```r
# Housing Dataset Assignment
# DSC 520
# Week 5
# Statistics for Data Science Assignment Week 5
# David Berberena
# 1/14/2024

# Assignment Start

# Upload readr library for file importing

library(readr)

# set working directory for smooth file importing

setwd("C:/Users/dbzda/Documents/School/DSC 520 Statistics for Data Science")

# Import the converted Housing CSV file to view its properties

housing <- read_csv("Housing.csv")
```

```
## Rows: 12865 Columns: 24
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (8): Sale Date, sale_warning, sitetype, addr_full, ctyname, postalctyn,...
## dbl (16): Sale Price, sale_reason, sale_instrument, zip5, lon, lat, building...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# Upload dplyr library for data manipulation

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
# Using the dplyr package, use the 6 different operations to analyze/transform
# the data - GroupBy, Summarize, Mutate, Filter, Select, and Arrange - Remember
# this isn't just modifying data, you are learning about your data also - so
# play around and start to understand your dataset in more detail

## Group_by() and Summarize() function will be used together in a pipe
```

```r
summary_grouped_data <- housing %>% group_by(year_built) %>%
summarize(AvgSalePrice = mean(`Sale Price`), AvgSqFtLot = mean(sq_ft_lot))
summary_grouped_data
```

```
## # A tibble: 109 x 3
##    year_built AvgSalePrice AvgSqFtLot
##         <dbl>        <dbl>      <dbl>
##  1       1900      394500.    305115.
##  2       1903      430000      85377
##  3       1905      620000      22237
##  4       1906      550000      37026
##  5       1909        1070     221284
##  6       1910      150000      13064
##  7       1912      619667.     83646
##  8       1913      457500      64810.
##  9       1914      835000     138085
## 10       1915      228150       5917
## # i 99 more rows
```

```r
## Mutate() function will be used to create new column that adds 500 to the
## square_feet_total_living column to account for a garage

mutated_data <- mutate(housing, square_feet_total_with_garage =
                       square_feet_total_living + 500)
mutated_data
```

```
## # A tibble: 12,865 x 25
##    `Sale Date` `Sale Price` sale_reason sale_instrument sale_warning sitetype
##    <chr>              <dbl>       <dbl>           <dbl> <chr>        <chr>
##  1 1/3/2006          698000           1               3 <NA>         R1
##  2 1/3/2006          649990           1               3 <NA>         R1
##  3 1/3/2006          572500           1               3 <NA>         R1
##  4 1/3/2006          420000           1               3 <NA>         R1
##  5 1/3/2006          369900           1               3 15           R1
##  6 1/3/2006          184667           1              15 18 51        R1
##  7 1/4/2006         1050000           1               3 <NA>         R1
##  8 1/4/2006          875000           1               3 <NA>         R1
##  9 1/4/2006          660000           1               3 <NA>         R1
## 10 1/4/2006          650000           1               3 <NA>         R1
## # i 12,855 more rows
## # i 19 more variables: addr_full <chr>, zip5 <dbl>, ctyname <chr>,
## #   postalctyn <chr>, lon <dbl>, lat <dbl>, building_grade <dbl>,
## #   square_feet_total_living <dbl>, bedrooms <dbl>, bath_full_count <dbl>,
## #   bath_half_count <dbl>, bath_3qtr_count <dbl>, year_built <dbl>,
## #   year_renovated <dbl>, current_zoning <chr>, sq_ft_lot <dbl>,
## #   prop_type <chr>, present_use <dbl>, square_feet_total_with_garage <dbl>
```

```r
## Filter() function will be used to filter rows where square_feet_total_living
## is higher than 6000

filtered_data <- filter(housing, square_feet_total_living > 6000)
filtered_data
```

```
## # A tibble: 90 x 24
##    `Sale Date` `Sale Price` sale_reason sale_instrument sale_warning sitetype
##    <chr>              <dbl>       <dbl>           <dbl> <chr>        <chr>
##  1 2/1/2006         1900000           1               3 15 52        R1
##  2 3/29/2006         200000           1               3 <NA>         R1
##  3 4/3/2006         1425000           1               3 40           R1
##  4 4/12/2006        1425000           1               3 41           R1
##  5 4/17/2006        2500000           1               3 <NA>         R1
##  6 5/8/2006          555000           1               3 <NA>         R1
##  7 6/8/2006         1968000           1               3 <NA>         R1
##  8 6/19/2006        2569000           1               3 <NA>         R1
##  9 6/20/2006         350000           1               3 49           R1
## 10 7/13/2006        1875000           1               3 <NA>         R1
## # i 80 more rows
## # i 18 more variables: addr_full <chr>, zip5 <dbl>, ctyname <chr>,
## #   postalctyn <chr>, lon <dbl>, lat <dbl>, building_grade <dbl>,
## #   square_feet_total_living <dbl>, bedrooms <dbl>, bath_full_count <dbl>,
## #   bath_half_count <dbl>, bath_3qtr_count <dbl>, year_built <dbl>,
## #   year_renovated <dbl>, current_zoning <chr>, sq_ft_lot <dbl>,
## #   prop_type <chr>, present_use <dbl>
```

```
## Select() function will be used on the filtered dataset to select sale price,
## square feet total living, and square foot lot only

selected_data <- select(filtered_data, "Sale Price", "square_feet_total_living",
                        "sq_ft_lot")
selected_data
```

```
## # A tibble: 90 x 3
##    `Sale Price` square_feet_total_living sq_ft_lot
##           <dbl>                   <dbl>     <dbl>
##  1      1900000                    6610     37017
##  2       200000                    6880    288367
##  3      1425000                    6050     38509
##  4      1425000                    6050     38509
##  5      2500000                    6310     36362
##  6       555000                    6380     15021
##  7      1968000                    6680    167270
##  8      2569000                    8090    176418
##  9       350000                    8490    118483
## 10      1875000                    6010     48787
## # i 80 more rows
```

```
## Arrange() function will be used on the selected dataset to arrange the
## square feet total living column in ascending order

arranged_data <- arrange(selected_data, square_feet_total_living)
arranged_data
```

```
## # A tibble: 90 x 3
##    `Sale Price` square_feet_total_living sq_ft_lot
##           <dbl>                   <dbl>     <dbl>
##  1      1875000                    6010     48787
```

```
##  2      1300000                    6010     48787
##  3      1675000                    6020     91476
##  4      1305615                    6030     97138
##  5      1425000                    6050     38509
##  6      1425000                    6050     38509
##  7      1818026                    6050     96703
##  8       900000                    6050     38509
##  9      1540000                    6070    103406
## 10       950000                    6110     25234
## # i 80 more rows
```

```
# Upload purrr package for data iteration

library(purrr)

# Using the purrr package - perform 2 functions on your dataset.
# You could use zip_n, keep, discard, compact, etc.

## map_dbl() function can be used to compute the means of all numeric columns
## and will return NA for those columns that are non-numeric

## This output cannot be stored within a variable as some of the output comes
## with warnings that return NA

housing %>% map_dbl(mean)
```

```
## Warning in mean.default(.x[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(.x[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(.x[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(.x[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(.x[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(.x[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(.x[[i]], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(.x[[i]], ...): argument is not numeric or logical:
## returning NA
```

```
##               Sale Date                 Sale Price                sale_reason
##                      NA               6.607377e+05               1.550019e+00
##          sale_instrument               sale_warning                   sitetype
```

```
##             3.677730e+00                       NA                       NA
##                addr_full                    zip5                  ctyname
##                       NA             9.805254e+04                       NA
##               postalctyn                      lon                      lat
##                       NA            -1.220792e+02             4.768358e+01
##          building_grade square_feet_total_living                 bedrooms
##             8.240420e+00             2.539506e+03             3.478663e+00
##          bath_full_count          bath_half_count          bath_3qtr_count
##             1.798445e+00             6.133696e-01             4.939759e-01
##               year_built          year_renovated          current_zoning
##             1.993003e+03             2.624431e+01                       NA
##                sq_ft_lot                prop_type             present_use
##             2.222857e+04                       NA             6.597746e+00
```

```
## discard_at() function will be used to discard columns whose length of
## characters is equal to a certain length (in this case the length is 8), which
## eliminates the sitetype and bedrooms columns

leftover_data <- housing %>% discard_at(~ nchar(.x) == 8)
leftover_data
```

```
## # A tibble: 12,865 x 22
##    `Sale Date` `Sale Price` sale_reason sale_instrument sale_warning addr_full
##    <chr>              <dbl>       <dbl>           <dbl> <chr>        <chr>
##  1 1/3/2006          698000           1               3 <NA>         17021 NE 1~
##  2 1/3/2006          649990           1               3 <NA>         11927 178T~
##  3 1/3/2006          572500           1               3 <NA>         13315 174T~
##  4 1/3/2006          420000           1               3 <NA>         3303 178TH~
##  5 1/3/2006          369900           1               3 15           16126 NE 1~
##  6 1/3/2006          184667           1              15 18 51        8101 229TH~
##  7 1/4/2006         1050000           1               3 <NA>         21634 NE 8~
##  8 1/4/2006          875000           1               3 <NA>         21404 NE 6~
##  9 1/4/2006          660000           1               3 <NA>         7525 238TH~
## 10 1/4/2006          650000           1               3 <NA>         17703 NE 2~
## # i 12,855 more rows
## # i 16 more variables: zip5 <dbl>, ctyname <chr>, postalctyn <chr>, lon <dbl>,
## #   lat <dbl>, building_grade <dbl>, square_feet_total_living <dbl>,
## #   bath_full_count <dbl>, bath_half_count <dbl>, bath_3qtr_count <dbl>,
## #   year_built <dbl>, year_renovated <dbl>, current_zoning <chr>,
## #   sq_ft_lot <dbl>, prop_type <chr>, present_use <dbl>
```

```
# Use the cbind and rbind function on your dataset

## To use the cbind() and rbind() functions, I will create an extremely small
## subset of the data using some of the previous principles

## cbind() function

cbind_and_rbind_data <- filter(arranged_data, square_feet_total_living < 6030)

add_column1 <- c(578, 632, 597)
add_column2 <- c(867, 943, 885)
```

```
cbind_data <- cbind(cbind_and_rbind_data, bedroom_sqft = add_column1,
                    kitchen_sqft = add_column2)
cbind_data
```

```
##   Sale Price square_feet_total_living sq_ft_lot bedroom_sqft kitchen_sqft
## 1    1875000                     6010     48787          578          867
## 2    1300000                     6010     48787          632          943
## 3    1675000                     6020     91476          597          885
```

```
## rbind() function

add_row1 = c(1674920, 6010, 90843)
add_row2 = c(1737180, 6000, 89619)

rbind_data <- rbind(cbind_and_rbind_data, add_row1, add_row2)
rbind_data
```

```
## # A tibble: 5 x 3
##   `Sale Price` square_feet_total_living sq_ft_lot
##          <dbl>                    <dbl>     <dbl>
## 1      1875000                     6010     48787
## 2      1300000                     6010     48787
## 3      1675000                     6020     91476
## 4      1674920                     6010     90843
## 5      1737180                     6000     89619
```

```
# Split a string, then concatenate the results back together

## To extract a character vector from the dataset, it must be filtered even more

short_filter <- head(filtered_data)
chr_vector <- short_filter$postalctyn

# Upload stringr library to perform string manipulation

library(stringr)

# String splitting

split_string <- strsplit(chr_vector, split = "")
split_string
```

```
## [[1]]
## [1] "R" "E" "D" "M" "O" "N" "D"
##
## [[2]]
## [1] "R" "E" "D" "M" "O" "N" "D"
##
## [[3]]
## [1] "R" "E" "D" "M" "O" "N" "D"
##
## [[4]]
```

```
## [1] "R" "E" "D" "M" "O" "N" "D"
##
## [[5]]
## [1] "R" "E" "D" "M" "O" "N" "D"
##
## [[6]]
## [1] "R" "E" "D" "M" "O" "N" "D"
```

```
# String concatenation

paste(split_string)
```

```
## [1] "c(\"R\", \"E\", \"D\", \"M\", \"O\", \"N\", \"D\")"
## [2] "c(\"R\", \"E\", \"D\", \"M\", \"O\", \"N\", \"D\")"
## [3] "c(\"R\", \"E\", \"D\", \"M\", \"O\", \"N\", \"D\")"
## [4] "c(\"R\", \"E\", \"D\", \"M\", \"O\", \"N\", \"D\")"
## [5] "c(\"R\", \"E\", \"D\", \"M\", \"O\", \"N\", \"D\")"
## [6] "c(\"R\", \"E\", \"D\", \"M\", \"O\", \"N\", \"D\")"
```