Data 607 - Project 2

Frank Namin

Introduction

In this project our task is use some practice data sets to improve our ability in manipulating different formats. In particular our mission is to

1. Select three "wide" datasets. For every one of the three selected datasets:
    - Create a.CSV file (or, optionally, a MySQL database!) containing all of the dataset's information.
    - Import the contents of your.CSV file into R, and use tidyr and dplyr to clean and transform your data as necessary.
    - Conduct the analysis outlined in the discussion item.
2. For each of the three datasets submit:
    - The URL to the.Rmd file in your GitHub repository
    - The URL to your rpubs.com web page.

We start with a brief description of wide and long data

Wide and Long Datasets

In wide data formats each variable is a column and each participant a row. As a result, wide format contains values that do not repeat in the first column.

In long format, each participant has multiple rows in dataset. Hence, a long format has values that do repeat in the first column.

Suppose we want a dataset which includes the final grades of students in 3 subject. The wide and long formats are shown below.

Student ID	math	physics	history
110	94	89	79
128	94	93	87
155	73	74	99
196	92	89	99
197	76	81	72

Figure 1: wide format

Datasets

We consider the following three datasets

- California Housing Prices
- Airbnb Dataset
- Air Quality

California Housing Prices

- This is a publicly available dataset that contains information about housing prices and other factors in California. The dataset was created in 1997 by Pace, R. Kelley and Ronald Barry, and is based on the 1990 California census data. The dataset contains 20,640 samples and 8 features, including:
  - Median income
  - Housing median age
  - Total rooms
  - Total bedrooms
  - Population
  - Households
  - Latitude
  - Longitude
  - Median house value

We start by loading the required libraries

Student ID	subject	grade
110	math	96
110	physics	87
110	history	78
128	math	100
128	physics	97
128	history	87
155	math	81
155	physics	93
155	history	88
196	math	88
196	physics	83
196	history	73
197	math	86
197	physics	91
197	history	90

Figure 2: long format

```r
# Library Imports
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidyr)
library(stringr)
library(knitr)
library(ggplot2)
library(zoo)
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
library(plotly)
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```r
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library(grid)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

Next we read the .CSV file into a dataframe and display the top rows

```r
CA <- read.csv("housing.csv")
head(CA)
```

```
##   longitude latitude housing_median_age total_rooms total_bedrooms population
## 1   -122.23    37.88                41         880            129        322
## 2   -122.22    37.86                21        7099           1106       2401
## 3   -122.24    37.85                52        1467            190        496
## 4   -122.25    37.85                52        1274            235        558
## 5   -122.25    37.85                52        1627            280        565
## 6   -122.25    37.85                52         919            213        413
##   households median_income median_house_value
## 1        126        8.3252             452600
## 2       1138        8.3014             358500
## 3        177        7.2574             352100
## 4        219        5.6431             341300
## 5        259        3.8462             342200
## 6        193        4.0368             269700
```

All the attributes are numerical.

Cleaning the Data

First we would like to see which attributes have missing or non-numerical values.

```r
names(CA)[which(colSums(is.na(CA)) > 0)]
```

```
## [1] "total_bedrooms"
```

The total bed_bedrooms is the only column with missing value. We can use the column mean to fill the missing values, but since the number of bedrooms is an integer, median is a better choice.

```r
# Fill missing values with median of column
CA_filled <- CA %>% mutate_all(list(~ifelse(is.na(.), median(., na.rm = TRUE), .)))
```

Analysis

We start by looking at the summary statistics of the data

```r
summary(CA_filled)
```

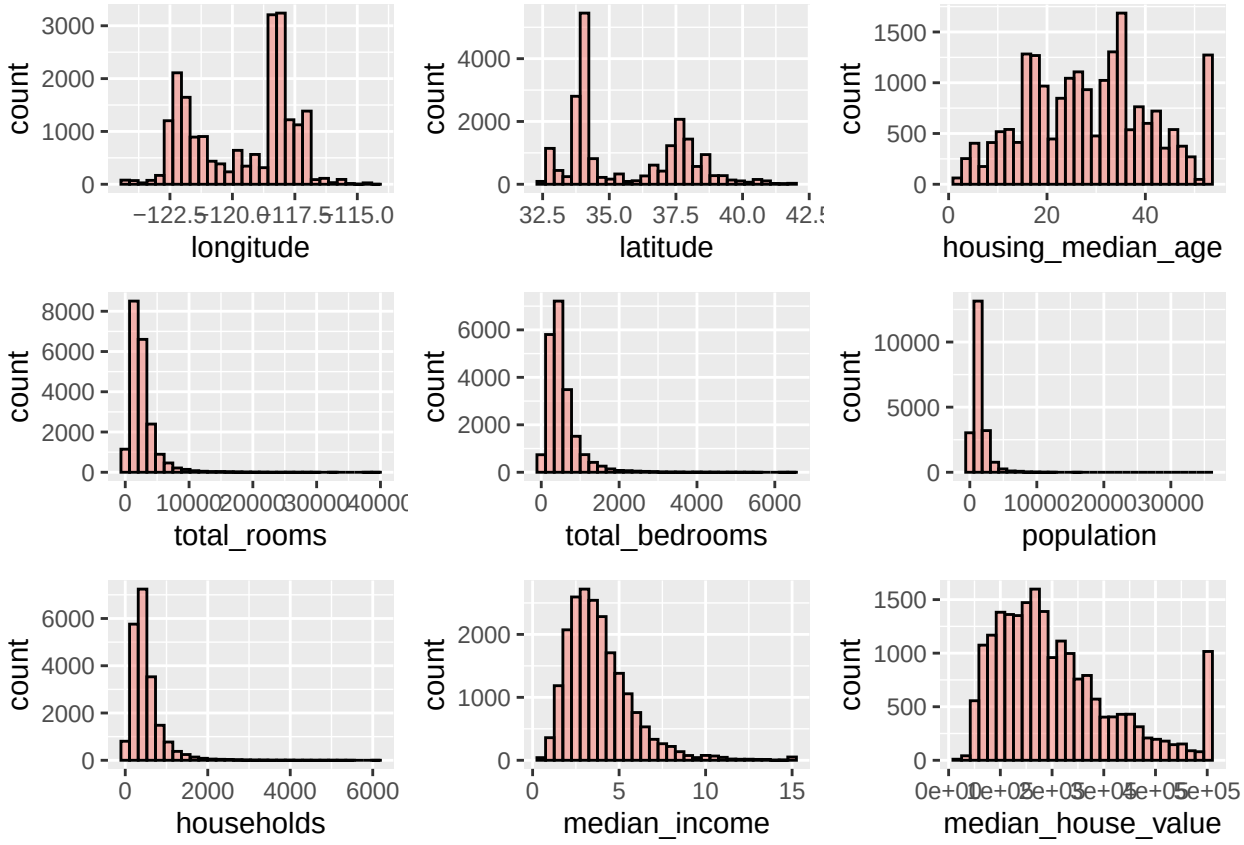
```
##    longitude         latitude     housing_median_age  total_rooms
##  Min.   :-124.3   Min.   :32.54   Min.   : 1.00      Min.   :    2
##  1st Qu.:-121.8   1st Qu.:33.93   1st Qu.:18.00      1st Qu.: 1448
##  Median :-118.5   Median :34.26   Median :29.00      Median : 2127
##  Mean   :-119.6   Mean   :35.63   Mean   :28.64      Mean   : 2636
##  3rd Qu.:-118.0   3rd Qu.:37.71   3rd Qu.:37.00      3rd Qu.: 3148
##  Max.   :-114.3   Max.   :41.95   Max.   :52.00      Max.   :39320
##  total_bedrooms     population      households     median_income
##  Min.   :   1.0   Min.   :    3   Min.   :   1.0   Min.   : 0.4999
##  1st Qu.: 297.0   1st Qu.:  787   1st Qu.: 280.0   1st Qu.: 2.5634
##  Median : 435.0   Median : 1166   Median : 409.0   Median : 3.5348
##  Mean   : 536.8   Mean   : 1425   Mean   : 499.5   Mean   : 3.8707
##  3rd Qu.: 643.2   3rd Qu.: 1725   3rd Qu.: 605.0   3rd Qu.: 4.7432
##  Max.   :6445.0   Max.   :35682   Max.   :6082.0   Max.   :15.0001
##  median_house_value
##  Min.   : 14999
##  1st Qu.:119600
```

```
##  Median :179700
##  Mean   :206856
##  3rd Qu.:264725
##  Max.   :500001
```

To get an idea about the distribution of the data, we can also look at the histograms

```
plots <- lapply(names(CA_filled), function(x) ggplot(CA_filled, aes(x = .data[[x]], fill = x)) +
    geom_histogram(alpha = 0.5, position = "identity", color = "black") +
    scale_fill_discrete(name = NULL, guide = FALSE))

# Create a 3 by 3 grid of plots
grid.arrange(grobs = plots, ncol = 3)
```

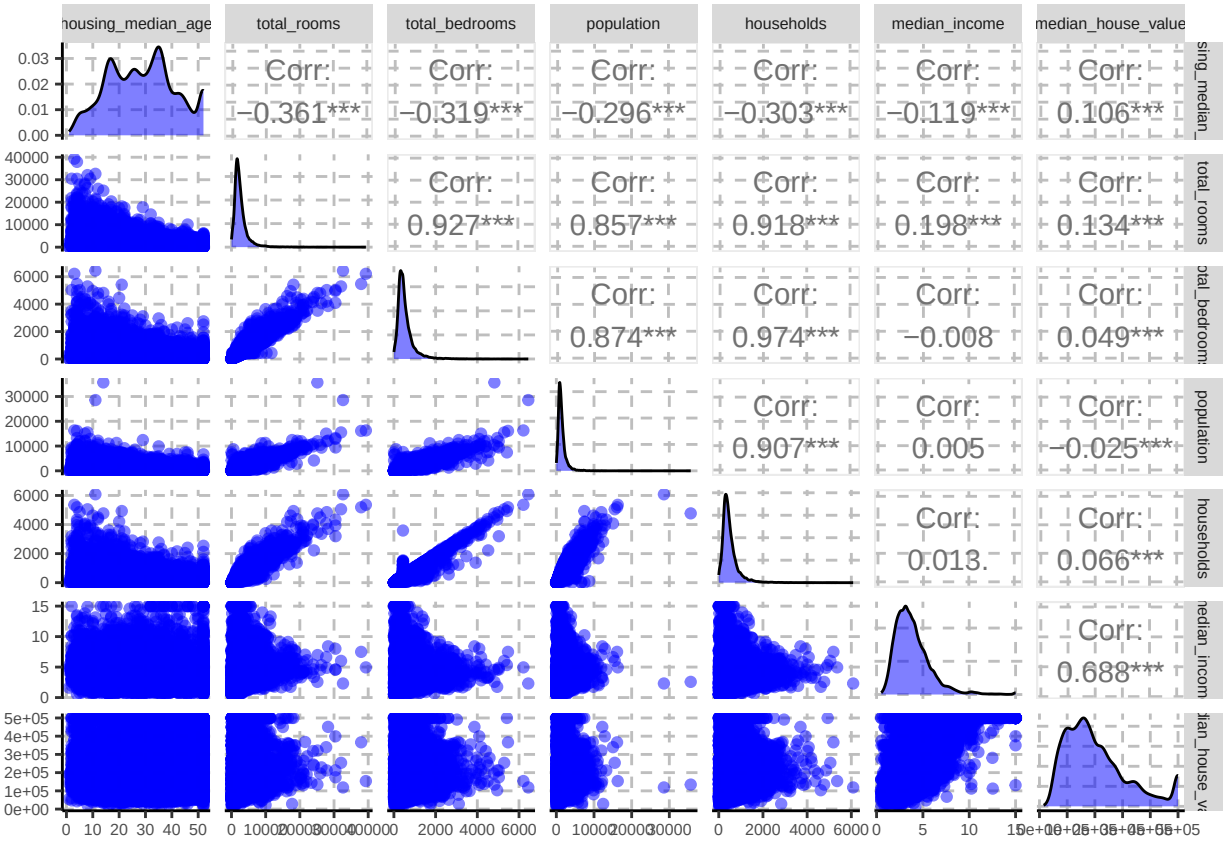


When given a multivariate dataset, one can get great insight by looking at the covariance matrix. For this purpose, we will ignore the first two columns

```
# Select the last 7 columns of the dataframe
CA_filled_subset <- CA_filled[, 3:9]


# Generate the scatter matrix with blue color
ggpairs(CA_filled_subset,
      lower = list(continuous = wrap("points", alpha = 0.5, color = "blue")),
      diag = list(continuous = wrap("densityDiag", fill = "blue", alpha = 0.5))) +
  theme(panel.background = element_rect(fill = "white", colour = "white"),
      panel.grid.major = element_line(colour = "gray", linetype = "dashed"),
      panel.grid.minor = element_blank(),
      axis.line = element_line(colour = "black"),
```

```
      text = element_text(size = 7))
```




```
# Save the plot as png
ggsave("scatter_matrix.png", width = 7, height = 7, dpi = 150)
```

Data Transformation

In general it is always a good idea to normalize our data. There are several ways to do this. Suppose our data is $x_1, x_2, \cdots, x_n$ with sample mean

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

and sample variance

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n}(x_i - \hat{\mu})^2}{n-1}$$

One way to transform our data is

$$Y = \frac{X - \mu}{\hat{\sigma}}$$

This way the transformed data will have zero mean and unit variance.

Let $x_{max} = \max(x_1, x_2, \cdots, x_n)$ and $x_{min} = \min(x_1, x_2, \cdots, x_n)$. Another transform will map our data either to $[0, 1]$ or $[-1, 1]$ depending on the data. Generally if the data is positive we map to $[0, 1]$ and if it is positive and negative we map to $[-1, 1]$.

The mapping to $[0, 1]$ is given by the transform

$$Y = \frac{X - x_{min}}{x_{max} - x_{min}}$$

In general, we can map from $[x_{min}, x_{max}]$ to any arbitrary interval $[a, b]$ using the transform

$$Y = a + \frac{(X - x_{min})(b - a)}{x_{max} - x_{min}}$$

Besides normalization there are other types of transformation that can be very useful. Looking at our distributions, we note that population and house holds have very skewed distributions. In general we would like our variable to have normal distributions. There are several statistical methods to make a distribution more normal. Probably the simplest one is the log transform.

$$Y = \ln X$$

This transform is not always ideal especially if we have negative values, however in our case this is not the case.

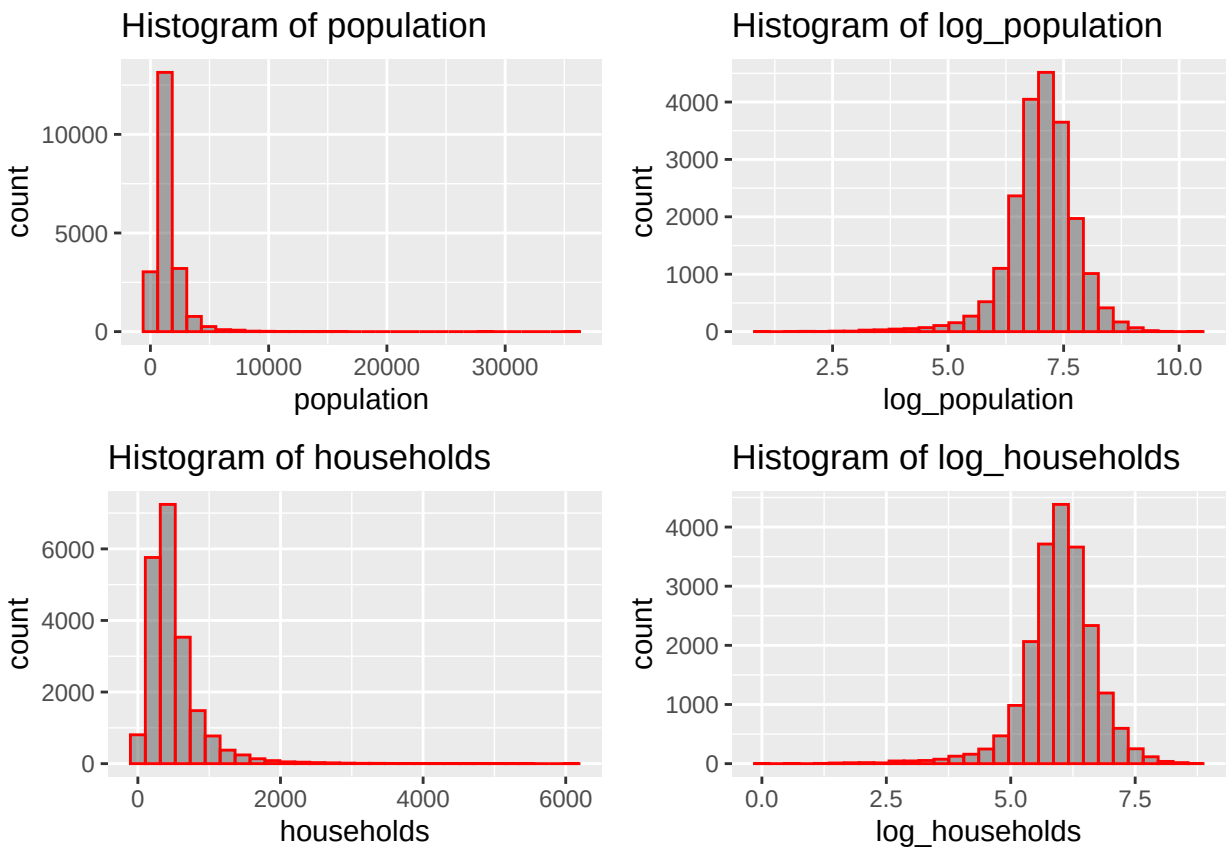
```
CA_filled$log_population <- log(CA_filled$population)
CA_filled$log_households <- log(CA_filled$households)

# Create a list of ggplot objects for each column with original and log-transformed histograms

plots <- lapply(c("population", "log_population", "households", "log_households"), function(x) {
    ggplot(CA_filled, aes(x = .data[[x]])) +
        geom_histogram(alpha = 0.5, position = "identity", color = "red") +
        scale_x_continuous(name = x) +
        ggtitle(paste("Histogram of", x))
})

# Create a 2 by 2 grid of plots
grid.arrange(grobs = plots, ncol = 2)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



As you can see the transformed variables are much more normal and symmetric.

Airbnb Dataset

This dataset contains a total of 279713 records in three .CSV files. As the first step we will merge the files.

```
# Load the csv files as dataframes
airbnb_listing_price <- read.csv("airbnb_listing_price.csv")
airbnb_location_info <- read.csv("airbnb_location_info.csv")
airbnb_property_info <- read.csv("airbnb_property_info.csv")
# Merge the dataframes based on listing_id
merged_df <- merge(airbnb_listing_price, airbnb_location_info, by = "listing_id")
merged_df <- merge(merged_df, airbnb_property_info, by = "listing_id")
# delete the seperate files to free up memory
rm(airbnb_listing_price, airbnb_location_info, airbnb_property_info)
# We only going to keep 7 columns
new_df <- merged_df %>%
    select(listing_id, price, city, name, property_type, room_type, bedrooms)
# delete the merged df
rm(merged_df)
```

To clean the data and remove the missing values, first we determine the columns with missing values

```
# get the count of missing or NA values in each column
missing_count <- colSums(is.na(new_df))

# get the names of columns with missing or NA values
missing_cols <- names(missing_count[missing_count > 0])
```

```
# print the names of columns with missing or NA values
print(missing_cols)
```

```
## [1] "name"     "bedrooms"
```

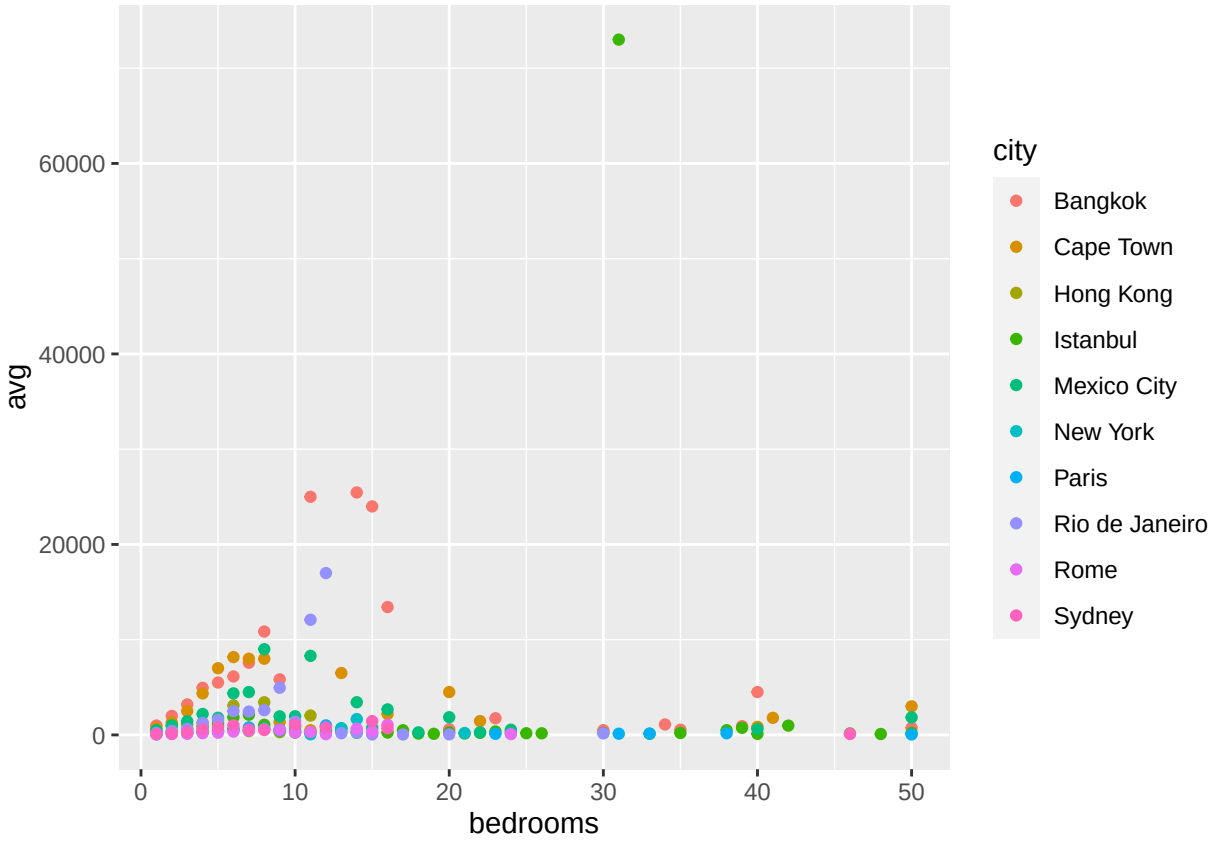
So the bedrooms column contains missing values. We will fill the missing values with the median of bedrooms since we would like an integer.

```
new_df <- new_df %>%
    mutate(bedrooms = ifelse(is.na(bedrooms), median(bedrooms, na.rm = TRUE), bedrooms))
```

Analysis

For analysis our aim is to find the most and least expensive of the cities for one, two, three, and greater than three bedrooms. It might be helpful to visualize the results first.

```
ggplot(new_df %>%
    group_by(bedrooms, city) %>%
    summarise(avg = median(price)),
    aes(x=bedrooms, y=avg, color=city)
) +
geom_point(stat='identity')
```




```
ggsave("airbnb.png", dpi=300)
```

We start by finding the most and least expensive one bedroom rentals.

```
one_bedroom_df <- new_df %>%
    filter(bedrooms == 1)
```

```
    # get the row with the highest price
highest_price_row <- one_bedroom_df %>%
        slice(which.max(price))

    # get the row with the lowest price
lowest_price_row <- one_bedroom_df %>%
        slice(which.min(price))

    # print the rows with the highest and lowest price
print(highest_price_row)
```

```
##   listing_id  price           city                                    name
## 1   13879989 625216 Rio de Janeiro Temporary rentals for Brazilian Cup.
##          property_type   room_type bedrooms
## 1 Shared room in house Shared room        1
```

```
print(lowest_price_row)
```

```
##   listing_id price     city                                  name
## 1   40560656     0 New York The Hoxton, Williamsburg - Cosy Room
##            property_type  room_type bedrooms
## 1 Room in boutique hotel Hotel room        1
```

Next, we consider two bedroom rentals.

```
two_bedroom_df <- new_df %>%
    filter(bedrooms == 2)
    # get the row with the highest price
highest_price_row <- two_bedroom_df %>%
        slice(which.max(price))

    # get the row with the lowest price
lowest_price_row <- two_bedroom_df %>%
        slice(which.min(price))

    # print the rows with the highest and lowest price
print(highest_price_row)
```

```
##   listing_id  price        city                                    name
## 1   23895520 350000 Mexico City Habitacion a 40min del centro historico
##           property_type    room_type bedrooms
## 1 Private room in house Private room        2
```

```
print(lowest_price_row)
```

```
##   listing_id price      city
## 1   43345284     1 Hong Kong
##
## 1 Hong Kong Ã§Å"â€¹Ã¥Â¾â€"Ã¨Ã¦â€¹Ã©Â¢Ã¨Ã¦â„¢Ã¯Ã§Å¡â€žÃ¦†Â¿Ã¥ÂÂ\u0090 . Ã¤Â°Â«Ã¥Â\u008fâ€"Ã¥Â¤Ã§Ã¨â€
##   property_type    room_type bedrooms
## 1          Tent Entire place        2
```

Finally we consider rentals with more than two rooms.

```
# select all the rows with rooms more than two
more_than_two_rooms_df <- new_df %>%
    filter(bedrooms > 2)
```

```r
# get the row with the highest price
highest_price_row <- more_than_two_rooms_df %>%
    slice(which.max(price))

# get the row with the lowest price
lowest_price_row <- more_than_two_rooms_df %>%
    slice(which.min(price))

# print the rows with the highest and lowest price
print(highest_price_row)
```

```
##   listing_id  price    city                                name property_type
## 1   12056763 300000 Bangkok Charoenkrung Road House/Office Rent  Entire house
##      room_type bedrooms
## 1 Entire place        3
```

```r
print(lowest_price_row)
```

```
##   listing_id price  city                                  name    property_type
## 1   37675534    10 Paris Joli petit appartement bien situe Entire apartment
##      room_type bedrooms
## 1 Entire place        3
```

Air Quality

This dataset is from UC Irvine Machine Learning Repository. It contains gas multisensor device field responses from an Italian city. Certified analyzer gas concentrations and hourly response averages are recorded. The dataset includes 9358 hourly averaged answers from 5 metal oxide chemical sensors in an Air Quality Chemical Multisensor Device. The longest freely available recordings of on-field deployed air quality chemical sensor device responses were from March 2004 to February 2005. A co-located reference certified analyzer produced hourly averaged CO, Non Metanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx), and NO2 concentrations that were ground truth. According to De Vito et al., Sens. And Act. B, Vol. 129,2,2008 (citation required), cross-sensitivities and concept and sensor drifts affect sensor concentration estimation. Missing values are -200.

The data has 15 columns which are

- Date (dd/mm/yyyy)
- Time (hh.mm.ss)
- CO(GT): Hourly averaged CO concentration
- PT08.S1(CO): Hourly averaged tin oxide
- NMHC(GT): hourly averaged overall Non Metanic HydroCarbons concentration
- C6H6(GT): hourly averaged Benzene concentration
- PT08.S2(NMHC): (titania) hourly averaged sensor response
- NOx(GT): hourly averaged NOx concentration
- PT08.S3(NOx): (tungsten oxide) hourly averaged sensor response
- NO2(GT): hourly averaged NO2 concentration
- PT08.S4(NO2): (tungsten oxide) hourly averaged sensor response
- PT08.S5(O3): (indium oxide) hourly averaged sensor response

- T : temperature
- RH: relative humidity
- AH: absolute humidity

We start by reading the data into a dataframe called air_quality. Next we combine Date and Time columns into a single date_time column and subsequently drop Date and Time.

```
air_quality <- read.csv("AirQualityUCI.csv")
air_quality$date_time <- paste(air_quality$Date, air_quality$Time)
air_quality <- air_quality %>%
    select(-c(Date, Time))
head(air_quality)
```

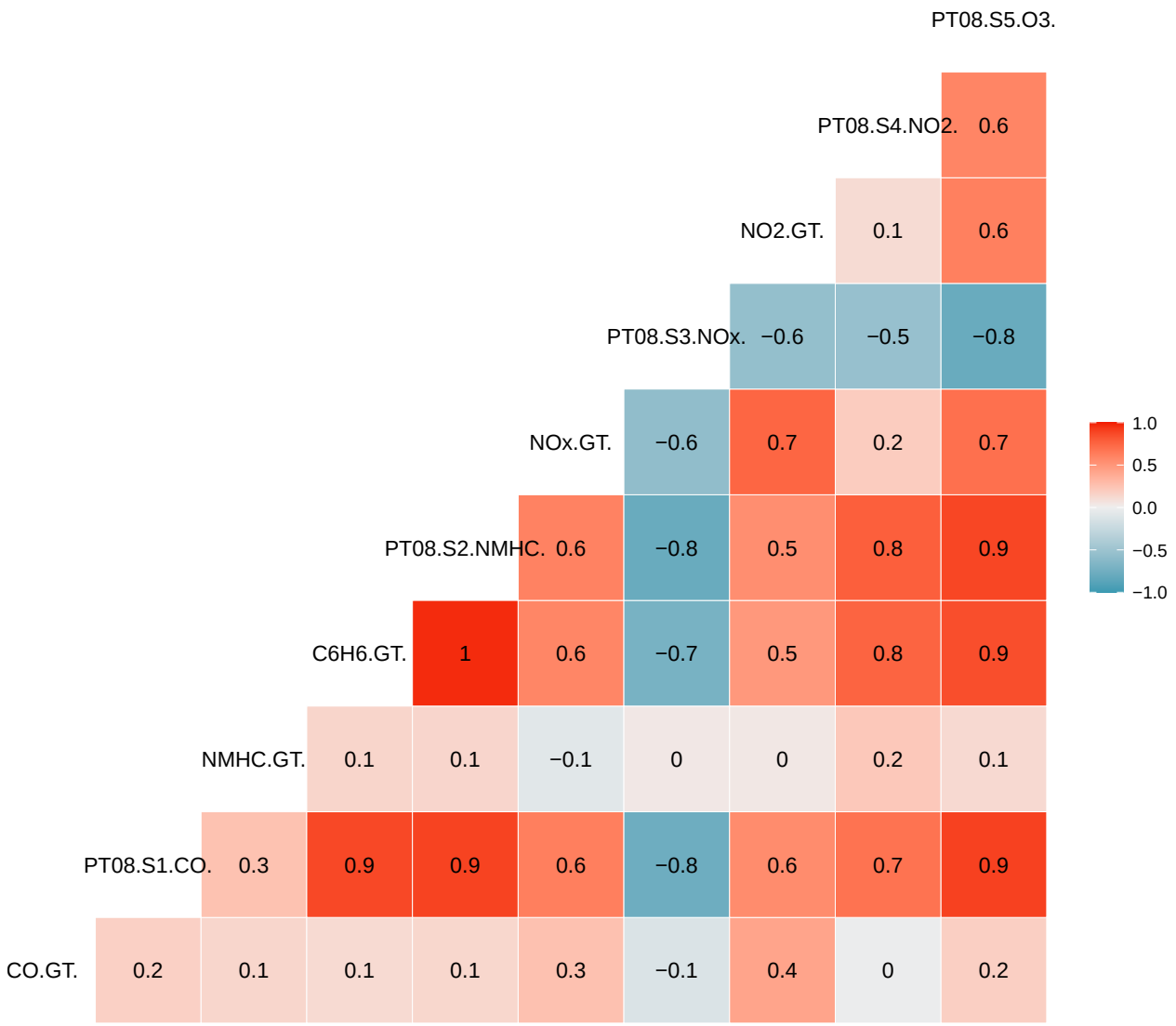
```
##   CO.GT. PT08.S1.CO. NMHC.GT. C6H6.GT. PT08.S2.NMHC. NOx.GT. PT08.S3.NOx.
## 1    2.6        1360      150     11.9          1046     166         1056
## 2    2.0        1292      112      9.4           955     103         1174
## 3    2.2        1402       88      9.0           939     131         1140
## 4    2.2        1376       80      9.2           948     172         1092
## 5    1.6        1272       51      6.5           836     131         1205
## 6    1.2        1197       38      4.7           750      89         1337
##   NO2.GT. PT08.S4.NO2. PT08.S5.O3.    T   RH     AH          date_time
## 1     113         1692        1268 13.6 48.9 0.7578 3/10/2004 18:00:00
## 2      92         1559         972 13.3 47.7 0.7255 3/10/2004 19:00:00
## 3     114         1555        1074 11.9 54.0 0.7502 3/10/2004 20:00:00
## 4     122         1584        1203 11.0 60.0 0.7867 3/10/2004 21:00:00
## 5     116         1490        1110 11.2 59.6 0.7888 3/10/2004 22:00:00
## 6      96         1393         949 11.2 59.2 0.7848 3/10/2004 23:00:00
```

In the data description it is noted that the values -200 denote missing values. We fill the missing values with the column mean.

```
# replace -200 values with column mean
air_quality <- air_quality %>%
    mutate_if(is.numeric, function(x) ifelse(x == -200, mean(x, na.rm = TRUE), x))
```

```
new_air_quality <- select(air_quality, -c(date_time, T, RH, AH))

ggcorr(new_air_quality, label = TRUE) +
    ggtitle("Correlation Matrix") +
    theme(plot.title = element_text(hjust = 0.5))
```




```
ggsave("corr_matrix3.png", width = 10, height = 8, dpi = 300)
```

Next, we scale our data

```
# Load the caret package
library(caret)

# Scale the air quality dataset
scaled_air_quality <- as.data.frame(scale(air_quality[, -which(names(air_quality) == "date_time")]))
```

As the final step we spplit our data in 85/15 split and choose Relative Humidity (RH) as the target variable and the rest as features and try to do a multivariate regression.

```
# Create a data partition for the RH column
set.seed(123) # for reproducibility
trainIndex <- createDataPartition(scaled_air_quality$RH, p = .85, list = FALSE, times = 1)
```

```r
train <- scaled_air_quality[trainIndex, ]
test <- scaled_air_quality[-trainIndex, ]

# Perform linear regression
model <- lm(RH ~ ., data = train)

# Print the summary of the model
summary(model)
```

```
##
## Call:
## lm(formula = RH ~ ., data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1872 -0.3158 -0.0485  0.2755  3.8760
##
## Coefficients:
##                Estimate Std. Error  t value Pr(>|t|)
## (Intercept)    0.002357   0.005000    0.471    0.637
## CO.GT.         0.029775   0.005616    5.302 1.18e-07 ***
## PT08.S1.CO.    0.134207   0.015098    8.889  < 2e-16 ***
## NMHC.GT.      -0.123246   0.006125  -20.123  < 2e-16 ***
## C6H6.GT.      -0.552960   0.032466  -17.032  < 2e-16 ***
## PT08.S2.NMHC. -0.553627   0.038720  -14.298  < 2e-16 ***
## NOx.GT.        0.363355   0.009486   38.303  < 2e-16 ***
## PT08.S3.NOx.  -0.288574   0.010705  -26.956  < 2e-16 ***
## NO2.GT.       -0.307532   0.009178  -33.507  < 2e-16 ***
## PT08.S4.NO2.   1.042321   0.012377   84.212  < 2e-16 ***
## PT08.S5.O3.    0.022613   0.014644    1.544    0.123
## T             -1.036477   0.008528 -121.540  < 2e-16 ***
## AH             0.461935   0.008276   55.818  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4459 on 7942 degrees of freedom
## Multiple R-squared:  0.8029, Adjusted R-squared:  0.8026
## F-statistic:  2696 on 12 and 7942 DF,  p-value: < 2.2e-16
```

And finally we test our model of test dateset.

```r
# predict the values of the response variable using the test set
predictions <- predict(model, newdata = test)

# calculate the performance metrics
mse <- mean((test$RH - predictions)^2)
rmse <- sqrt(mse)

# print the performance metrics
cat("MSE:", mse, "\n")
```

```
## MSE: 0.187805
```

```r
cat("RMSE:", rmse, "\n")
```

```
## RMSE: 0.4333647
```