Chapter-3-Figures-Code.R

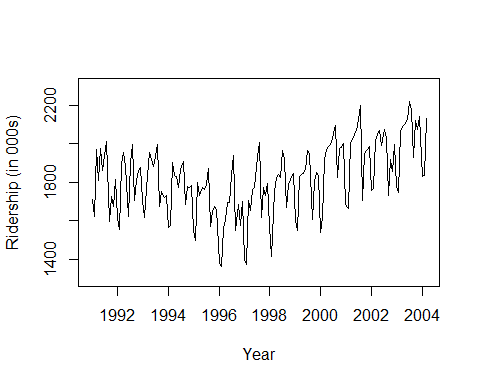
Stephanie

2025-03-31

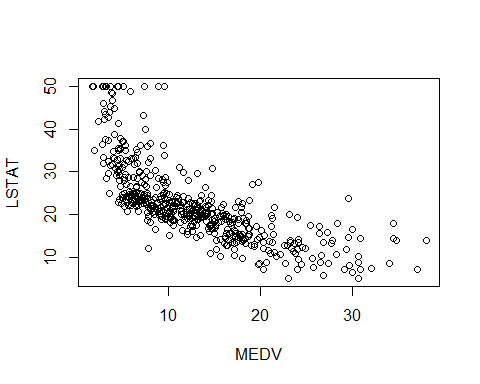
#### Chapter 3 Coding Exercises ####  
  
  
#### Figure 3.1: Basic Plots ####  
  
## line chart for the Amtrak data  
Amtrak.df <- read.csv("Amtrak.csv")  
  
# use time series analysis  
library(forecast)

## Registered S3 method overwritten by 'quantmod':  
## method from  
## as.zoo.data.frame zoo

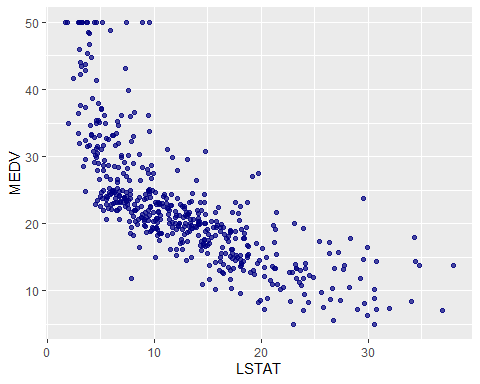
ridership.ts <- ts(Amtrak.df$Ridership, start = c(1991,1),   
 end = c(2004,3), freq = 12)  
plot(ridership.ts, xlab = "Year",   
 ylab = "Ridership (in 000s)", ylim = c(1300, 2300))



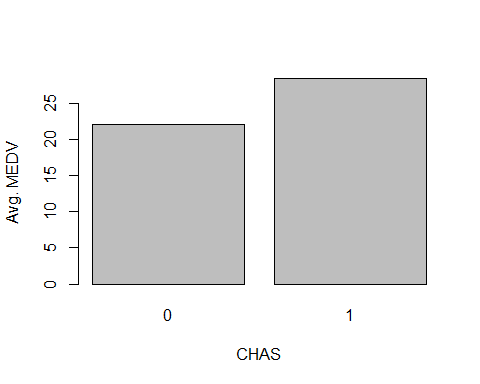
## Boston Housing Data  
housing.df <- read.csv("BostonHousing.csv")  
  
## scatter plot with axes names  
plot(housing.df$MEDV ~ housing.df$LSTAT,   
 xlab = "MEDV",  
 ylab = "LSTAT")



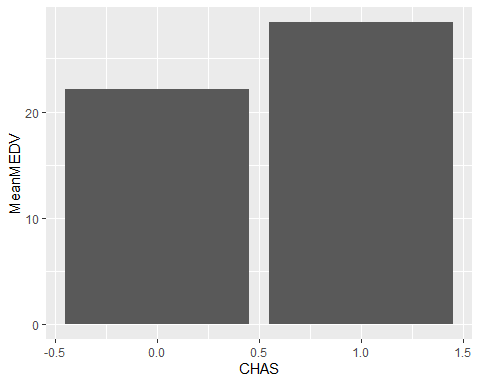
# alternative plot with ggplot  
library(ggplot2)  
ggplot(housing.df) +   
 geom\_point(aes(x = LSTAT, y = MEDV),  
 colour = "navy",  
 alpha = 0.7)



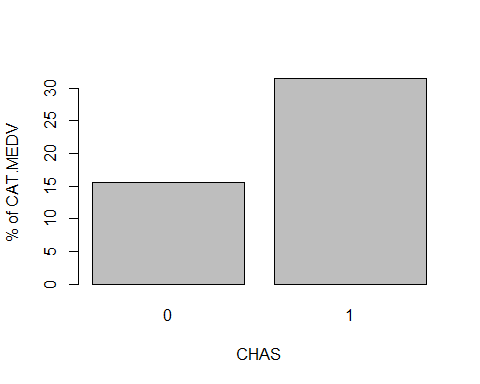
## barchart of CHAS vs. mean MEDV  
# compute mean MEDV per CHAS = (0, 1)  
data.for.plot <- aggregate(housing.df$MEDV,   
 by = list(housing.df$CHAS),   
 FUN = mean)  
names(data.for.plot) <- c("CHAS", "MeanMEDV")  
barplot(data.for.plot$MeanMEDV, names.arg = data.for.plot$CHAS,  
 xlab = "CHAS",   
 ylab = "Avg. MEDV")



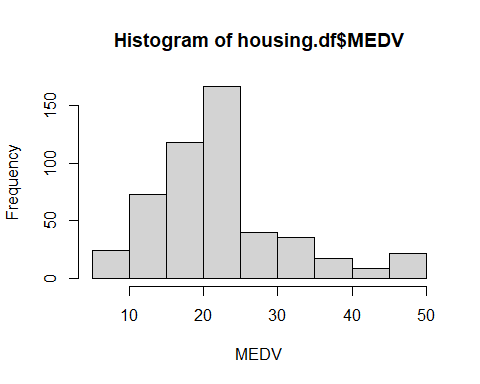
# alternative plot with ggplot  
ggplot(data.for.plot) +   
 geom\_bar(aes(x = CHAS, y = MeanMEDV),  
 stat = "identity")



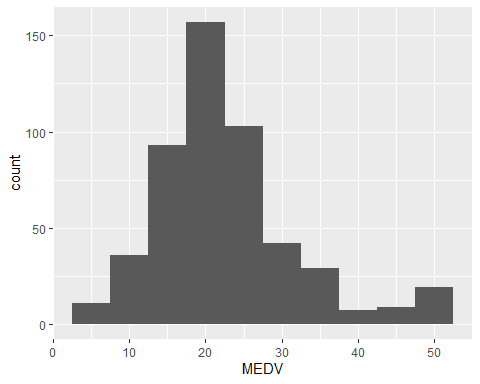
## barchart of CHAS vs. % CAT.MEDV  
data.for.plot <- aggregate(housing.df$CAT..MEDV,  
 by = list(housing.df$CHAS),   
 FUN = mean)  
names(data.for.plot) <- c("CHAS", "MeanCATMEDV")  
barplot(data.for.plot$MeanCATMEDV \* 100,   
 names.arg = data.for.plot$CHAS,  
 xlab = "CHAS",   
 ylab = "% of CAT.MEDV")



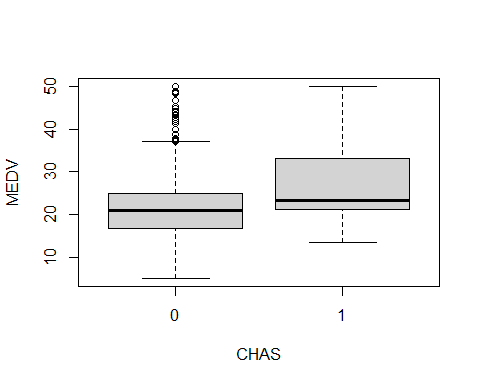
#### Figure 3.2: Distribution Charts for Numerical Variable MEDV. ####  
  
## histogram of MEDV  
hist(housing.df$MEDV, xlab = "MEDV")



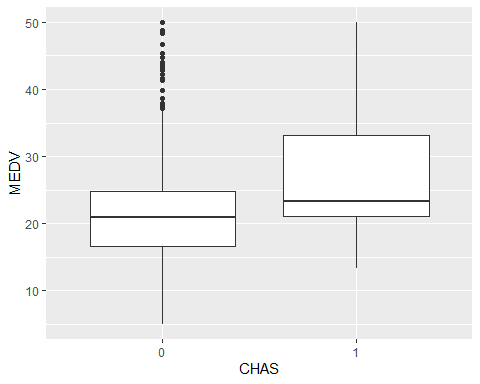
# alternative with ggplot  
library(ggplot2)  
ggplot(housing.df) +   
 geom\_histogram(aes(x = MEDV),   
 binwidth = 5)



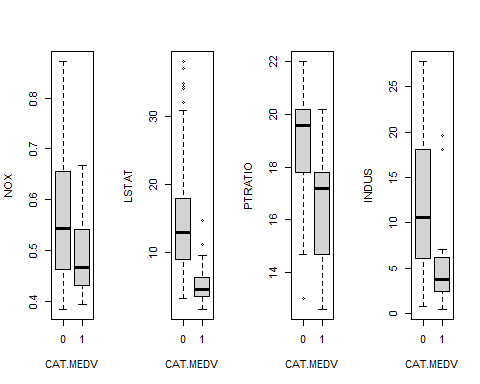
## boxplot of MEDV for different values of CHAS  
boxplot(housing.df$MEDV ~ housing.df$CHAS,  
 xlab = "CHAS",   
 ylab = "MEDV")



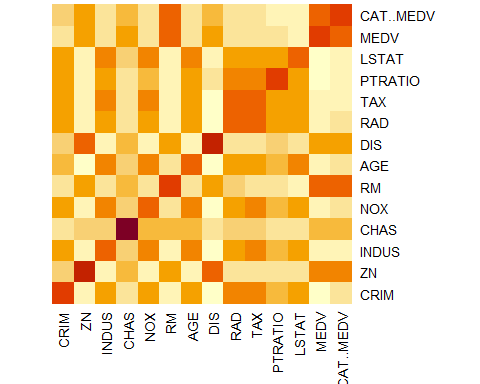
# alternative plot with ggplot  
ggplot(housing.df) +   
 geom\_boxplot(aes(x = as.factor(CHAS),  
 y = MEDV)) +   
 xlab("CHAS")



#### Figure 3.3: Side-by-Side Boxplots #####  
  
## side-by-side boxplots  
# use par() to split the plots into panels.  
par(mfcol = c(1,4))  
boxplot(housing.df$NOX ~ housing.df$CAT..MEDV,   
 xlab = "CAT.MEDV",  
 ylab = "NOX")  
boxplot(housing.df$LSTAT ~ housing.df$CAT..MEDV,  
 xlab = "CAT.MEDV",  
 ylab = "LSTAT")  
boxplot(housing.df$PTRATIO ~ housing.df$CAT..MEDV,  
 xlab = "CAT.MEDV",  
 ylab = "PTRATIO")  
boxplot(housing.df$INDUS ~ housing.df$CAT..MEDV,  
 xlab = "CAT.MEDV",  
 ylab = "INDUS")



#### Figure 3.4: Heatmap of a Correlation Table ####  
  
## simple heatmap of correlations (without values)  
heatmap(cor(housing.df), Rowv = NA, Colv = NA)

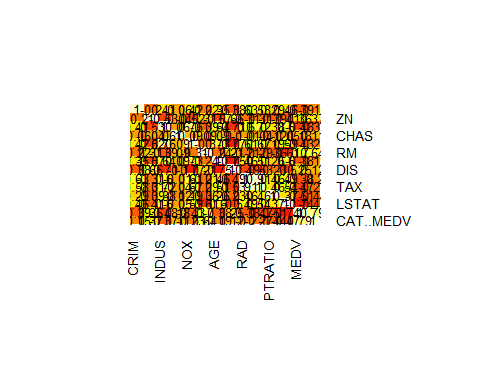


# must install gplots package  
#install.packages("gplots")  
  
## heatmap with values  
library(gplots)

##   
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':  
##   
## lowess

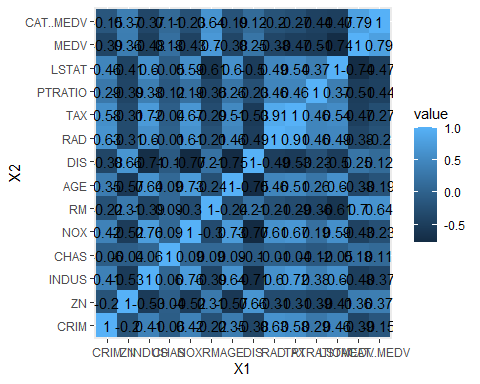
heatmap.2(cor(housing.df),   
 Rowv = FALSE, Colv = FALSE,   
 dendrogram = "none",  
 cellnote = round(cor(housing.df),2),  
 notecol = "black", key = FALSE,   
 trace = 'none', margins = c(10,10))



# alternative plot with ggplot  
library(ggplot2)  
# to generate input for the plot  
library(reshape)  
cor.mat <- round(cor(housing.df),2) # rounded correlation matrix  
melted.cor.mat <- melt(cor.mat)

## Warning in type.convert.default(X[[i]], ...): 'as.is' should be specified by  
## the caller; using TRUE  
## Warning in type.convert.default(X[[i]], ...): 'as.is' should be specified by  
## the caller; using TRUE

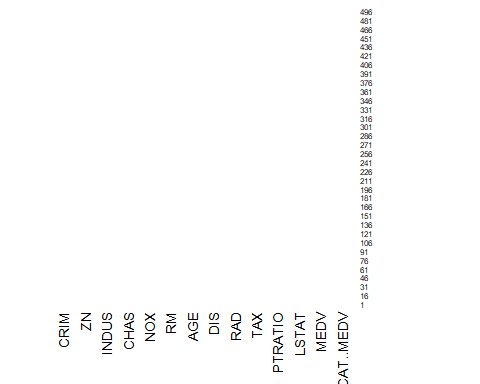
ggplot(melted.cor.mat, aes(x = X1, y = X2, fill = value)) +   
 geom\_tile() +   
 geom\_text(aes(x = X1, y = X2, label = value))



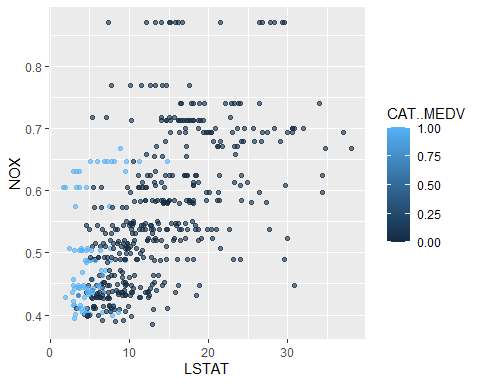
#### Figure 3.5: Heatmap of Missing Values ####  
  
# replace dataFrame with your data.  
# is.na() returns a Boolean (TRUE/FALSE) output indicating   
# the location of missing values.  
# multiplying the Boolean value by 1 converts the output into  
# binary (0/1).  
heatmap(1 \* is.na(housing.df), Rowv = NA, Colv = NA)

## Warning in min(x): no non-missing arguments to min; returning Inf

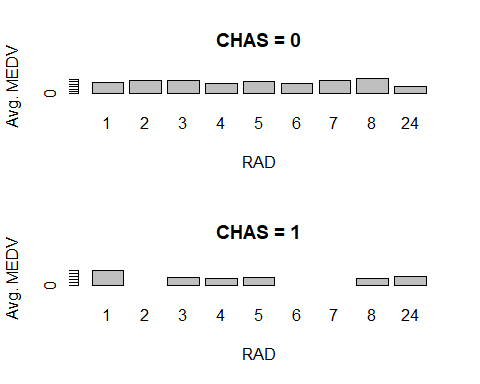
## Warning in max(x): no non-missing arguments to max; returning -Inf



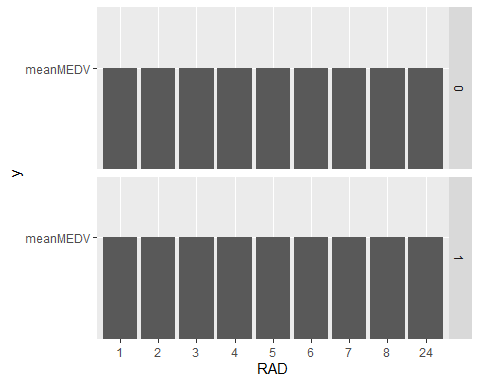
# I don't have other data so that will always be correct  
  
  
#### Figure 3.6: Adding Categorical Values ####  
  
## color plot  
par(xpd=TRUE) # allow legend to display outside of plot area  
plot(housing.df$NOX ~ housing.df$LSTAT, ylab = "NOX", xlab = "LSTAT",  
 col = ifelse(housing.df$CAT..MEDV == 1, "black", "gray"))  
# add legend outside of plotting area  
# In legend() use argument inset = to control the location of the   
# legend relative to the plot.  
legend("topleft", inset = c(0, -0.2),  
 legend = c("CAT.MEDV = 1", "CAT.MEDV = 0"),  
 col = c("black", "gray"),  
 pch = 1, cex = 0.5)  
  
# alternative plot with ggplot  
library(ggplot2)  
ggplot(housing.df, aes(y = NOX, x = LSTAT, colour = CAT..MEDV)) +  
 geom\_point(alpha = 0.6)  
  
## panel plots  
# compute mean MEDV per RAD and CHAS  
# In aggregate() use argument drop = FALSE to include all combinations  
# (exiting and missing) of RAD X CHAS.  
data.for.plot <- aggregate(housing.df$MEDV,   
 by = list(housing.df$RAD, housing.df$CHAS),  
 FUN = mean, drop = FALSE)  
names(data.for.plot) <- c("RAD", "CHAS", "meanMEDV")  
# plot the data  
par(mfcol = c(2,1))



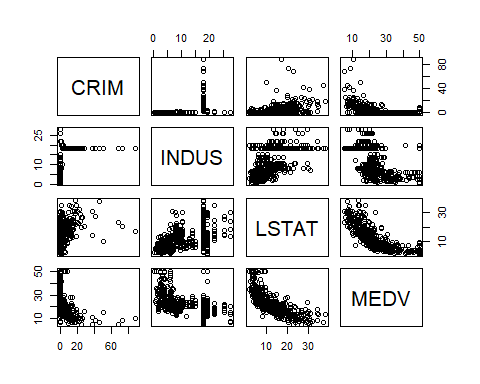
barplot(height = data.for.plot$meanMEDV[data.for.plot$CHAS == 0],  
 names.arg = data.for.plot$RAD[data.for.plot$CHAS == 0],  
 xlab = "RAD", ylab = "Avg. MEDV", main = "CHAS = 0")  
barplot(height = data.for.plot$meanMEDV[data.for.plot$CHAS == 1],  
 names.arg = data.for.plot$RAD[data.for.plot$CHAS == 1],  
 xlab = "RAD", ylab = "Avg. MEDV", main = "CHAS = 1")



# alternative plot with ggplot  
ggplot(data.for.plot) +   
 geom\_bar(aes(x = as.factor(RAD),  
 y = 'meanMEDV'),  
 stat = "identity") +   
 xlab("RAD") +   
 facet\_grid(CHAS ~ .)



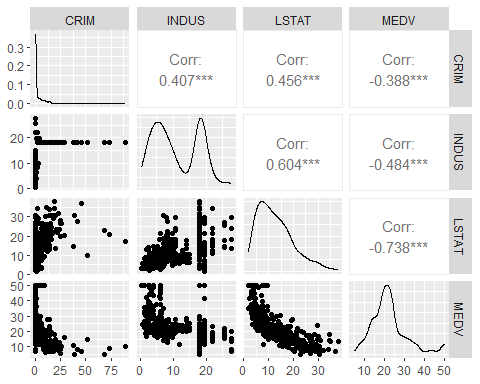
#### Figure 3.7: Scatter Plot Matrix ####  
  
## simple plot  
# use plot() to generate a matrix of 4X4 panels with  
# variable name on the diagonal, and scatter plots on  
# the remaining panels.  
plot(housing.df[, c(1, 3, 12, 13)])



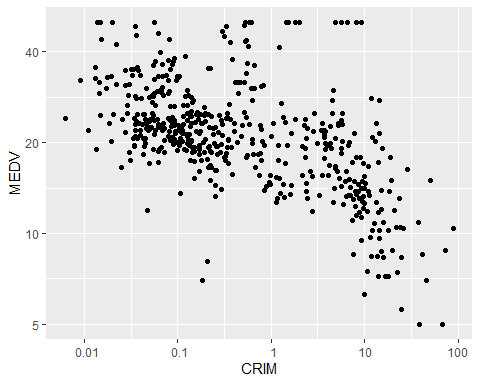
# alternative, nicer plot (displayed)  
# install.packages("GGally")  
library(GGally)

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

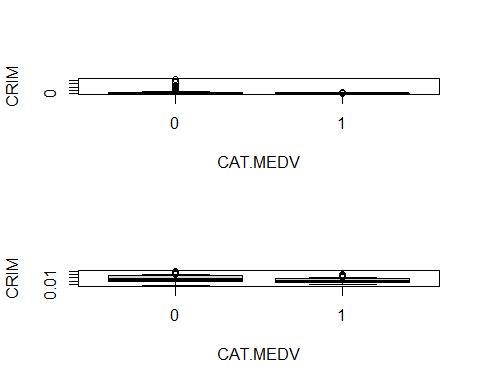
ggpairs(housing.df[, c(1, 3, 12, 13)])



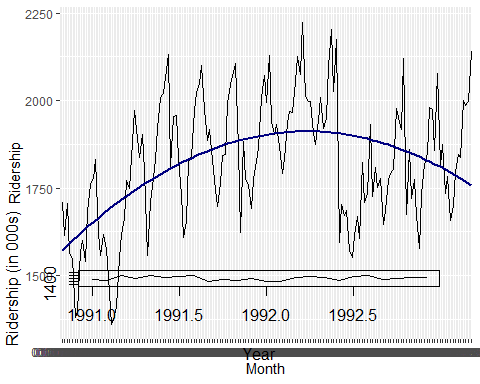
#### Figure 3.8: Rescaling and Patterns ####  
  
# avoid scientific notation  
options(scipen=999)  
  
## scatter plot: regular and log scale   
# must comment out to compile  
# plot(housing.df$MEDV ~ housing.df$CRIM, xlab = "CRIM", y = "MEDV")  
# to use logarithmic scale set argument log = to either 'x', 'y', or 'xy'.  
# plot(housing.df$MEDV ~ housing.df$CRIM,  
 # xlab = "CRIM", ylab = "MEDV", log = 'xy')  
  
# alternative log-scale plot with ggplot  
library(ggplot2)  
ggplot(housing.df) +   
 geom\_point(aes(x = CRIM, y = MEDV)) +   
 scale\_x\_log10(breaks = 10^(-2:2),  
 labels = format(10^(-2:2),  
 scientific = FALSE,  
 drop0trailing = TRUE)) +  
 scale\_y\_log10(breaks = c(5, 10, 20, 40))



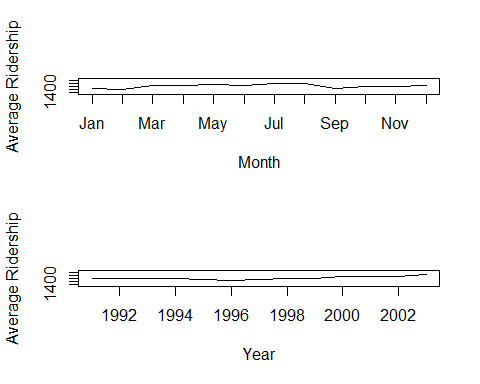
## boxplot: regular and log scale  
boxplot(housing.df$CRIM ~ housing.df$CAT..MEDV,  
 xlab = "CAT.MEDV", ylab = "CRIM")  
boxplot(housing.df$CRIM ~ housing.df$CAT..MEDV,  
 xlab = "CAT.MEDV", ylab = "CRIM", log = 'y')



#### Figure 3.9: Time Series Line Graphs ####  
  
library(forecast)  
Amtrak.df <- read.csv("Amtrak data.csv")  
ridership.ts <- ts(Amtrak.df$Ridership,  
 start = c(1991, 1),  
 end = c(2004,3),  
 freq = 12)  
  
## fit curve  
ridership.lm <- tslm(ridership.ts ~ trend + I(trend^2))  
plot(ridership.ts, xlab = "Year", ylab = "Ridership (in 000s)",  
 ylim = c(1300, 2300))  
lines(ridership.lm$fitted, lwd = 2)  
  
# alternative plot with ggplot  
library(ggplot2)  
ggplot(Amtrak.df, aes(y = Ridership, x = Month, group = 12)) +   
 geom\_line() + geom\_smooth(formula = y ~ poly(x, 2),   
 method = "lm", colour = "navy",  
 se = FALSE, na.rm = TRUE)  
  
## zoom in, monthly, and annual plots  
ridership.2yrs <- window(ridership.ts, start = c(1991,1),  
 end = c(1992,12))  
plot(ridership.2yrs, xlab = "Year", ylab = "Ridership (in 000s)",  
 ylim = c(1300, 2300))

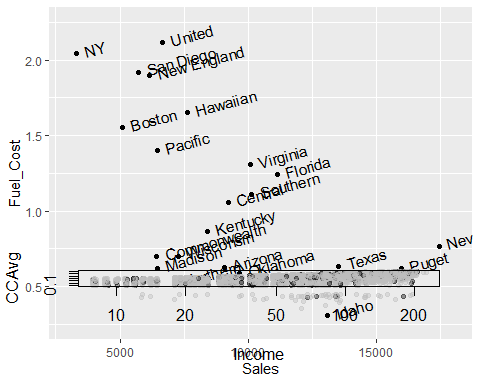


monthly.ridership.ts <- tapply(ridership.ts, cycle(ridership.ts), mean)  
plot(monthly.ridership.ts, xlab = "Month", ylab = "Average Ridership",  
 ylim = c(1300, 2300), type = "l", xaxt = 'n')  
## set x labels  
axis(1, at = c(1:12), labels = c("Jan", "Feb", "Mar",  
 "Apr", "May", "Jun",  
 "Jul", "Aug", "Sep",  
 "Oct", "Nov", "Dec"))  
annual.ridership.ts <- aggregate(ridership.ts, FUN = mean)  
plot(annual.ridership.ts, xlab = "Year",   
 ylab = "Average Ridership",  
 ylim = c(1300, 2300))



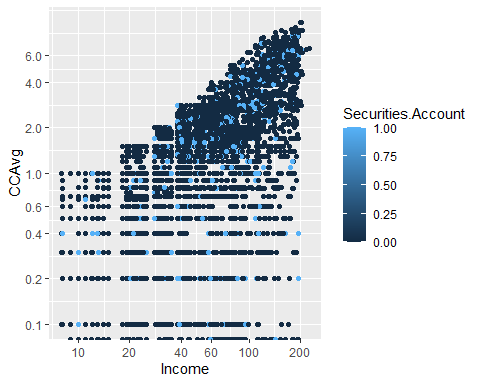
#### Figure 3.10: Scatter Plot with Labeled Points ####  
  
utilities.df <- read.csv("Utilities.csv")  
  
plot(utilities.df$Fuel\_Cost - utilities.df$Sales,  
 xlab = "Sales", ylab = "Fuel Cost",   
 xlim = c(2000, 20000))  
text(x = utilities.df$Sales, y = utilities.df$Fuel\_Cost,  
 labels = utilities.df$Company, pos = 4, cex = 0.8,  
 srt = 20, offset = 0.2)  
  
# alternative with ggplot  
library(ggplot2)  
ggplot(utilities.df, aes(y = Fuel\_Cost, x = Sales)) +   
 geom\_point() +   
 geom\_text(aes(label = paste(" ", Company)),  
 size = 4, hjust = 0.0, angle = 15) +   
 ylim(0.25, 2.25) + xlim(3000, 18000)  
  
#### Figure 3.11: Scatter Plot of Large Dataset ####  
  
# use function alpha() in library scales to   
# add transparent colors  
  
# load data first!  
universal.df <- read.csv("UniversalBank.csv")  
  
library(scales)  
plot(jitter(universal.df$CCAvg, 1) ~   
 jitter(universal.df$Income, 1),  
 col = alpha(ifelse(universal.df$Securities.Account == 0,  
 "gray", "black"), 0.4),  
 pch = 20, log = 'xy', ylim = c(0.1, 10),  
 xlab = "Income", ylab = "CCAvg")

## Warning in xy.coords(x, y, xlabel, ylabel, log): 49 y values <= 0 omitted from  
## logarithmic plot

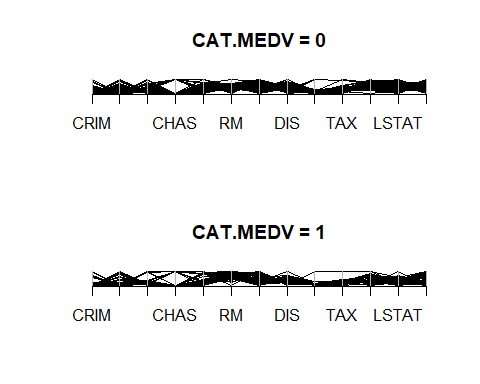


# alternative with ggplot  
library(ggplot2)  
ggplot(universal.df) +   
 geom\_jitter(aes(x = Income, y = CCAvg,   
 colour = Securities.Account)) +  
 scale\_x\_log10(breaks = c(10, 20, 40, 60, 100, 200)) +   
 scale\_y\_log10(breaks = c(0.1, 0.2, 0.4, 0.6, 1.0, 2.0, 4.0, 6.0))

## Warning in scale\_y\_log10(breaks = c(0.1, 0.2, 0.4, 0.6, 1, 2, 4, 6)): log-10  
## transformation introduced infinite values.



#### Figure 3.12: Parallel Coordinates Plot for Boston Housing Data ####  
  
# load Boston Housing Data  
housing.df <- read.csv("BostonHousing.csv")  
  
library(MASS)  
par(mfcol = c(2,1))  
parcoord(housing.df[housing.df$CAT..MEDV == 0, -14],  
 main = "CAT.MEDV = 0")  
parcoord(housing.df[housing.df$CAT..MEDV == 1, -14],   
 main = "CAT.MEDV = 1")



#### Figure 3.14: Network Graph of eBay Sellers ####  
  
## install packages first!  
# install.packages("igraph")  
  
library(igraph)

##   
## Attaching package: 'igraph'

## The following objects are masked from 'package:stats':  
##   
## decompose, spectrum

## The following object is masked from 'package:base':  
##   
## union

ebay.df <- read.csv("eBayNetwork.csv")  
  
# transform node IDs to factors  
ebay.df[,1] <- as.factor(ebay.df[,1])  
ebay.df[,2] <- as.factor(ebay.df[,2])  
  
graph.edges <- as.matrix(ebay.df[,1:2])  
g <- graph.edgelist(graph.edges, directed = FALSE)

## Warning: `graph.edgelist()` was deprecated in igraph 2.0.0.  
## ℹ Please use `graph\_from\_edgelist()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.

isBuyer <- V(g)$name %in% graph.edges[,2]  
  
plot(g, vertex.label = NA,   
 vertex.color = ifelse(isBuyer, "gray",   
 "black"),  
 vertex.size = ifelse(isBuyer, 7, 10))  
  
  
#### Figure 3.15: Treemap Showing Nearly 11,000 eBay Auctions ####  
  
# install package first!  
# install.packages("treemap")  
  
library(treemap)  
tree.df <- read.csv("EbayTreemap.csv")  
  
# add column for negative feedback  
tree.df$negative.feedback <- 1\* (tree.df$Seller.Feedback < 0)  
  
# draw treemap  
treemap(tree.df, index = c("Category", "Sub.Category", "Brand"),  
 vSize = "High.Bid", vColor = "negative.feedback",  
 fun.aggregate = "mean",  
 align.labels = list(c("left", "top"),  
 c("right", "bottom"),  
 c("center", "center")),  
 palette = rev(gray.colors(3)), type ="manual",  
 title = "")  
  
  
#### Figure 3.16: Map Chart on a Google Map ####  
  
## install packages first!  
# install.packages("ggmap")  
  
library(ggmap)

## ℹ Google's Terms of Service: <https://mapsplatform.google.com>  
## Stadia Maps' Terms of Service: <https://stadiamaps.com/terms-of-service/>  
## OpenStreetMap's Tile Usage Policy: <https://operations.osmfoundation.org/policies/tiles/>  
## ℹ Please cite ggmap if you use it! Use `citation("ggmap")` for details.

# registering Google API to make this work  
register\_google(key = "AIzaSyBS2NXqjSruf6Bm5ERVEYILoHULXa9RFPo")  
  
# continued code from textbook  
SCstudents <- read.csv("SC-US-students-GPS-data-2016.csv")  
Map <- get\_map("Denver, CO", zoom = 3)

## ℹ <https://maps.googleapis.com/maps/api/staticmap?center=Denver,%20CO&zoom=3&size=640x640&scale=2&maptype=terrain&language=en-EN&key=xxx>

## ℹ <https://maps.googleapis.com/maps/api/geocode/json?address=Denver,+CO&key=xxx>

ggmap(Map) +   
 geom\_point(aes(x = longitude,  
 y = latitude),  
 data = SCstudents,  
 alpha = 0.4,   
 colour = "red",  
 size = 0.5)  
  
# error due to needing API key for Google  
  
  
#### Figure 3.17: World Maps Comparing "Well-Being" to GDP ####  
# install packages first!  
# install.packages("mosaic")  
  
# library(mosaic)  
  
# gdp.df <- read.csv("gdp.csv",   
 # skip = 4,  
 # stringsAsFactors = FALSE)  
# names(gdp.df)[5] <- "GDP2015"  
# happiness.df <- read.csv("Veerhoven.csv")  
  
# checking header first and then keep troubleshooting  
# head(gdp.df)  
  
## trying to standardize country names and country codes  
# installing countrycode package  
# install.packages("countrycode")  
  
# library(countrycode)  
# gdp.df$Country.Standard <- countrycode(gdp.df$Country.Code,  
 # origin = 'iso3c',  
 # destination = 'country.name')  
  
# Based on the error, I found many countries I need to check information for  
# unique\_country\_codes <- unique(gdp.df$Country.Code)  
# print(sort(unique\_country\_codes))  
  
# Excluding non-countries so I can graph the countries  
# exclude\_codes <- c("ARB", "CEB", "CHI", "CSS", "EAP", "EAR", "EAS",  
 # "ECA", "ECS", "EUU", "FCS", "HIC", "HPC", "IBD",   
 # "IBT", "IDA", "IDB", "IDX", "INX", "KSV", "LAC",  
 # "LCN", "LDC", "LIC", "LMC", "LMY", "LTE", "MEA",  
 # "MIC", "MNA", "NAC", "OED", "OSS", "PRE", "PSS",   
 # "PST", "SAS", "SSA", "SSF", "SST", "TEA", "TEC",  
 # "TLA", "TMN", "TSA", "TSS", "UMC", "WLD", "EMU")  
  
# gdp\_countries\_only <- gdp.df[!gdp.df$Country.Code %in% exclude\_codes, ]  
  
# Trying countrycode conversion on filtered data  
# gdp\_countries\_only$Country.Standard <- countrycode(gdp\_countries\_only$Country.Code,  
 # origin = 'iso3c',  
 # destination = 'country.name')  
  
# gdp map  
# mWorldMap(gdp\_countries\_only,   
 # key = "Country.Standard",  
 # fill = "GDP2015") +   
 # coord\_map()  
  
# Figuring out the 19 items not translated  
# unique\_remaining\_codes <- unique(gdp\_countries\_only$Country.Code)  
# print(sort(unique\_remaining\_codes))  
  
# unmatched\_codes\_gdp <- gdp\_countries\_only$Country.Code[is.na(  
 # gdp\_countries\_only$Country.Standard  
# )]  
# print(sort(unique(unmatched\_codes\_gdp)))  
  
# gdp map  
# mWorldMap(gdp\_countries\_only,  
 # key = "Country.Standard",  
 # fill = "GDP2015") +   
 # coord\_map()  
  
# still finding the 19 missing  
# summary(gdp\_countries\_only$Country.Standard)  
  
# examining unique standardized country names  
# unique\_standard\_names <- unique(gdp\_countries\_only$Country.Standard)  
# print(sort(unique\_standard\_names))  
  
# ?mWorldMap  
  
# Trying to figure out the names  
# mergedData <- mWorldMap(gdp\_countries\_only,   
 # key = "Country.Standard",  
 # plot = "none")  
# head(mergedData)  
  
# GDP map - Attempt 2, using 'Country.Name' as the key  
# mWorldMap(gdp\_countries\_only,  
 # key = 'Country.Name',  
 # fill = "GDP2015") +   
 # coord\_map()  
  
# unique\_gdp\_names <- unique(gdp\_countries\_only$Country.Name)  
# print(sort(unique\_gdp\_names))  
  
## Testing by Changing Bahamas's Name  
# library(mosaic)  
  
# gdp.df <- read.csv("gdp.csv",  
 # skip = 4,  
 # stringsAsFactors = FALSE)  
# names(gdp.df)[5] <- "GDP2015"  
  
# Excluding Non-Countries  
# exclude\_codes <- c("ARB", "CEB", "EAR", "EAS", "EAP", "ECS", "EMU",  
 # "EUU", "FCS", "HIC", "HPC", "IBD", "IBT", "IDA",  
 # "IDB", "IDX", "INX", "LAC", "LCN", "LDC",   
 # "LIC", "LMC", "LMY", "LTE", "MEA", "MIC", "MNA",   
 # "NAC", "OED", "OSS", "PRE", "PSS", "PST", "SAS",   
 # "SSA", "SSF", "TEA", "TEC", "TLA", "TMN", "TSA",   
 # "TSS", "UMC", "WLD")  
# gdp\_countries\_only <- gdp.df[!gdp.df$Country.Code %in% exclude\_codes, ]  
  
# Remapping "Bahamas, The" to "Bahamas"  
# gdp\_countries\_only$Country.Name[gdp\_countries\_only$Country.Name == "Bahamas, The"] <- "Bahamas"  
  
# gdp map  
# mWorldMap(gdp\_countries\_only,  
 # key = "Country.Name",  
 # fill = "GDP2015") +   
 # coord\_map()  
  
# head(gdp\_countries\_only)  
# str(gdp\_countries\_only)  
  
# Remapping "Cabo Verde" to "Cape Verde"  
# gdp\_countries\_only$Country.Name[gdp\_countries\_only$Country.Name == "Cabo Verde"] <- "Cape Verde"  
  
# Testing gdp map  
# mWorldMap(gdp\_countries\_only,  
 # key = "Country.Name",  
 # fill = "GDP2015") +   
 # coord\_map()  
# Test successful --> down to 8  
  
# Remapping "Korea, Dem. Peoples Rep." to "North Korea"  
# gdp\_countries\_only$Country.Name[gdp\_countries\_only$Country.Name == "Korea, Dem. Peoples Rep."] <- "North Korea"  
  
# Testing gdp map  
# mWorldMap(gdp\_countries\_only,  
 # key = "Country.Name",  
 # fill = "GDP2015") +   
 # coord\_map()  
# Test successful --> down to 7  
  
# Remapping "West Bank and Gaza" to "Palestine" --> Successful  
# gdp\_countries\_only$Country.Name[gdp\_countries\_only$Country.Name == "West Bank and Gaza"] <- "Palestine"  
  
# Testing gdp map  
# mWorldMap(gdp\_countries\_only,  
 # key = "Country.Name",  
 # fill = "GDP2015") +   
 # coord\_map()  
# Test successful --> down to 6  
  
# Remapping "St. Martin (French part)" to "Saint Martin" --> Successful  
# gdp\_countries\_only$Country.Name[gdp\_countries\_only$Country.Name == "St. Martin (French part)"] <- "Saint Martin"  
  
# Testing gdp map  
# mWorldMap(gdp\_countries\_only,  
 # key = "Country.Name",  
 # fill = "GDP2015") +   
 # coord\_map()  
# Test successful --> down to 5  
  
# Did not test beyond this as some countries might be islands.. etc.  
  
# Well-Being Map  
# must comment out due to not working and to compile the file  
# happiness.df <- read.csv("Veerhoven.csv")  
  
# mWorldMap(happiness.df,  
 # key = "Nation",  
 # fill = happiness.df$Score) +   
 # coord\_map() +   
 # scale\_fill\_continuous(name = "Happiness")  
  
# Figuring out plotting errors  
# happiness.df <- read.csv("Veerhoven.csv")  
# unique\_nations <- unique(happiness.df$Nation)  
# print(unique\_nations)  
  
# Remapping Test 1: "Bosina Herzegovina" --> "Bosnia and Herzegovina" -> Success  
# happiness.df$Nation[happiness.df$Nation == "Bosnia Herzegovina"] <- "Bosnia and Herzegovina"  
  
# Testing for new country  
# mWorldMap(happiness.df,  
 # key = "Nation",   
 # fill = happiness.df$Score) +   
 # coord\_map() +   
 # scale\_fill\_continuous(name = "Happiness")  
  
# Remapping Test 2: "Central African Rep" to "Central African Republic" -> Success  
# happiness.df$Nation[happiness.df$Nation == "Central African Rep"] <- "Central African Republic"  
  
# Testing for new country  
# mWorldMap(happiness.df,  
 # key = "Nation",  
 # fill = happiness.df$Score) +   
 # coord\_map() +   
 # scale\_fill\_continuous(name = "Happiness")  
  
# Remapping "Cote dIvoire" to "Côte d'Ivoire" -> Successful  
# happiness.df$Nation[happiness.df$Nation == "Cote dIvoire"] <- "Cote d'Ivoire"  
  
# Testing for new country  
# mWorldMap(happiness.df,  
 # key = "Nation",  
 # fill = happiness.df$Score) +   
 # coord\_map() +   
 # scale\_fill\_continuous(name = "Happiness")  
  
  
# Remapping "Korea (South)" to "South Korea"  
# happiness.df$Nation[happiness.df$Nation == "Korea (South)"] <- "South Korea"  
  
# Testing for new country  
# mWorldMap(happiness.df,  
 # key = "Nation",  
 # fill = happiness.df$Score) +   
 # coord\_map() +   
 # scale\_fill\_continuous(name = "Happiness")  
  
# Remapping "North Cyprus" to "Cyprus"  
# happiness.df$Nation[happiness.df$Nation == "North Cyprus"] <- "Cyprus"  
  
# Testing for new country  
# mWorldMap(happiness.df,  
 # key = "Nation",  
 # fill = happiness.df$Score) +   
 # coord\_map() +   
 # scale\_fill\_continuous(name = "Happiness")  
  
# Remapping "Palestina" to "Palestine"  
# happiness.df$Nation[happiness.df$Nation == "Palestina"] <- "Palestine"  
  
# Testing for new country  
# mWorldMap(happiness.df,  
 # key = "Nation",  
 # fill = happiness.df$Score) +   
 # coord\_map() +   
 # scale\_fill\_continuous(name = "Happiness")  
  
# Remapping "United Arab Emirate" to "United Arab Emirates"  
# happiness.df$Nation[happiness.df$Nation == "United Arab Emirate"] <- "United Arab Emirates"  
  
# Testing for new country  
# mWorldMap(happiness.df,  
 # key = "Nation",  
 # fill = happiness.df$Score) +   
 # coord\_map() +   
 # scale\_fill\_continuous(name = "Happiness")  
   
# Remapping "Viet Nam" to "Vietnam"  
# happiness.df$Nation[happiness.df$Nation == "Viet Nam"] <- "Vietnam"  
  
# Testing for new country  
# mWorldMap(happiness.df,  
 # key = "Nation",  
 # fill = happiness.df$Score) +   
 # coord\_map() +   
 # scale\_fill\_continuous(name = "Happiness")  
  
# Ensuring that Happiness is numeric  
# happiness.df$Score <- as.numeric(as.character(happiness.df$Score))

