server.R

rstudio-user

2020-02-22

#  
# This is the server logic of a Shiny web application. You can run the  
# application by clicking 'Run App' above.  
#  
# Find out more about building applications with Shiny here:  
#  
# http://shiny.rstudio.com/  
#  
  
library(shiny)  
library(tidyverse)

## ── Attaching packages ────────────────────────────────────────────────────────────────────────────────────────────────────────────────── tidyverse 1.3.0 ──

## ✓ ggplot2 3.2.1 ✓ purrr 0.3.3  
## ✓ tibble 2.1.3 ✓ dplyr 0.8.4  
## ✓ tidyr 1.0.2 ✓ stringr 1.4.0  
## ✓ readr 1.3.1 ✓ forcats 0.4.0

## ── Conflicts ───────────────────────────────────────────────────────────────────────────────────────────────────────────────────── tidyverse\_conflicts() ──  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

library(maps)

##   
## Attaching package: 'maps'

## The following object is masked from 'package:purrr':  
##   
## map

library(scales)

##   
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':  
##   
## discard

## The following object is masked from 'package:readr':  
##   
## col\_factor

library(sf)

## Linking to GEOS 3.5.1, GDAL 2.2.2, PROJ 4.9.2

king <- read\_csv("data/KING COUNTY House Data.csv")

## Parsed with column specification:  
## cols(  
## .default = col\_double(),  
## date = col\_datetime(format = "")  
## )

## See spec(...) for full column specifications.

# Define server logic required to draw a histogram  
shinyServer(function(input, output, session) {  
   
 # Get County data  
 counties <- st\_as\_sf(map("county", plot = FALSE, fill = TRUE))  
 counties\_wa <-counties %>%  
 filter(str\_detect(ID, 'washington,')) # Filter Washinton state counties  
 counties\_wa\_king <- counties\_wa %>%  
 filter(str\_detect(ID, "king")) #Filter king county data  
 sites <- data.frame(longitude = c(-122.3321), latitude = c(47.6062))  
   
   
 average\_price\_data <- reactive({  
 floorsFilter <- input$floors # Floors Filter  
 waterfrontView <- input$waterfront #Water front filter  
 livingSqFeet <- input$livingSqFeet #Living area sq feet filter  
 gradeMin <- input$gradeRange[1] #Minimum grade from slider input  
 gradeMax <- input$gradeRange[2] #Maximum grade from slider input  
   
 #Set appropriate water front filter vector, based on the drop down input from ui  
 waterfrontFilter <- c(0,1)  
 if (waterfrontView == 1){  
 waterfrontFilter <- c(1)  
 }  
 else if (waterfrontView == 0){  
 waterfrontFilter <- c(0)  
 }  
 else if (waterfrontView == -1){  
 waterfrontFilter <- c(0,1)   
 }  
 # Number of bedrooms filter   
 numberOfBedRooms <- input$numberOfBedRooms  
   
 #Build the data for the plot averagePriceEachYear  
 averagePriceEachYear <- king %>%  
 filter(condition %in% input$condition & waterfront %in% waterfrontFilter) %>%  
 filter(waterfront %in% waterfrontFilter) %>%  
 filter(floors %in% floorsFilter) %>%  
 filter(bedrooms >= numberOfBedRooms) %>%  
 filter(sqft\_living >= livingSqFeet) %>%  
 filter (grade > gradeMin & grade < gradeMax) %>%  
 filter (price >= input$priceRange[1] & price <= input$priceRange[2]) %>%  
 group\_by(yr\_built) %>%  
 summarise(averagePrice = mean(price))  
 })  
  
 #Plot 1: Average price Year on Year  
 output$averagePricePlot <- renderPlot({  
 ggplot(data = average\_price\_data()) +  
 geom\_point(aes(x = yr\_built, y = averagePrice)) +  
 geom\_smooth(aes(yr\_built, averagePrice)) +  
 scale\_y\_continuous(labels = scales::dollar) +  
 labs(x = "Year", y = "Average Price") +  
 theme\_minimal()  
 })  
   
 houses\_data <- reactive({  
 waterfrontView <- input$waterfront #get the waterfront filter  
 floorsFilter <- input$floors #get floors filter  
   
 #Set appropriate water front filter vector, based on the drop down input from ui  
 waterfrontFilter <- c(0,1)  
 if (waterfrontView == 1){  
 waterfrontFilter <- c(1)  
 }  
 else if (waterfrontView == 0){  
 waterfrontFilter <- c(0)  
 }  
 else if (waterfrontView == -1){  
 waterfrontFilter <- c(0,1)   
 }  
 numberOfBedRooms <- input$numberOfBedRooms #Number of bedrooms filter  
 livingSqFeet <- input$livingSqFeet #Living Area Sq feet filter  
 gradeMin <- input$gradeRange[1] #Minimum grade from slider input  
 gradeMax <- input$gradeRange[2] #Maximum grade from slider input  
   
 # Apply filter to houses data for king county  
 king %>%   
 filter(condition %in% input$condition) %>%  
 filter(waterfront %in% waterfrontFilter) %>%  
 filter(floors %in% floorsFilter) %>%  
 filter(bedrooms >= numberOfBedRooms) %>%  
 filter(sqft\_living >= livingSqFeet) %>%  
 filter (grade > gradeMin & grade < gradeMax) %>%  
 filter (price >= input$priceRange[1] & price <= input$priceRange[2])  
 })  
   
 # Plot 2: Price by Geography  
 output$geopraphicPlot <- renderPlot({  
 counties\_wa\_king %>%  
 ggplot() +  
 geom\_sf() +  
 geom\_point(data = houses\_data(), aes(x = long, y = lat, color = price), alpha= .05) +  
 geom\_point(data = sites, aes(x = longitude, y = latitude), size = 4,  
 shape = 23, fill = "red") +  
 geom\_text(data = sites, aes(x = longitude, y = latitude), label = 'Seattle', position =  
 position\_dodge(width = 0.8), size = 3, vjust = -1.0) +  
 scale\_colour\_viridis\_c("Price", limits = c(input$range[1], input$range[2]), labels = scales::dollar) +  
 theme\_minimal() +  
 labs(x = "Longitude",  
 y = "Latitude")  
 })  
   
 # Plot 3: Price over time and geographic space  
 output$geopraphicPlotByTime <- renderPlot({  
 counties\_wa\_king %>%  
 ggplot() +  
 geom\_sf() +  
 geom\_point(data = houses\_data(), aes(x = long, y = lat, color = price ), alpha= .05) +  
 geom\_point(data = sites, aes(x = longitude, y = latitude), size = 2,  
 shape = 23, fill = "red") +  
 scale\_colour\_viridis\_c("Price", limits = c(input$range[1], input$range[2]), labels = dollar) +  
 facet\_wrap(~decade) +  
 theme(axis.text.x = element\_text(angle =50, hjust=0.75))+  
 labs(x = "Longitude",  
 y = "Latitude")  
 });  
   
 # Printing input variables for debugging purpose  
 observe({  
 print(" ================================================== ")  
 print(input$priceRange)  
 print(input$condition)  
 })  
  
})