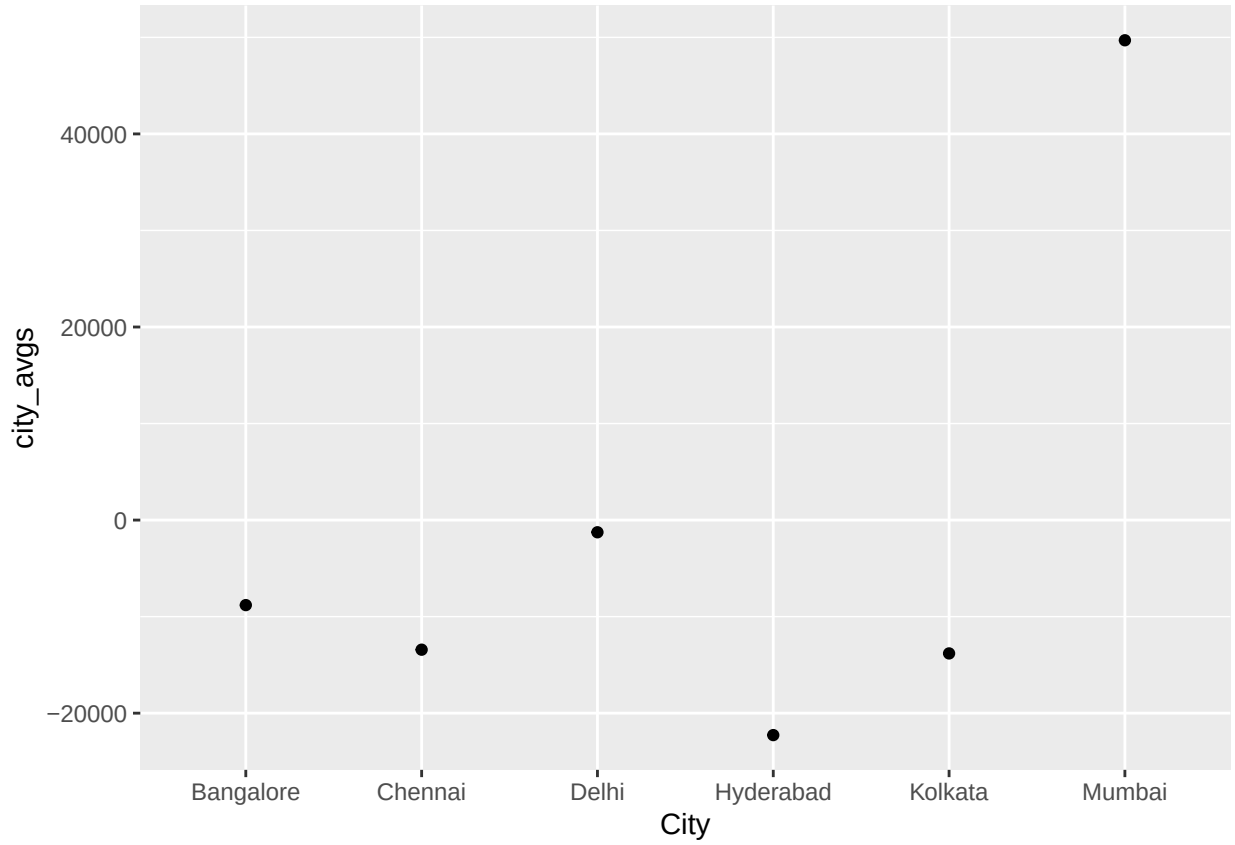
```
linear_model<- rmse(y_hat_lm, test_data$Rent)
size_strata <- rmse(y_hat_size_strata, test_data$Rent)


rmse_results <- data.frame(method = "Linear Regression on Size",
                           RMSE = linear_model)

rmse_results <- bind_rows(rmse_results,
                          data.frame(method= "Property Size Brackets",
                                     RMSE = size_strata))
rmse_results %>% knitr::kable()
```

method	RMSE
Linear Regression on Size	1924.819
Property Size Brackets	1098.706

City Averages as a Predictor

Add in averages across each city listed then add these as an additional bias.

```
city_averages <- train_data %>%
  group_by(City) %>%
  summarise(city_avgs = mean(Rent - level_avg- mu))


city_averages %>%
  ggplot(aes(City,city_avgs))+
  geom_point()
```




```
train_data <- train_data %>%
  left_join(city_averages, by ="City")

#Model taking into account biases of city and size strata
test_data <- test_data %>%
  left_join(city_averages, by = "City")

y_hat_3 <- mu + test_data$city_avgs + test_data$level_avg
strat_cit <- rmse(y_hat_3, test_data$Rent)

rmse_results <- bind_rows(rmse_results,
                          data.frame(method= "Size Bracket and City",
                                     RMSE = strat_cit))

rmse_results %>% knitr::kable()
```

method	RMSE
Linear Regression on Size	1924.819
Property Size Brackets	1098.706
Size Bracket and City	3646.952

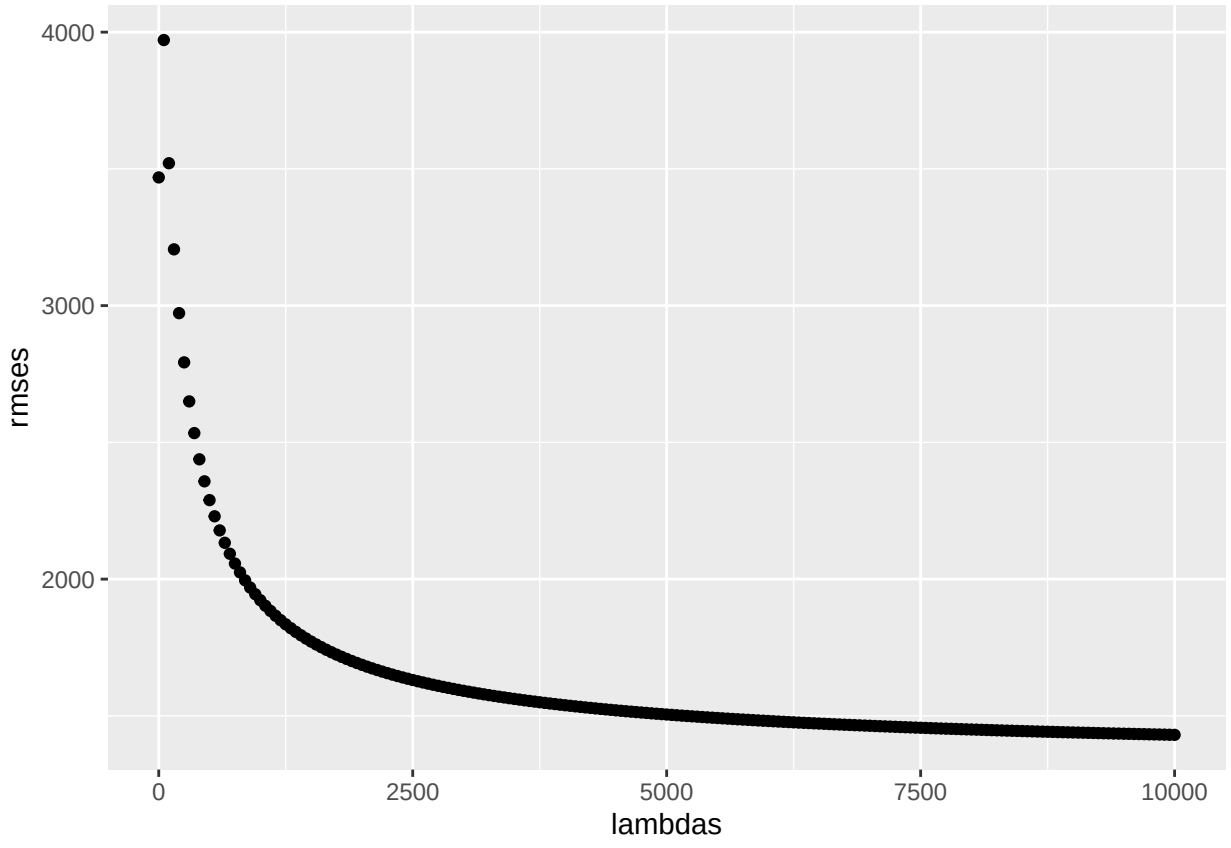
Finding Optimal Lambda for Regularisation

```
lambdas <- seq(0, 10^4, 50)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_data$Rent)
  level_average_reg <- train_data %>%
#Can't use group by either
    group_by(level_lwer_bound) %>%
    summarize(level_avg_reg = sum(Rent - mu)/(n()+l))

#left join will be problematic
#can't left join on a list but what exactly does this bug mean?

#colnames(level_average_reg) <- level_avg_reg
  city_average_reg <- train_data %>%
    group_by(level_lwer_bound) %>%
    left_join(level_average_reg, by = "level_lwer_bound") %>%
    ungroup() %>%
    group_by(City) %>%
    summarize(city_avg_reg = sum(Rent - level_avg_reg - mu)/(n()+l))
  predicted_rent <-
    test_data %>%
    group_by(level_lwer_bound) %>%
    left_join(level_average_reg, by = "level_lwer_bound") %>%
    ungroup() %>%
    group_by(City)%>%
    left_join(city_average_reg, by = "City") %>%
    mutate(pred = mu + level_avg_reg + city_avg_reg) %>%
    .$pred

  #Replace the few NAs with linear fit
  for (i in 1:length(predicted_rent)) {
    predicted_rent[i] <- ifelse(is.na(predicted_rent[i] == TRUE |
                                        predicted_rent[i] <= 0),
                                (predict(linear_fit, newdata = test_data))[i]
                                ,predicted_rent[i])
  }
  return(rmse(predicted_rent, test_data$Rent))
})
rmses
```

```
##   [1] 3468.809 3971.161 3520.651 3205.868 2972.536 2792.636 2649.864 2533.949
##   [9] 2438.061 2357.477 2288.832 2229.662 2178.129 2132.834 2092.696 2056.867
##  [17] 2024.674 1995.577 1969.135 1944.990 1922.844 1902.448 1883.595 1866.107
##  [25] 1849.835 1834.651 1820.442 1807.114 1794.582 1782.774 1771.626 1761.080
##  [33] 1751.086 1741.600 1732.583 1723.997 1715.812 1707.999 1700.531 1693.384
##  [41] 1686.539 1679.974 1673.672 1667.617 1661.794 1656.190 1650.791 1645.586
```

```
##  [49] 1640.564 1635.715 1631.031 1626.502 1622.121 1617.880 1613.772 1609.791
##  [57] 1605.931 1602.187 1598.552 1595.023 1591.593 1588.260 1585.019 1581.865
##  [65] 1578.796 1575.808 1572.898 1570.062 1567.298 1564.603 1561.974 1559.409
##  [73] 1556.905 1554.460 1552.073 1549.741 1547.461 1545.233 1543.055 1540.924
##  [81] 1538.840 1536.800 1534.804 1532.850 1530.936 1529.061 1527.225 1525.425
##  [89] 1523.661 1521.932 1520.236 1518.573 1516.942 1515.341 1513.771 1512.230
##  [97] 1510.716 1509.231 1507.772 1506.339 1504.931 1503.548 1502.189 1500.853
## [105] 1499.540 1498.249 1496.980 1495.731 1494.504 1493.296 1492.108 1490.938
## [113] 1489.788 1488.655 1487.541 1486.443 1485.363 1484.298 1483.250 1482.218
## [121] 1481.201 1480.200 1479.213 1478.240 1477.281 1476.336 1475.405 1474.487
## [129] 1473.581 1472.689 1471.808 1470.940 1470.084 1469.239 1468.406 1467.583
## [137] 1466.772 1465.971 1465.181 1464.401 1463.631 1462.871 1462.121 1461.381
## [145] 1460.649 1459.927 1459.214 1458.509 1457.813 1457.126 1456.447 1455.776
## [153] 1455.113 1454.459 1453.811 1453.172 1452.540 1451.915 1451.298 1450.687
## [161] 1450.084 1449.487 1448.897 1448.314 1447.737 1447.167 1446.603 1446.045
## [169] 1445.493 1444.948 1444.408 1443.873 1443.345 1442.822 1442.305 1441.793
## [177] 1441.287 1440.785 1440.289 1439.798 1439.312 1438.831 1438.355 1437.884
## [185] 1437.417 1436.955 1436.498 1436.045 1435.597 1435.153 1434.713 1434.278
## [193] 1433.846 1433.419 1432.996 1432.577 1432.162 1431.751 1431.343 1430.940
## [201] 1430.540
```

```
qplot(lambdas, rmses)
```




```
ind<- which.min(rmses)
lambda_best <- (ind-1) *50
lambda_best
```

```
## [1] 10000
```

Update RMSE table with optimal regularisation factor

```
rmse_results <- bind_rows(rmse_results,
                          data.frame(method= paste("Regularisation, lambda =",
                                     lambda_best),
                                     RMSE = rmses[ind]))
rmse_results %>% knitr::kable()
```

method	RMSE
Linear Regression on Size	1924.819
Property Size Brackets	1098.706
Size Bracket and City	3646.952
Regularisation, lambda = 10000	1430.540

```
just_mu <- rmse(mu, test_data$Rent)

rmse_results <- bind_rows(rmse_results,
                          data.frame(method= "Just using average rent",
                                     RMSE = just_mu))
```

Bootstrapping

Due to the variability in RMSE based upon which seed is selected and each time the program is run, we bootstrap our train data to prevent over-fitting in our regularisation and to provide a more stable prediction of the test set.

We recreate our predictors after resampling our data.

```
indexes_boot <- createResample(train_data$Rent, 10)
# indexes_boot
#What we had was we were splitting the data based on columns not rows
# length(train_data)
#Test out bootstrapping, just calculating mu

mu_boot <- sapply(indexes_boot, function(ind){
  rent_boot<- train_data$Rent[ind]
  mean(rent_boot)
})
# mu_boot
# mean(mu_boot)
# mean(mu_boot)-mu
#Why are mu and mean(mu_boot) so different???
# train_data$Rent
# mu
y_hat_mu_boot <- mean(mu_boot)
rmse_mu_boot <- rmse(y_hat_mu_boot, test_data$Rent)
# rmse_mu_boot
```

```
rmse_results <- bind_rows(rmse_results,
                          data.frame(method= "Bootstrap using just Mu",
                                     RMSE = rmse_mu_boot))

rmse_results %>% knitr::kable()
```

method	RMSE
Linear Regression on Size	1924.819
Property Size Brackets	1098.706
Size Bracket and City	3646.952
Regularisation, lambda = 10000	1430.540
Just using average rent	1032.446
Bootstrap using just Mu	1159.192

```
levels_boot <- sapply(indexes_boot, function(ind){
  level_boot<- train_data$level_avg[ind]
  rent_l_boot <- mu - level_boot
  mean(rent_l_boot)
})
y_hat_mu_l_boot <- mean(levels_boot)
rmse_mu_l_boot <- rmse(y_hat_mu_l_boot, test_data$Rent)

rmse_results <- bind_rows(rmse_results,
                          data.frame(method= "Bootstrap with Size Level",
                                     RMSE = rmse_mu_l_boot))

rmse_results %>% knitr::kable()
```

method	RMSE
Linear Regression on Size	1924.819
Property Size Brackets	1098.706
Size Bracket and City	3646.952
Regularisation, lambda = 10000	1430.540
Just using average rent	1032.446
Bootstrap using just Mu	1159.192
Bootstrap with Size Level	1085.804

```
# #Now we do the same for regularisation
# #We will need to still take a value for Mu (do we select the median?)
# rmses
# #Resampled indexes
# indexes_boot
```

When finding the optimal regularisation coefficient we need to use an sapply/for loop to step through all of the different sampling possiblities. We need to take row means of our results to average over each bootstrap. Limited the range of this lambda max size for computational time reasons.

```
######## rmse function using the relevant lambdas
#lambdas only go up to 10^3 to reduce run time
lambdas<- seq(0, 10^3, 50)
rmse_boot<- sapply(indexes_boot, function(ind){
    sapply(lambdas, function(l){
  mu <- mean(train_data$Rent[ind])
  #To select the rows based upon index we need to turn our train and test data
  # into data frames
  train_df <- train_data %>% data.frame(.)
  test_df <- test_data %>% data.frame(.)
  train_df_boot <- train_df[ind,]
  test_df_boot <- test_df[-ind,]

  level_average_reg <- train_df_boot %>%
    #Can't use group by either
    group_by(level_lwer_bound) %>%
    summarize(level_avg_reg = sum(Rent - mu)/(n()+l))

  #left join will be problematic
  #can't left join on a list but what exactly does this bug mean?

  #colnames(level_average_reg) <- level_avg_reg
  city_average_reg <- train_df_boot %>%
    group_by(level_lwer_bound) %>%
    left_join(level_average_reg, by = "level_lwer_bound") %>%
    ungroup() %>%
    group_by(City) %>%
    summarize(city_avg_reg = sum(Rent - level_avg_reg - mu)/(n()+l))
  predicted_rent <-
    test_df_boot %>%
    group_by(level_lwer_bound) %>%
    left_join(level_average_reg, by = "level_lwer_bound") %>%
    ungroup() %>%
    group_by(City)%>%
    left_join(city_average_reg, by = "City") %>%
    mutate(pred = mu + level_avg_reg + city_avg_reg) %>%
    .$pred

  #Replace the few NAs with linear fit
  for (i in 1:length(predicted_rent)) {
    predicted_rent[i] <- ifelse(is.na(predicted_rent[i] == TRUE |
                                        predicted_rent[i] <= 0),
                                (predict(linear_fit, newdata = test_data))[i]
                                ,predicted_rent[i])
  }
  #predicted_rent_avg<- mean(predicted_rent)
  #print(predicted_rent)
  predicted_rent_avg<- mean(predicted_rent)
  return(rmse(predicted_rent_avg, test_data$Rent))
})
})
```

```
rmse_boot_mat<- as.matrix(rmse_boot)
average_accross_boots <- rowMeans(rmse_boot_mat)
average_accross_boots
```

```
##  [1] 3162.745 3096.862 2722.530 2452.270 2247.237 2086.210 1956.415 1849.614
##  [9] 1760.222 1689.509 1635.348 1595.085 1560.273 1529.918 1503.246 1479.646
## [17] 1458.633 1439.814 1422.870 1407.540 1393.607
```

```
ind_b<- which.min(average_accross_boots)
ind_b
```

```
## [1] 21
```

```
lambda_best_b <- (ind_b-1) *50
lambda_best_b
```

```
## [1] 1000
```

```
rmse_results <- bind_rows(rmse_results,
                          data.frame(method= paste("Reg with Boot, lambda =",
                                                   lambda_best_b),
                                     RMSE = average_accross_boots[ind_b]))
rmse_results %>% knitr::kable()
```

method	RMSE
Linear Regression on Size	1924.819
Property Size Brackets	1098.706
Size Bracket and City	3646.952
Regularisation, lambda = 10000	1430.540
Just using average rent	1032.446
Bootstrap using just Mu	1159.192
Bootstrap with Size Level	1085.804
Reg with Boot, lambda = 1000	1393.607

Here are the final RMSE results

```
rmse_results
```

```
##                           method     RMSE
## 1      Linear Regression on Size 1924.819
## 2         Property Size Brackets 1098.706
## 3          Size Bracket and City 3646.952
## 4 Regularisation, lambda = 10000 1430.540
## 5        Just using average rent 1032.446
## 6        Bootstrap using just Mu 1159.192
## 7      Bootstrap with Size Level 1085.804
## 8   Reg with Boot, lambda = 1000 1393.607
```