CS 5720: Data Visualization

February 8, 2018

Chapter 1

./visuals/fig_1_47

1.1 Description

```
# Cars scatter plot

"Interactive Data Visualization: Foundations, Techniques, and Applications, Second Edition"
by M. Ward, G. Grinstein, and D. Kim.
Figure 1.47 in on page 45

Original data retrieved here:
[http://www.idvbook.com/teaching-aid/data-sets/2004-cars-and-trucks-data/]

Data converted to CSV with column headers available in this repository:
[../../data/cars04.csv]
```

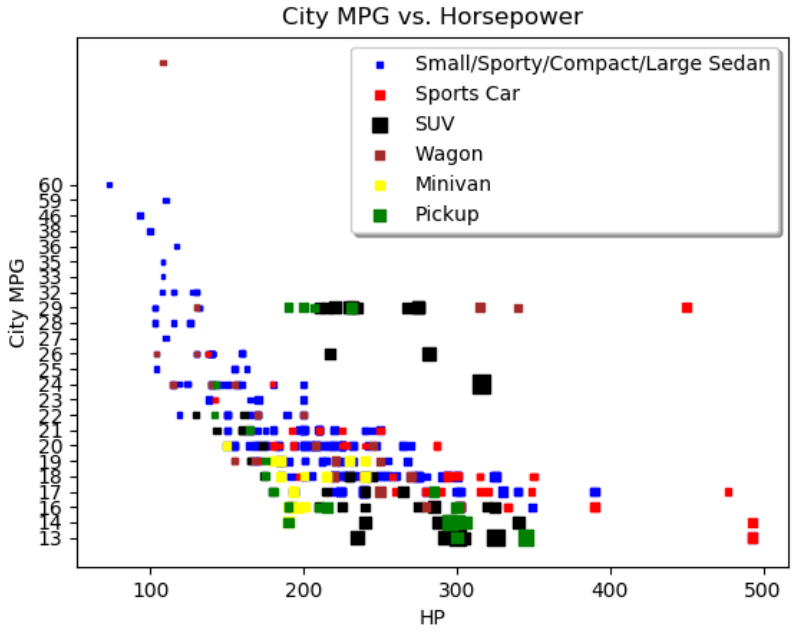
1.2 ./visuals/fig_1_47/caldwellc1

1.2.1 ./visuals/fig_1_47/caldwellc1/fig_1_47.py

```
import matplotlib.pyplot as plt
import sys
import pandas as pd
import numpy as np

# city MPG vs Horsepower
# color for vehicle type
# size for weight (area proportional to weight


def main():
    #car_data = pd.read_csv('cars04.csv')
    car_data = pd.read_csv(sys.argv[1])
    car_data = car_data.reset_index()
    car_data = car_data[['Small/Sporty/ Compact/Large Sedan','Sports
↪   Car','SUV','Wagon', 'Minivan','Pickup','HP','City MPG','Weight']]
    car_data = car_data.rename(columns=lambda x:x.strip().replace('
↪   ','_'))
    car_data = car_data.rename(columns=lambda
↪   x:x.strip().replace('/','_'))
    car_data = car_data.replace('*', np.nan)
    car_data = car_data.dropna(subset=['HP'])
    car_data = car_data.dropna(subset=['City_MPG'])
    car_data = car_data.dropna(subset=['Weight'])
    car_data = car_data.reset_index()

    small =
↪   car_data.drop(car_data[car_data.Small_Sporty__Compact_Large_Sedan <
↪   1].index)
    small = small.reset_index()
    sport = car_data.drop(car_data[car_data.Sports_Car < 1].index)
    sport = sport.reset_index()
    suv = car_data.drop(car_data[car_data.SUV < 1].index)
    suv = suv.reset_index()
    wagon = car_data.drop(car_data[car_data.Wagon < 1].index)
    wagon = wagon.reset_index()
    minivan = car_data.drop(car_data[car_data.Minivan < 1].index)
    minivan = minivan.reset_index()
    pick = car_data.drop(car_data[car_data.Pickup < 1].index)
    pick = pick.reset_index()
    fig, ax = plt.subplots()
    ax.scatter(small['HP'], small['City_MPG'], s=[2**(float(n)/1000) for
↪   n in small['Weight']], c='blue', marker='s',
↪   label='Small/Sporty/Compact/Large Sedan')
    ax.scatter(sport['HP'], sport['City_MPG'], s=[2**(float(n)/1000) for
↪   n in sport['Weight']], c='red', marker='s', label='Sports Car')
```

```
    ax.scatter(suv['HP'], suv['City_MPG'], s=[2**(float(n)/1000) for n in
↪   suv['Weight']], c='black', marker='s', label='SUV')
    ax.scatter(wagon['HP'], wagon['City_MPG'], s=[2**(float(n)/1000) for
↪   n in wagon['Weight']], c='brown', marker='s', label='Wagon')
    ax.scatter(minivan['HP'], minivan['City_MPG'], s=[2**(float(n)/1000)
↪   for n in minivan['Weight']], c='yellow', marker='s',
↪   label='Minivan')
    ax.scatter(pick['HP'], pick['City_MPG'], s=[2**(float(n)/1000) for n
↪   in pick['Weight']], c='green', marker='s', label='Pickup')

    ax.set_xlabel('HP')
    ax.set_ylabel('City MPG')
    ax.set_title('City MPG vs. Horsepower')
    ax.legend(loc='upper right', shadow=True, markerscale=1)

    #plt.show()
    print(sys.argv[2])
    plt.savefig(sys.argv[2])


if __name__ == '__main__':
    main()
```

1.2.2 ./visuals/fig_1_47/caldwellc1/fig_1_47.png



1.3 ./visuals/fig_1_47/campellcl

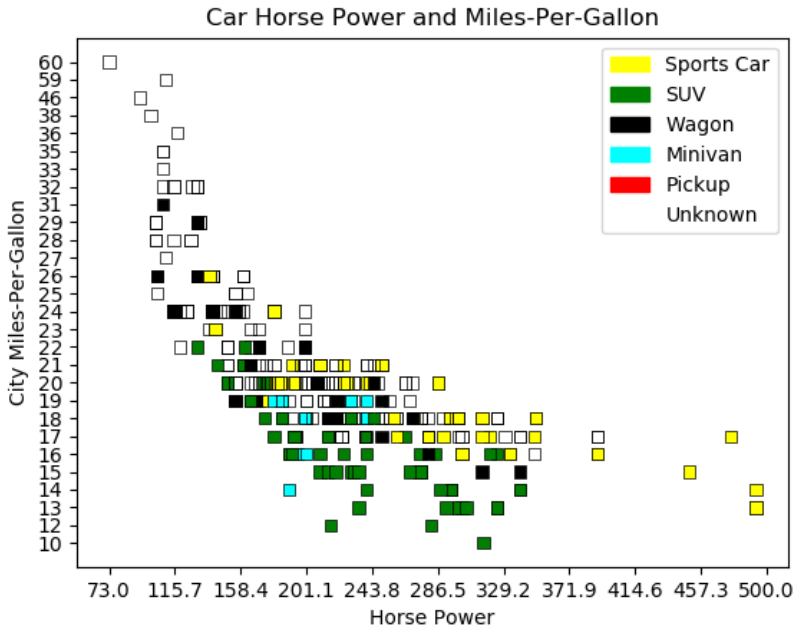
1.3.1 ./visuals/fig_1_47/campellcl/VisualizationOne.py

```
"""
VisualizationOne.py
Implementation of Programming Assignment One for CS5720.
"""

__author__ = "Chris Campell"
__version__ = "1/25/2018"

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import matplotlib.patches as mpatches
from sklearn.preprocessing import normalize
from matplotlib import cm
import sys

# Load file:
with open(sys.argv[1], 'r') as fp:
    data = pd.read_csv(fp, header=0)

# Convert to dataframe:
df_cars = pd.DataFrame(data=data)

# Is there missing data?
df_cars.__str__().__contains__('*')

# Remove extraneous columns:
# Notice that 'Vehicle Name' is included because Figure 1.47 is only
↪  Toyotas
df_cars = df_cars[['Vehicle Name', 'HP', 'City MPG', 'Len', 'Width',
↪  'Weight', 'Sports Car', 'SUV', 'Wagon', 'Minivan', 'Pickup']]

# Remove records with an unknown HP, City MPG, Len, or Width:
df_cars = df_cars.replace(r'[*]', np.nan, regex=True)
df_cars = df_cars.dropna(axis=0, how='any')

# Add in column with vehicle area:
df_cars['Area'] = [int(l)*int(w) for l,w in zip(df_cars['Len'],
↪  df_cars['Width'])]

# Ensure all nan's have been dropped from 'HP':
# df = df[np.isfinite(df['HP'])]

# Filter by Toyota vehicles:
toyota_only = df_cars[df_cars['Vehicle Name'].str.contains('Toyota')]
```

```
toyota_hp_vs_mpg = toyota_only[['Vehicle Name', 'HP', 'City MPG', 'Area',
↪  'Weight']]

# Create the scatter plot:
# Reference URL:
↪  https://stackoverflow.com/questions/17682216/scatter-plot-and-color-mapping-in-python
#
↪  https://stackoverflow.com/questions/4143502/how-to-do-a-scatter-plot-with-empty-circles-in-python
#
↪  http://nbviewer.jupyter.org/github/jvns/pandas-cookbook/blob/v0.1/cookbook/Chapter%207%20-%20Clea
x = df_cars['HP']
y = df_cars['City MPG']
fig, ax = plt.subplots()
# Color based on vehicle type:
#
↪  https://stackoverflow.com/questions/26139423/plot-different-color-for-different-categorical-level

def map_color_to_vehicle_type(df_row):
    if int(df_row['Sports Car']) == 1:
        color = 'Yellow'
    elif int(df_row['SUV']) == 1:
        color = 'Green'
    elif int(df_row['Wagon']) == 1:
        color = 'Black'
    elif int(df_row['Minivan']) == 1:
        color = 'Cyan'
    elif int(df_row['Pickup']) == 1:
        color = 'Red'
    else:
        # print("Vehicle type not identified")
        color = 'None'
    return color

def map_vehicle_type_to_string(df_row):
    if int(df_row['Sports Car']) == 1:
        vehicle_type = 'Sports'
    elif int(df_row['SUV']) == 1:
        vehicle_type = 'Sports'
    elif int(df_row['Wagon']) == 1:
        vehicle_type = 'Wagon'
    elif int(df_row['Minivan']) == 1:
        vehicle_type = 'Minivan'
    elif int(df_row['Pickup']) == 1:
        vehicle_type = 'Pickup'
    else:
        # print("Vehicle type not identified")
        vehicle_type = 'Unknown'
    return vehicle_type

df_cars['Color'] = df_cars.apply(map_color_to_vehicle_type, axis=1)
```

```
df_cars['Vehicle Type'] = df_cars.apply(map_vehicle_type_to_string,
↪  axis=1)

# y_min = int(toyota_hp_vs_mpg['City MPG'].min(0))
# y_max = int(toyota_hp_vs_mpg['City MPG'].max(0))
# x_min = int(toyota_hp_vs_mpg['HP'].min(0))
# x_max = int(toyota_hp_vs_mpg['HP'].max(0))

# Let the size of the marker represent the weight of the vehicle:
#
↪  https://stackoverflow.com/questions/14827650/pyplot-scatter-plot-marker-size
# size = [int(w) for w in df_cars['Area'].values]
# normalize:
# size = size / np.linalg.norm(size)
vehicle_scatter = plt.scatter(x, y, marker='s',
↪  facecolors=df_cars['Color'], edgecolor='black', linewidths=0.5)

# ax.scatter(x, y, marker='s', c='bue', facecolors='None')
# ax.scatter(toyota_only['HP'], toyota_only['City MPG'], marker='s',
↪  c='green')
# plt.axis(y=np.arange(10, 60, 5), x=np.arange(73, 500, 42.7))
plt.xticks(np.arange(73, 542.7, 42.7))
# ax.legend((df_cars['Sports Car'], df_cars['SUV'], df_cars['Wagon'],
↪  df_cars['Minivan'], df_cars['Pickup']), ('Yellow', 'Green', 'Black',
↪  'Cyan', 'Purple'))
yellow_patch = mpatches.Patch(color='yellow', label='Sports Car')
green_patch = mpatches.Patch(color='green', label='SUV')
black_patch = mpatches.Patch(color='black', label='Wagon')
cyan_patch = mpatches.Patch(color='cyan', label='Minivan')
purple_patch = mpatches.Patch(color='red', label='Pickup')
none_patch = mpatches.Patch(color='none', label='Unknown')
plt.legend(handles=[yellow_patch, green_patch, black_patch, cyan_patch,
↪  purple_patch, none_patch])
plt.xlabel('Horse Power')
plt.ylabel('City Miles-Per-Gallon')
plt.title('Car Horse Power and Miles-Per-Gallon')
plt.savefig(fname=sys.argv[2])
# plt.show()
```

1.3.2 ./visuals/fig_1_47/campellcl/fig_1_47.png



1.4 ./visuals/fig_1_47/wascherb

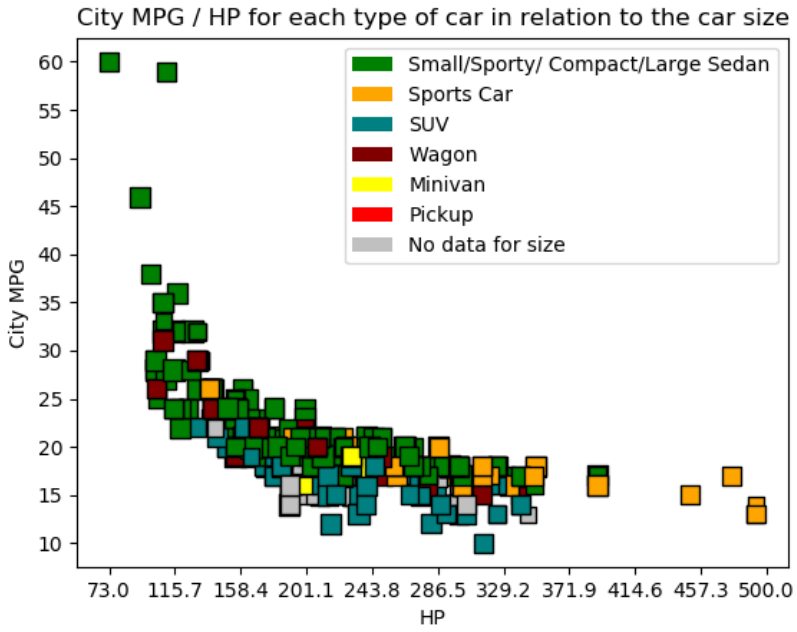
1.4.1 ./visuals/fig_1_47/wascherb/fig_1_41.py

```
import matplotlib.pylab as plt
import matplotlib.patches as mpatches
import numpy as np
import sys


def results(file_name):
    """
    clean data for representation
    :param file_name: input filename for data
    :return: None
    """

    x = []
    y = []
    car_type = []
    size = []
    leg_names = []
    cnt = 0
    index_x, index_y = 0, 0
    with open(file_name) as f:
        for line in f:
            content = str(line).strip()
            elements = content.split(',')

            if cnt != 0:
                last_element = len(elements) - 1
                try:
                    if elements[index_x].isdigit() and
↪   elements[index_y].isdigit():
                        x.append(int(elements[index_x]))
                        y.append(int(elements[index_y]))
                        if str(elements[last_element])[:-1].isdigit() and
↪   elements[last_element - 1].isdigit() and \
                                elements[last_element - 3].isdigit():
                            car_type.append(elements.index('1'))
                            area = int(str(elements[last_element])[:-1])
↪   * int(elements[last_element - 1])
                            weight = int(elements[last_element - 3])
                            size.append(area / weight * 250)
                        else:
                            car_type.append(7)
                            # print('%-30s: %s' % (elements[0],
↪   elements.index('1')))
                            # size.append(0.3567027132923991 * 250)
                            # print('FAULT')
```

```
                except:
                    print(elements[index_x])
                    print(elements[index_y])
            else:
                index_x = elements.index('HP')
                index_y = elements.index('City MPG')
                for i in range(1, 7):
                    leg_names.append(elements[i])
                leg_names.append('No data for size')
            cnt += 1

    plot(x, y, car_type, size, leg_names)


def plot(x, y, car_type, size, leg_names):
    """
    creating the scatter plot of the data
    :param x: HP data
    :param y: MPG data
    :param car_type: type classes
    :param size: sizes of rectangles
    :param leg_names: legend names extracted from the data sorces
    :return: None
    """

    plt.title('City MPG / HP for each type of car in relation to the car
↪   size')

    color_map = {1: 'green', 2: 'orange', 3: 'teal', 4: 'maroon', 5:
↪   'yellow', 6: 'red', 7: 'silver'}
    colors = []
    for index, type in enumerate(car_type):
        colors.append(color_map[type])
        car_type[index] = color_map[type]

    plt.scatter(x, y, color=colors, s=size, marker='s',
↪   edgecolors='black')
    plt.xlabel('HP')
    plt.ylabel('City MPG')

    # scale steps
    plt.yticks(np.arange(10, 65, 5))
    plt.xticks(np.arange(min(x), max(x) + 42.7, 42.7))

    # Add legend
    recs = []
    for i in color_map.values():
        recs.append(mpatches.Rectangle((0, 0), 1, 1, fc=i))
    plt.legend(recs, leg_names, loc=1)
```

```
    # plt.show()
    plt.savefig(sys.argv[2])


if __name__ == '__main__':
    if sys.argv == 1:
        print('Accept one argument: No input file for data')
        exit(0)

    results(sys.argv[1])
```

1.4.2 ./visuals/fig_1_47/wascherb/fig_1_47.png



1.5 ./visuals/fig_1_47/stokesnl

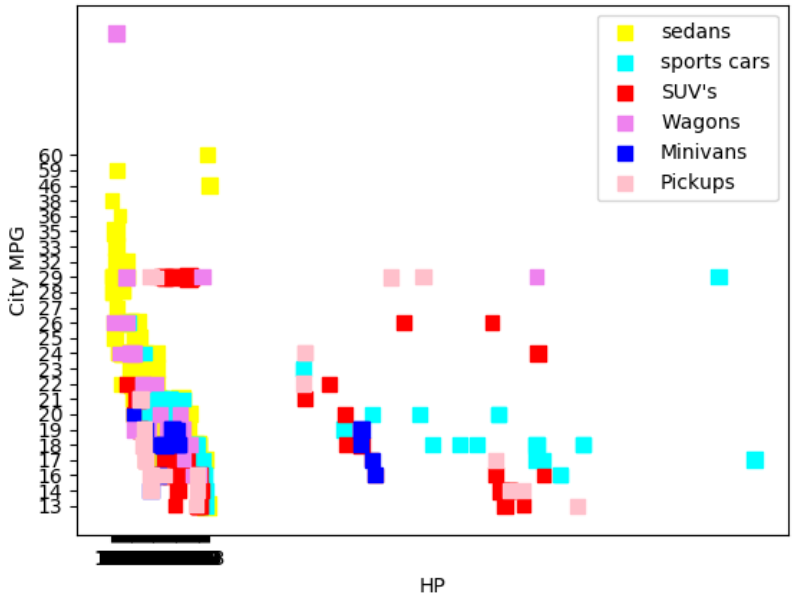
1.5.1 ./visuals/fig_1_47/stokesnl/fig_1_47.py

```
#!/usr/bin/env python
import csv
import matplotlib.pyplot as plt
from collections import defaultdict
import sys

columns = defaultdict(list) # each value in each column is appended to a
↪  list

with open(sys.argv[1]) as f:
    reader = csv.DictReader(f) # read rows into a dictionary format
    for row in reader: # read a row as {column1: value1, column2:
↪  value2,...}
        for (k,v) in row.items(): # go over each column name and value
            columns[k].append(v) # append the value into the appropriate
↪  list
                                 # based on column name k
#test = ['225', '125', '231']
toremove = []
for i in range(1,len(columns['HP'])):
    if columns['City MPG'][i] == '*':
        toremove.append(i)
for i in reversed(toremove):
    del columns['HP'][i]
    del columns['City MPG'][i]
    del columns['Weight'][i]
    del columns['Small/Sporty/ Compact/Large Sedan'][i]
    del columns['Sports Car'][i]
    del columns['SUV'][i]
    del columns['Wagon'][i]
    del columns['Minivan'][i]
    del columns['Pickup'][i]
toremove = []
for i in range(1,len(columns['Weight'])):
    if columns['Weight'][i] == '*':
        toremove.append(i)
for i in reversed(toremove):
    del columns['HP'][i]
    del columns['City MPG'][i]
    del columns['Weight'][i]
    del columns['Small/Sporty/ Compact/Large Sedan'][i]
    del columns['Sports Car'][i]
    del columns['SUV'][i]
    del columns['Wagon'][i]
    del columns['Minivan'][i]
    del columns['Pickup'][i]
```

```
sedan = []
sports =[]
suv = []
wagon = []
minivan = []
pickup = []
for i in range(1,len(columns['Weight'])):
    if columns['Small/Sporty/ Compact/Large Sedan'][i] == '1':
        sedan.append(i)
    elif columns['Sports Car'][i] == '1':
        sports.append(i)
    elif columns['SUV'][i] == '1':
        suv.append(i)
    elif columns['Wagon'][i] == '1':
        wagon.append(i)
    elif columns['Minivan'][i] == '1':
        minivan.append(i)
    elif columns['Pickup'][i] == '1':
        pickup.append(i)
x = [float(i) for i in columns['HP']]
y = [float(i) for i in columns['City MPG']]
weight = [float(i) for i in columns['Weight']]
weight = [x / 70 for x in weight]
#[float(i) for i in columns['City MPG']]
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax1.scatter([columns['HP'][i] for i in sedan], [columns['City MPG'][i]
 ↪  for i in sedan],
s = weight, marker="s", c="yellow", edgecolors="face", label="sedans")

ax1.scatter([columns['HP'][i] for i in sports], [columns['City MPG'][i]
 ↪  for i in sports],
s = weight, marker="s", c="cyan",edgecolors="face", label="sports cars")

ax1.scatter([columns['HP'][i] for i in suv], [columns['City MPG'][i] for
 ↪  i in suv],
s = weight, marker="s", c="r",edgecolors="face", label="SUV's")

ax1.scatter([columns['HP'][i] for i in wagon], [columns['City MPG'][i]
 ↪  for i in wagon],
s = weight, marker="s", c="violet",edgecolors="face", label="Wagons")

ax1.scatter([columns['HP'][i] for i in minivan], [columns['City MPG'][i]
 ↪  for i in minivan],
s = weight, marker="s", c="b",edgecolors="face", label="Minivans")

ax1.scatter([columns['HP'][i] for i in pickup], [columns['City MPG'][i]
 ↪  for i in pickup],
s = weight, marker="s", c="pink",edgecolors="face", label="Pickups")
ax1.legend()
```

```
#plt.scatter(x, y, s=weight, marker="s");
plt.ylabel("City MPG")
plt.xlabel("HP")
plt.savefig(sys.argv[2]);
#print(columns['Vehicle Name'])
```

1.5.2 ./visuals/fig_1_47/stokesnl/fig_1_47.png

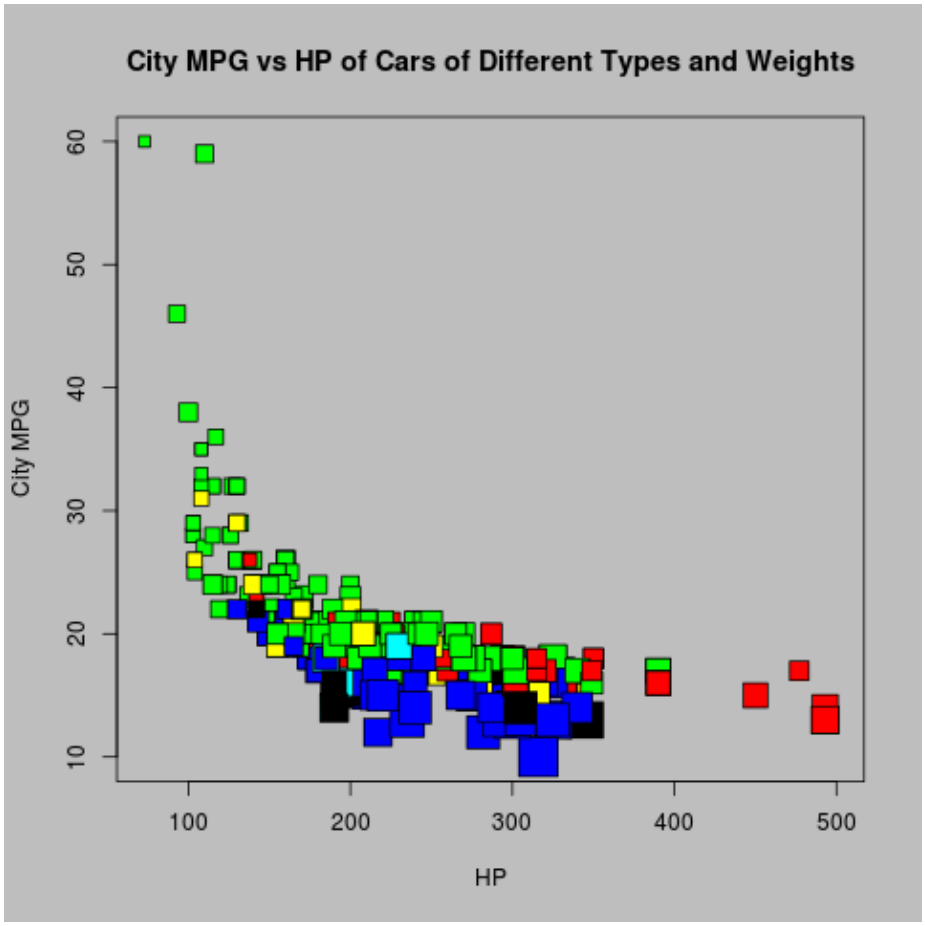


1.6 ./visuals/fig_1_47/davisjb2

1.6.1 ./visuals/fig_1_47/davisjb2/fig_1_41.R

```
args <- commandArgs()
x <- read.table(args[6],header=T,sep=",")
str(x)
ex <- as.numeric(as.character(x$City.MPG))
qu <- as.numeric(as.character(x$HP))
w <- as.numeric(as.character(x$Weight))
x$Color = "black"
x$Color[x$Small.Sporty..Compact.Large.Sedan == 1] = "green"
x$Color[x$Sports.Car == 1] = "red"
x$Color[x$SUV == 1] = "blue"
x$Color[x$Wagon == 1] = "yellow"
x$Color[x$Minivan == 1] = "cyan"
x$Color[x$Pickup == 1] = "black"

png(filename=args[7])
par(bg = "grey")
plot(qu,ex,pch=22,xlab = "HP",ylab = "City MPG", main = 'City MPG vs HP
↪  of Cars of Different Types and Weights', cex=w/1500, bg=x$Color)
dev.off()
```

1.6.2 ./visuals/fig_1_47/davisjb2/fig_1_47.png



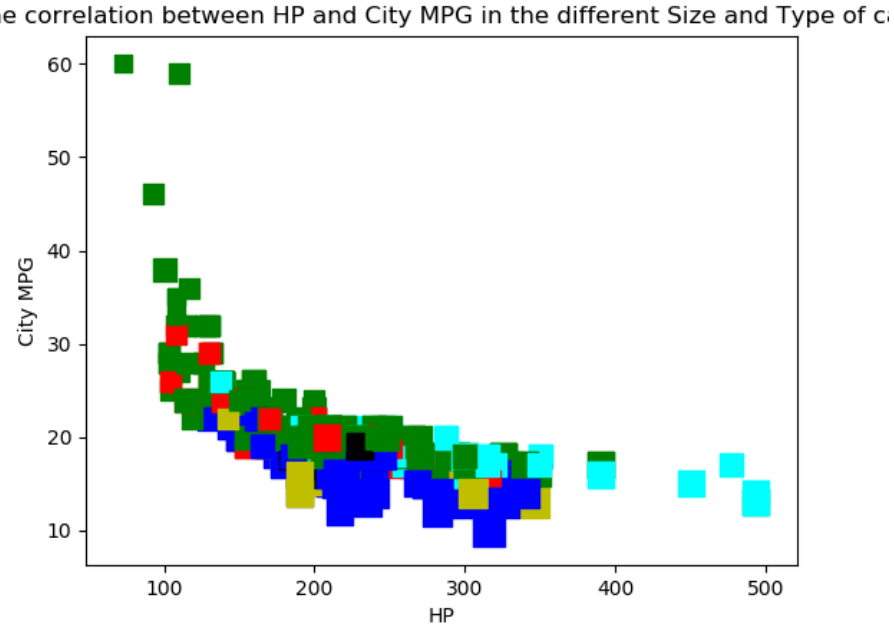
1.7 ./visuals/fig_1_47/zhengn

1.7.1 ./visuals/fig_1_47/zhengn/fig_1_41.py

```
__author__ = 'Naibin Zheng'
import numpy as np
import matplotlib.pyplot as plt
import csv
import pandas as pd
from matplotlib.patches import Rectangle
import sys


def main():
    data = pd.read_csv(sys.argv[1])
    hp = data['HP']
    mpg = data['City MPG']
    weight = data['Weight']
    sedan = data['Small/Sporty/ Compact/Large Sedan']
    sc = data['Sports Car']
    suv = data['SUV']
    wagon = data['Wagon']
    minivan = data['Minivan']
    pickup=data['Pickup']


    #print(type)

    plt.xlabel('HP')
    plt.ylabel('City MPG')
    plt.title('The correlation between HP and City MPG in the different
↪   Size and Type of car')

    #for h, m, w, s, su, w, m, p in zip(hp, mpg, weight, sc, suv, wagon,
↪   minivan, pickup):
    for h, m, w, sd, s, su, wg, mv, p in zip(hp, mpg, weight, sedan, sc,
↪   suv, wagon, minivan, pickup):
        if h != '*' and m != '*' and w != '*' and sd == 1:
            area = float(w)/25
            h = float(h)
            m = float(m)
            plt.scatter(h, m, marker='s', s=area, c='g')
        elif h != '*' and m != '*' and w != '*' and s == 1:
            area = float(w)/25
            h = float(h)
            m = float(m)
            plt.scatter(h, m, marker='s', s=area, c='cyan')
        elif h != '*' and m != '*' and w != '*' and su == 1:
            area = float(w)/25
            h = float(h)
```

```
            m = float(m)
            plt.scatter(h, m, marker='s', s=area, c='blue')
        elif h != '*' and m != '*' and w != '*' and p == 1:
            area = float(w)/25
            h = float(h)
            m = float(m)
            plt.scatter(h, m, marker='s', s=area, c='y')
        elif h != '*' and m != '*' and w != '*' and wg == 1:
            area = float(w)/25
            h = float(h)
            m = float(m)
            plt.scatter(h, m, marker='s', s=area, c='r')
        elif h != '*' and m != '*' and w != '*' and mv == 1:
            area = float(w)/25
            h = float(h)
            m = float(m)
            plt.scatter(h, m, marker='s', s=area, c='black')

    #plt.show()
    plt.savefig(sys.argv[2])


if __name__ == '__main__':
    main()
```

1.7.2 ./visuals/fig_1_47/zhengn/fig_1_47.png

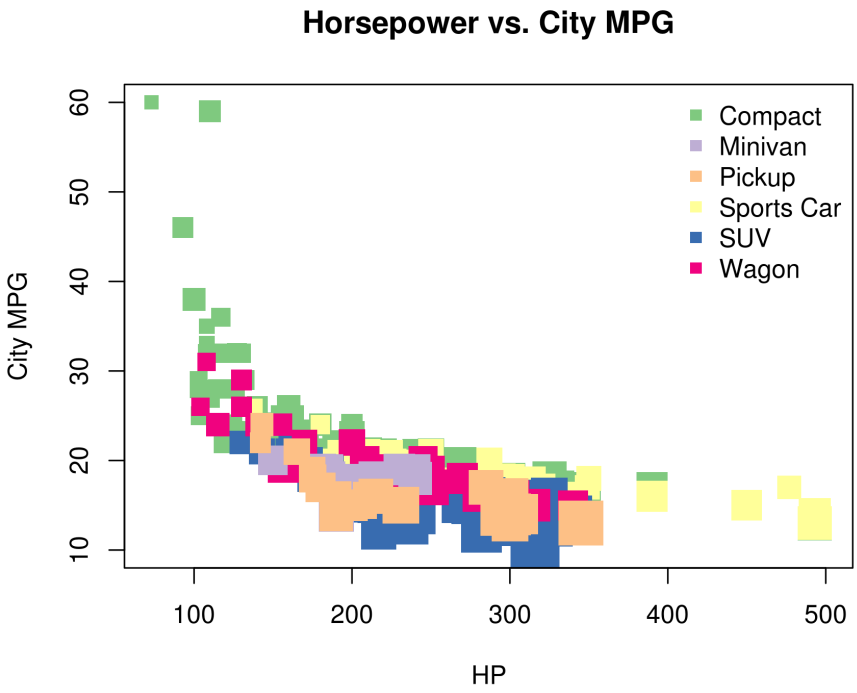


1.8 ./visuals/fig_1_47/smithkj2

1.8.1 ./visuals/fig_1_47/smithkj2/Program1.R

```
library(readr)
library(dplyr)
library(tidyr)
library(RColorBrewer)
t <- proc.time()

args <- commandArgs(T)
print(args)
name <- args[1]
cars <- read_csv(name)

cars <- cars %>%
  gather(class, value, c(`Small/Sporty/ Compact/Large Sedan`, `Sports
↪  Car`, SUV, Wagon, Minivan, Pickup)) %>%
  filter(value == 1)
cars$class <- ifelse(cars$class == 'Small/Sporty/ Compact/Large Sedan',
↪  'Compact', cars$class)
cars$Weight[is.na(cars$Weight)] <- median(cars$Weight)
cars$Weight <- as.numeric(cars$Weight)

colors <- with(cars,
               data.frame(class = levels(factor(class)),
                          color =
↪  I(brewer.pal(nlevels(factor(cars$class)),
                                                 name = 'Accent'))))
png(filename=args[2],
    width = 6,
    height = 5,
    units = 'in',
    res = 300)
plot(cars$HP,
     cars$`City MPG`,
     xlab = 'HP',
     ylab = 'City MPG',
     main = 'Horsepower vs. City MPG',
     col = colors$color[match(cars$class, colors$class)],
     pch = 15,
     cex = cars$Weight/1500)
legend(x = 'topright',
       legend = as.character(colors$class),
       col = colors$color,
       pch = 15,
       bty = 'n')

#dev.off()
t-proc.time()
```

1.8.2 ./visuals/fig_1_47/smithkj2/fig_1_47.png

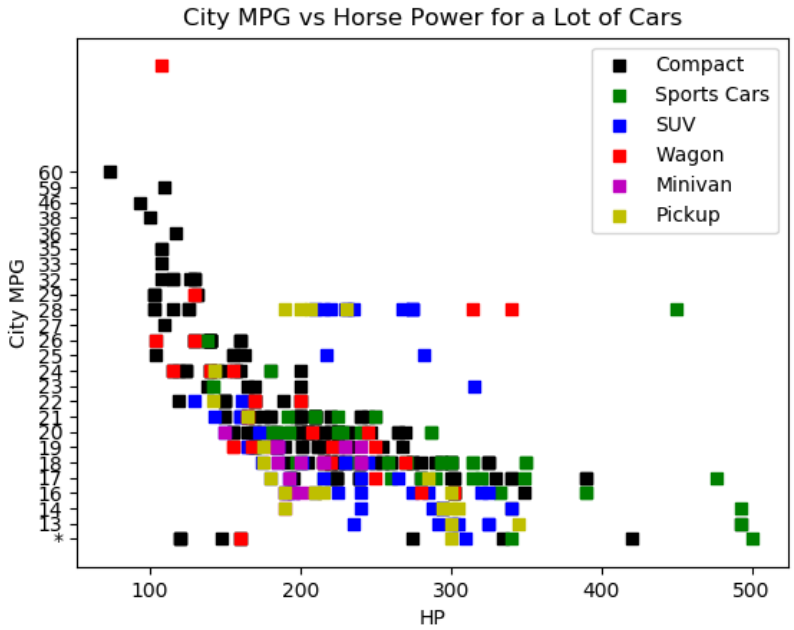


1.9 ./visuals/fig_1_47/carnsds

1.9.1 ./visuals/fig_1_47/carnsds/fig_1_47.py

```
#Dillon Carns
#1/28/2018
import matplotlib.pyplot as plt
import pandas as pand

import sys

fields = ['Small/Sporty/ Compact/Large Sedan','Sports Car','SUV','Wagon',
↪  'Minivan',
'Pickup','HP', 'City MPG', 'Weight']
df = pand.read_csv(sys.argv[1], skipinitialspace=True, usecols=fields)


plt.title("City MPG vs Horse Power for a Lot of Cars")
plt.xlabel("HP")
plt.ylabel("City MPG")
plt.scatter(df.loc[df['Small/Sporty/ Compact/Large Sedan'] == 1, 'HP'],
 df.loc[df['Small/Sporty/ Compact/Large Sedan'] == 1, 'City MPG'], c='k',
↪  marker='s')
plt.scatter(df.loc[df['Sports Car'] == 1, 'HP'],
 df.loc[df['Sports Car'] == 1, 'City MPG'], c='g', marker='s')
plt.scatter(df.loc[df['SUV'] == 1, 'HP'],
 df.loc[df['SUV'] == 1, 'City MPG'], c='b', marker = 's')
plt.scatter(df.loc[df['Wagon'] == 1, 'HP'],
 df.loc[df['Wagon'] == 1, 'City MPG'], c='r', marker = 's')
plt.scatter(df.loc[df['Minivan'] == 1, 'HP'],
 df.loc[df['Minivan'] == 1, 'City MPG'], c='m', marker = 's')
plt.scatter(df.loc[df['Pickup'] == 1, 'HP'],
 df.loc[df['Pickup'] == 1, 'City MPG'], c='y', marker = 's')
plt.legend(['Compact', 'Sports Cars', 'SUV', 'Wagon', 'Minivan',
↪  'Pickup'])
plt.savefig(sys.argv[2], Transparent=True)
```

1.9.2 ./visuals/fig_1_47/carnsds/fig_1_47.png

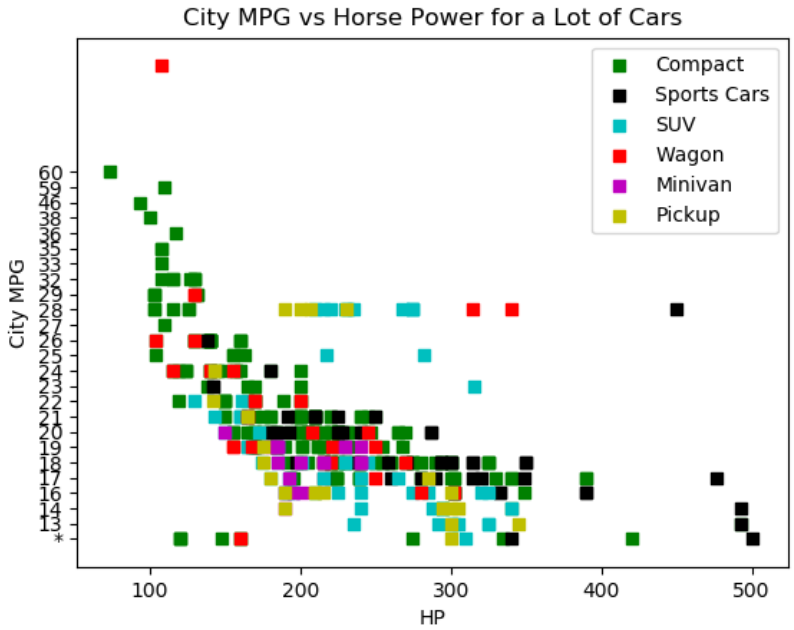


1.10 ./visuals/fig_1_47/oliverj

1.10.1 ./visuals/fig_1_47/oliverj/fig_1_47.py

```
#Hunter Oliver
#1/28/2018
import matplotlib.pyplot as plt
import pandas as pand
import sys

vehicles = ['Small/Sporty/ Compact/Large Sedan','Sports
↪  Car','SUV','Wagon', 'Minivan',
'Pickup','HP', 'City MPG', 'Weight']
df = pand.read_csv(sys.argv[1], skipinitialspace=True, usecols=vehicles)


plt.title("City MPG vs Horse Power for a Lot of Cars")
plt.xlabel("HP")
plt.ylabel("City MPG")
plt.scatter(df.loc[df['Small/Sporty/ Compact/Large Sedan'] == 1, 'HP'],
 df.loc[df['Small/Sporty/ Compact/Large Sedan'] == 1, 'City MPG'], c='g',
↪  marker='s')
plt.scatter(df.loc[df['Sports Car'] == 1, 'HP'],
 df.loc[df['Sports Car'] == 1, 'City MPG'], c='k', marker='s')
plt.scatter(df.loc[df['SUV'] == 1, 'HP'],
 df.loc[df['SUV'] == 1, 'City MPG'], c='c', marker = 's')
plt.scatter(df.loc[df['Wagon'] == 1, 'HP'],
 df.loc[df['Wagon'] == 1, 'City MPG'], c='r', marker = 's')
plt.scatter(df.loc[df['Minivan'] == 1, 'HP'],
 df.loc[df['Minivan'] == 1, 'City MPG'], c='m', marker = 's')
plt.scatter(df.loc[df['Pickup'] == 1, 'HP'],
 df.loc[df['Pickup'] == 1, 'City MPG'], c='y', marker = 's')
plt.legend(['Compact', 'Sports Cars', 'SUV', 'Wagon', 'Minivan',
↪  'Pickup'])
plt.savefig(sys.argv[2], Transparent=True)
```

1.10.2 ./visuals/fig_1_47/oliverj/fig_1_47.png



1.11 ./visuals/fig_1_47/halvorsenca

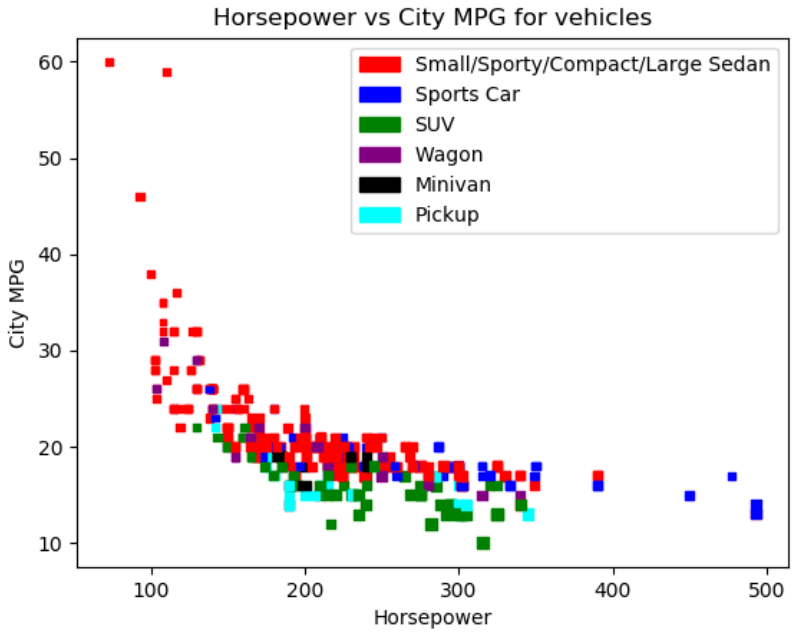
1.11.1 ./visuals/fig_1_47/halvorsenca/scatterCar.py

```
import csv
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import numpy as np
import sys

file = sys.argv[1]
data = pd.read_csv(file)

details= pd.DataFrame(data, columns=['Vehicle Name', 'Small/Sporty/
↪  Compact/Large Sedan','Sports Car', 'SUV', 'Wagon', 'Minivan',
↪  'Pickup', 'AWD', 'RWD', 'Retail Price', 'Dealer Cost', 'Engine Size
↪  (l)', 'Cyl', 'HP', 'City MPG', 'Hwy MPG', 'Weight', 'Wheel Base',
↪  'Len', 'Width'])

HP = list(details['HP'])
MPG = list(details['City MPG'])
WEIGHT = details['Weight']


temp = []
for m in MPG:
    if m == '*':
        temp.append(0)
    else:
        temp.append(float(m))

MPG = temp

temp2 = []
for w in WEIGHT:
    if w == '*':
        temp2.append(0)
    else:
        temp2.append(float(w))

WEIGHT=temp2

type={}

types = pd.DataFrame(data, columns=['Small/Sporty/ Compact/Large Sedan',
↪  'Sports Car', 'SUV', 'Wagon', 'Minivan', 'Pickup'])
cars = pd.DataFrame(types).to_dict()

for c in cars:
```

```
    for i in cars[c]:
        if cars[c][i] == 1:
            type[i] = c

color_dict = {'Small/Sporty/ Compact/Large Sedan': 'red', 'Sports Car':
↪  'blue', 'SUV': 'green', 'Wagon': 'purple', 'Minivan': 'black',
↪  'Pickup':'cyan'}

colors = {}
for every in type:
    colors[every] = color_dict[type[every]]

color_list = []
for i in range(len(colors)):
    color_list.append(colors[i])

x = zip(HP, MPG, color_list, WEIGHT)
x = filter(lambda item: item[0] != 0, x)
x = filter(lambda item: item[1] != 0, x)
x = filter(lambda item: item[3] != 0, x)

"""
for i in x:
    if 0 in i:
        x.remove(i)
"""
Hp, Mpg, colorr, weight = map(list, zip(*x))

"""
Mpg = np.array(MPG)
weight = np.array(WEIGHT)
Hp = np.array(HP)[Mpg != 0].tolist()
colorr = np.array(color_list)[Mpg != 0].tolist()
weight= np.array(WEIGHT)[Mpg != 0].tolist()
Mpg = Mpg[Mpg != 0].tolist()

"""


r_patch = mpatches.Patch(color='red', label= 'Small/Sporty/Compact/Large
↪  Sedan')
b_patch = mpatches.Patch(color='blue', label='Sports Car')
g_patch = mpatches.Patch(color='green', label='SUV')
p_patch = mpatches.Patch(color='purple', label='Wagon')
bl_patch = mpatches.Patch(color='black', label='Minivan')
c_patch = mpatches.Patch(color='cyan', label='Pickup')

tempw = []
for w in weight:
    tempw.append(w * .005)
```

```
weight = tempw


plt.scatter(Hp, Mpg, s=weight, marker="s", color=colorr)
plt.legend(handles=[r_patch,b_patch,g_patch,p_patch,bl_patch,c_patch])
plt.xlabel('Horsepower')
plt.ylabel('City MPG')
plt.title('Horsepower vs City MPG for vehicles')
plt.savefig(sys.argv[2])
```

1.11.2 ./visuals/fig_1_47/halvorsenca/fig_1_47.png



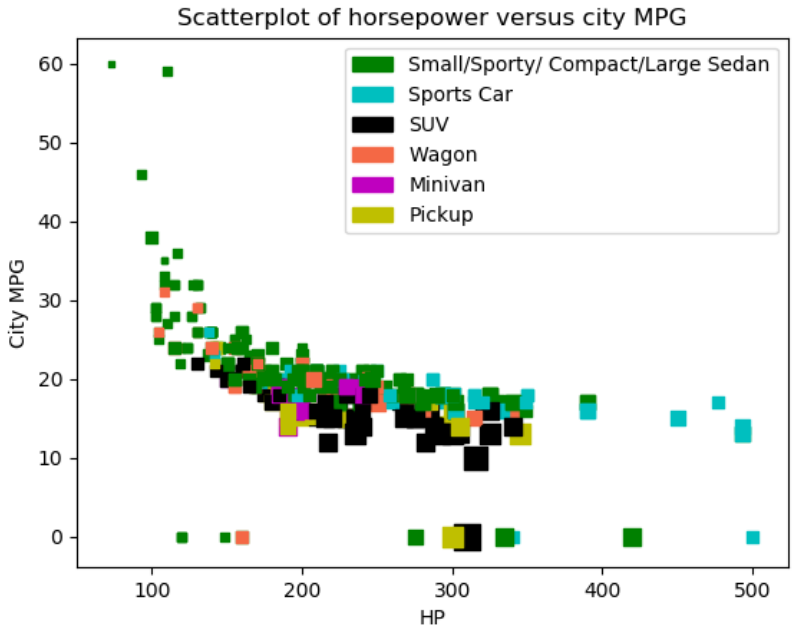
1.12 ./visuals/fig_1_47/laresaguilared

1.12.1 ./visuals/fig_1_47/laresaguilared/ScatterPlot.py

```
import csv
import matplotlib.patches as mpatches
from matplotlib import pyplot as plt
import sys

def read_file():
    table_list = []
    with open(sys.argv[1]) as csvfile:
            readCSV = csv.reader(csvfile, delimiter=',')
            for row in readCSV:
                if len(row) > 0:
                    table_list.append(row)
    return table_list



def main():
    table = read_file()
    # print(table)
    counter = 0
    for cars in table:
        if counter > 0:

            try:
                x = float(cars[13])
            except:
                x = 0
            try:
                y = float(cars[14])
            except:
                y = 0

            try:
                size = (float(cars[16])*float(cars[16])) * 0.000003
            except:
                size = 0

            if cars[1] == "1":
                plt.scatter(x, y, c="g", marker='s', s=size)
            elif cars[2] == "1":
                plt.scatter(x, y, c="c", marker='s', s=size)
            elif cars[3] == "1":
                plt.scatter(x, y, c="k", marker='s', s=size)
            elif cars[4] == "1":
                plt.scatter(x, y, c="#f46845", marker='s', s=size)
            elif cars[5] == "1":
```

```
                plt.scatter(x, y, c="m", marker='s', s=size)
            elif cars[6] == "1":
                plt.scatter(x, y, c="y", marker='s', s=size)
        else:
            counter += 1

    plt.xlabel("HP")
    plt.ylabel("City MPG")
    plt.title("Scatterplot of horsepower versus city MPG")
    small = mpatches.Patch(color='g', label='Small/Sporty/ Compact/Large
↪  Sedan')
    Sports = mpatches.Patch(color='c', label='Sports Car')
    SUV = mpatches.Patch(color='k', label='SUV')
    Wagon = mpatches.Patch(color='#f46845', label='Wagon')
    Minivan = mpatches.Patch(color='m', label='Minivan')
    Pickup = mpatches.Patch(color='y', label='Pickup')
    plt.legend(handles=[small, Sports, SUV, Wagon, Minivan, Pickup])
    plt.savefig(sys.argv[2])


if __name__ == '__main__':
    main()
```

1.12.2 ./visuals/fig_1_47/laresaguilared/fig_1_47.png



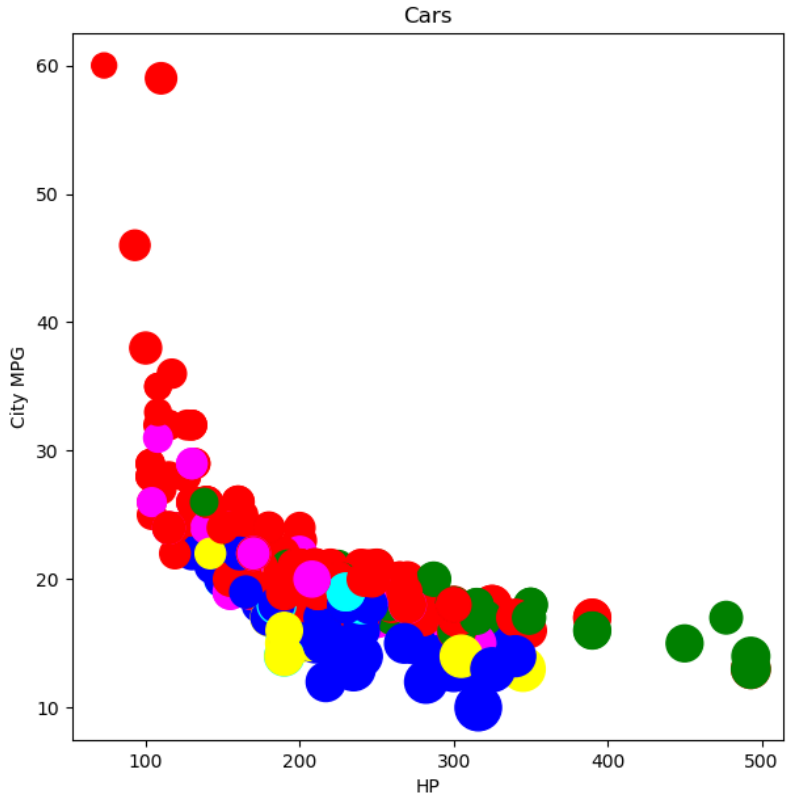
1.13 ./visuals/fig_1_47/beasonke

1.13.1 ./visuals/fig_1_47/beasonke/fig_1_47.py

```
import matplotlib.pyplot as plt
import pip
pip.main(['install','pandas'])
import pandas as pd
import seaborn
import sys

df = pd.read_csv(sys.argv[1])
df = df[["HP", "City MPG", "Weight", "Small/Sporty/ Compact/Large Sedan",
 ↪  "Sports Car", "SUV", "Wagon", "Minivan", "Pickup"]]
df = df[df["City MPG"] != '*']
df = df[df["Weight"] != '*']


def cat(row):
    if row['Small/Sporty/ Compact/Large Sedan'] == 1:
        return 'Small/Sporty/ Compact/Large Sedan'
    if row['Sports Car'] == 1:
        return 'Sports Car'
    if row['SUV'] == 1:
        return 'SUV'
    if row['Wagon'] == 1:
        return 'Wagon'
    if row['Minivan'] == 1:
        return 'Minivan'
    if row['Pickup'] == 1:
        return 'Pickup'
    else:
        return 'no cat'


categories = []
for index, row in df.iterrows():
    categories.append(cat(row))

df['category'] = categories
columns = ["Small/Sporty/ Compact/Large Sedan", "Sports Car", "SUV",
 ↪  "Wagon", "Minivan", "Pickup"]
df.drop(columns, inplace=True, axis=1)
# print(df)
# pd.set_option('display.max_rows', 500)
df["City MPG"] = pd.to_numeric(df["City MPG"])
df["Weight"] = pd.to_numeric(df["Weight"])

colors = {'Small/Sporty/ Compact/Large Sedan': 'red', 'Sports Car':
 ↪  'green', 'SUV': 'blue',
```

```
          'Wagon': 'magenta', 'Minivan': 'cyan', 'Pickup': 'yellow'}
plot = df.plot(x="HP", y="City MPG", s=df['Weight'] / 10,
↪   c=df['category'].apply(lambda x: colors[x]),
               kind='scatter', figsize = (7,7), title='Cars')
plot.set_xlabel("HP")
plot.set_ylabel("City MPG")

plt.savefig(sys.argv[2])
```

1.13.2 ./visuals/fig_1_47/beasonke/fig_1_47.png



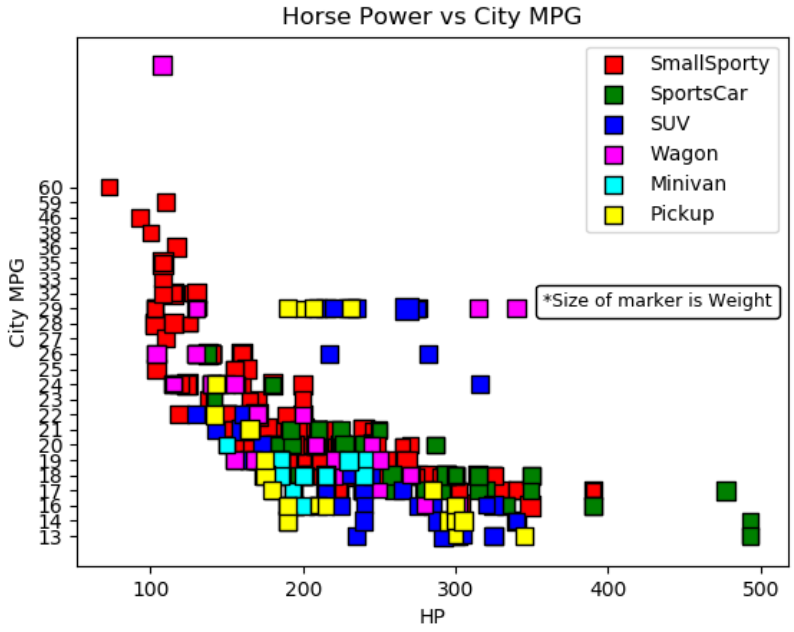
1.14 ./visuals/fig_1_47/beekmanpc

1.14.1 ./visuals/fig_1_47/beekmanpc/hw1.py

```
import sys
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid.anchored_artists import AnchoredText
import pandas as pd


def main():
    cars = pd.read_csv(sys.argv[1])
    cars.columns =
↪   ['VehicleName','SmallSporty','SportsCar','SUV','Wagon','Minivan','Pickup',
↪   'AWD',

↪   'RWD','RetailPrice','DealerCost','EngineSize(l)','Cyl','HP','CityMPG','HwyMPG','Weight','WheelBas

    cars['Type'] = 0

    # set the Type value based on the car
    cars.loc[cars.SmallSporty == 1, 'Type'] = 1
    cars.loc[cars.SportsCar == 1, 'Type'] = 2
    cars.loc[cars.SUV == 1, 'Type'] = 3
    cars.loc[cars.Wagon == 1, 'Type'] = 4
    cars.loc[cars.Minivan == 1, 'Type'] = 5
    cars.loc[cars.Pickup == 1, 'Type'] = 6

    # clean the data removing any '*' and converting str to ints
    cars = cars[cars.CityMPG != '*']
    cars = cars[cars.Weight != '*']
    cars.Weight = pd.to_numeric(cars.Weight, errors='coerce')

    # create and display the scatterplot
    num = 1
    fig, ax = plt.subplots()
    for color in ['red','green','blue','magenta','cyan','yellow']:
        X = cars['HP'].where(cars['Type'] == num).dropna()
        Y = cars['CityMPG'].where(cars['Type'] == num).dropna()
        size = cars['Weight'] / 50
        ax.scatter(X, Y, c=color, s=size, marker='s', edgecolors=(0,0,0),
↪   label=cars.columns[num])
        num = num + 1

    at = AnchoredText("*Size of marker is Weight",
                      prop=dict(size=9), frameon=True,
                      loc=7,
                      )
    at.patch.set_boxstyle("round,pad=0.,rounding_size=0.2")
    ax.add_artist(at)
```

```
    ax.legend()
    plt.xlabel("HP")
    plt.ylabel("City MPG")
    plt.title("Horse Power vs City MPG")
    plt.savefig(sys.argv[2])
    #plt.show()


if __name__ == "__main__":
    main()
```

1.14.2 ./visuals/fig_1_47/beekmanpc/fig_1_47.png

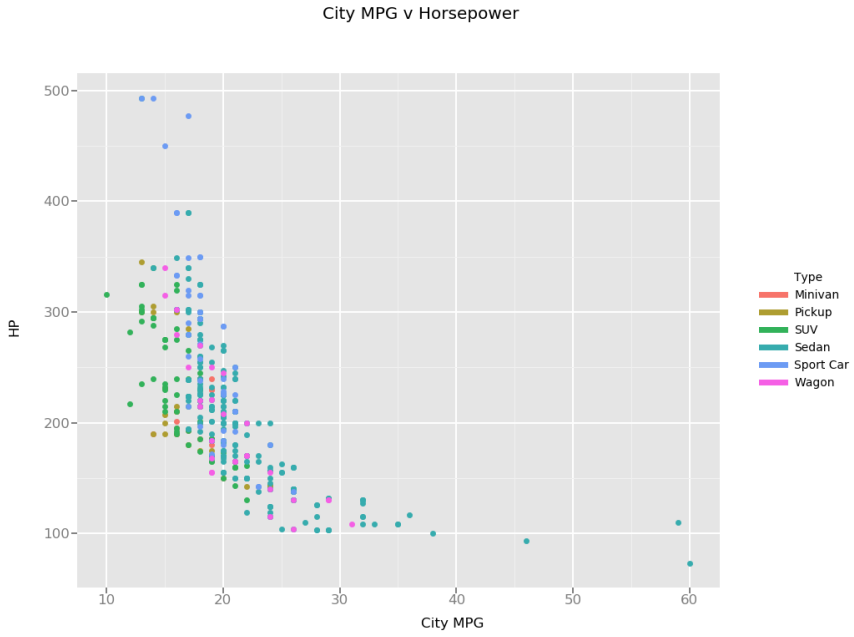


1.15 ./visuals/fig_1_47/emeryde

1.15.1 ./visuals/fig_1_47/emeryde/emery_fig_1_47.py

```
import sys

from pandas import read_csv
from ggplot import *

#df = read_csv('http://cs.appstate.edu/~rmp/cs5720/cars04.csv')
df = read_csv(sys.argv[1])

df = df.drop(df[(df['City MPG'] == '*')].index)
df = df.drop(df[(df['Weight'] == '*')].index)

def get_type():
    tmp = df['Vehicle Name'].values
    for i in range(len(tmp)):
        if(df['Small/Sporty/ Compact/Large Sedan'].values[i] == 1):
            tmp[i] = 'Sedan'
        elif df['SUV'].values[i] == 1:
            tmp[i] = 'SUV'
        elif df['Sports Car'].values[i] == 1:
            tmp[i] = 'Sport Car'
        elif df['Wagon'].values[i] == 1:
            tmp[i] = 'Wagon'
        elif df['Pickup'].values[i] == 1:
            tmp[i] = 'Pickup'
        else:
            tmp[i] = 'Minivan'
    return tmp

df['Type'] = get_type()
df['City MPG'] = df['City MPG'].astype(int)
df['Weight'] = df['Weight'].astype(int)/100

    #geom_point(aes(size = 'Weight')) +\
p=ggplot(df, aes(x='City MPG', y='HP', color = 'Type')) +\
    geom_point() +\
    xlab("City MPG") + ylab("HP") + ggtitle("City MPG v Horsepower")

p.save(sys.argv[2])
```

1.15.2 ./visuals/fig_1_47/emeryde/fig_1_47.png



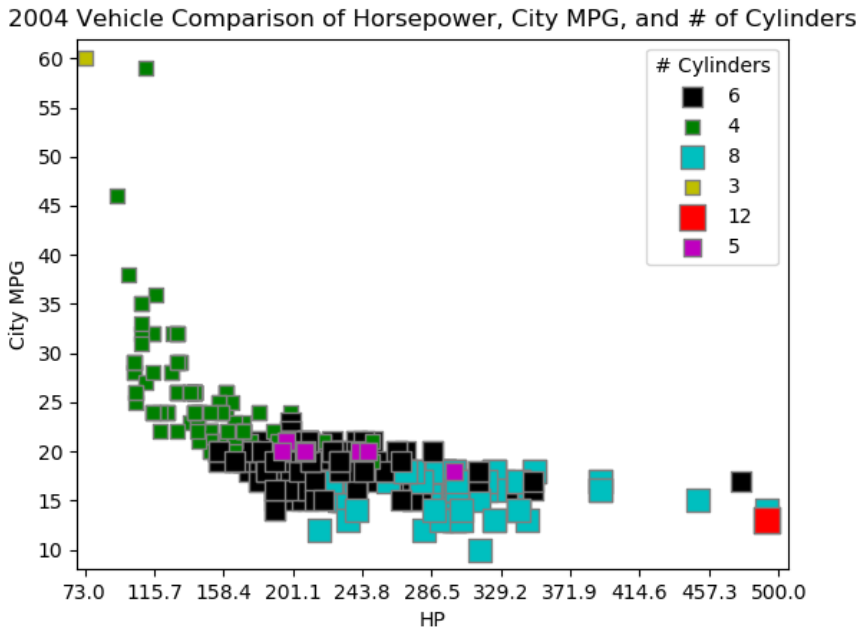
1.16 ./visuals/fig_1_47/parkerat2

1.16.1 ./visuals/fig_1_47/parkerat2/fig_1_47.py

```
import pandas as pd
import sys
import numpy as np
import matplotlib.pyplot as plt
from collections import OrderedDict

def main():
    args = sys.argv
    carsdf = pd.read_csv(args[1])
    # carsdf = pd.read_csv("cars04.csv")

    h = []
    c = []
    cy = set()
    for cmpg, hp, cyl, in zip(carsdf['City MPG'],
                carsdf['HP'], carsdf['Cyl']):
        if hp is not '*' and cmpg is not '*' and cyl is not -1:
            h.append(int(hp))
            c.append(int(cmpg))
            cy.add(cyl)
            hp=float(hp)
            cmpg=float(cmpg)
            cyl=float(cyl)
            if cyl == 3:
                plt.scatter(x=hp, y=cmpg, marker='s', s=(cyl * 15),
↪   color='y', edgecolors='gray', label=3)
            if cyl == 4:
                plt.scatter(x=hp, y=cmpg, marker='s', s=(cyl * 15),
↪   color='g', edgecolors='gray', label=4)
            if cyl == 5:
                plt.scatter(x=hp, y=cmpg, marker='s', s=(cyl * 15),
↪   color='m', edgecolors='gray', label=5)
            if cyl == 6:
                plt.scatter(x=hp, y=cmpg, marker='s', s=(cyl * 15),
↪   color='k', edgecolors='gray', label=6)
            if cyl == 8:
                plt.scatter(x=hp, y=cmpg, marker='s', s=(cyl * 15),
↪   color='c', edgecolors='gray', label=8)
            if cyl == 10:
                plt.scatter(x=hp, y=cmpg, marker='s', s=(cyl * 15),
↪   color='gray', edgecolors='gray', label=10)
            if cyl == 12:
                plt.scatter(x=hp, y=cmpg, marker='s', s=(cyl * 15),
↪   color='r', edgecolors='gray', label=12)

    plt.xticks(np.arange(min(h), max(h) + 10, 42.7))
```

```
    plt.yticks(np.arange(min(c), max(c)+1, 5))
    plt.xlim(xmin=min(h)-5, xmax=max(h)+13)
    plt.ylim(ymin=min(c)-2, ymax=max(c)+2)
    plt.xlabel("HP")
    plt.ylabel("City MPG")
    plt.title("2004 Vehicle Comparison of Horsepower, City MPG, and # of
↪   Cylinders")
    handles, labels = plt.gca().get_legend_handles_labels()
    by_label = OrderedDict(zip(labels, handles))
    plt.legend(by_label.values(), by_label.keys(), title="# Cylinders")
    # plt.show()
    plt.savefig(sys.argv[2])
    plt.clf()


if __name__ == '__main__':
    main()
```

1.16.2 ./visuals/fig_1_47/parkerat2/fig_1_47.png



Chapter 2

./visuals/fig_2_4

2.1 Description

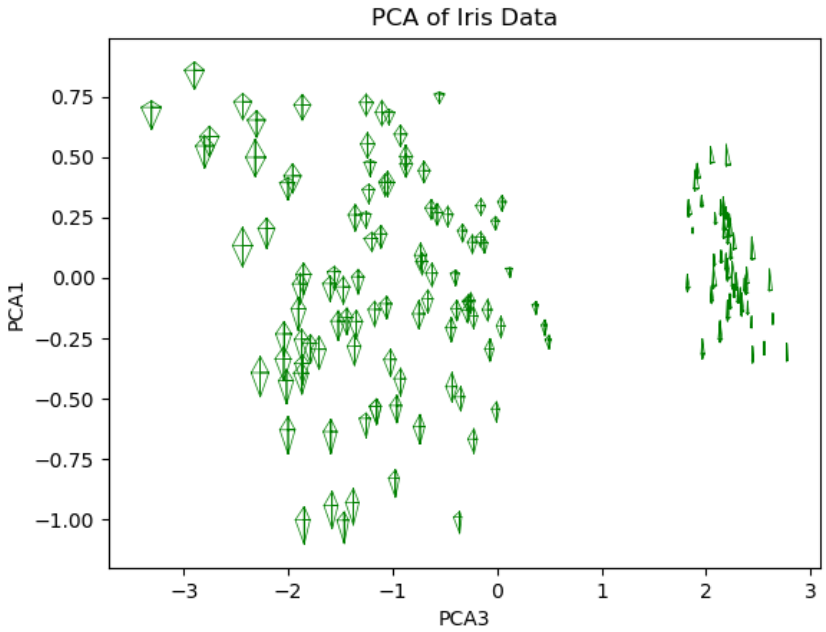
2.2 ./visuals/fig_2_4/caldwellc1

2.2.1 ./visuals/fig_2_4/caldwellc1/fig_2_4.py

```
import csv
import numpy
import sys
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import matplotlib.path as mpath


def main():
    data = []
    file_name = sys.argv[1]
    with open(file_name, 'r') as csvfile:
        read = csv.reader(csvfile, delimiter=',')
        for row in read:
            data.append(row)
    data = numpy.delete(data, 0, axis=0)
    data = numpy.delete(data, 4, axis=1)
    data = data.astype(float)
    sl_list = []
    sw_list = []
    pl_list = []
    pw_list = []
    for i in range(len(data)):
        sl_list.append(data[i][0])
        sw_list.append(data[i][1])
        pl_list.append(data[i][2])
        pw_list.append(data[i][3])
    data_s = StandardScaler().fit_transform(data)
    pca = PCA(n_components=4)
    x_r = pca.fit_transform(data_s)
    fig, ax = plt.subplots()
    for i in range(len(x_r)):
        x = x_r[i][0]
        y = x_r[i][2]
        sl =
↪   (data[i][0]-numpy.min(sl_list)+.05)/(numpy.max(sl_list)-numpy.min(sl_list)+.05)
        sl = sl*.1
        sw =
↪   (data[i][1]-numpy.min(sw_list)+.05)/(numpy.max(sw_list)-numpy.min(sw_list)+.05)
        sw = sw*.1
        pl =
↪   (data[i][2]-numpy.min(pl_list)+.05)/(numpy.max(pl_list)-numpy.min(pl_list)+.05)
        pl = pl*.1
        pw =
↪   (data[i][3]-numpy.min(pw_list)+.05)/(numpy.max(pw_list)-numpy.min(pw_list)+.05)
```

```
        pw = pw*.1
        path = mpath.Path
        path_data = [(path.MOVETO, (-x,-y)), (path.LINETO, (-x,-y+sw)),
↪   (path.MOVETO, (-x,-y)), (path.LINETO, (-x,-y-pw)),
                        (path.MOVETO, (-x,-y)), (path.LINETO,
↪   (-x+sl,-y)),(path.MOVETO, (-x,-y)), (path.LINETO, (-x-pl,-y)),
                    (path.LINETO, (-x,-y+sw)), (path.LINETO,
↪   (-x+sl,-y)),(path.LINETO, (-x,-y-pw)),(path.LINETO, (-x-pl,-y))]
        codes, verts = zip(*path_data)
        paths = mpath.Path(verts, codes)
        x1, y1 = zip(*paths.vertices)
        ax.plot(x1,y1,'g-', linewidth=.5)

    ax.set_xlabel("PCA3")
    ax.set_ylabel("PCA1")
    ax.set_title("PCA of Iris Data")
    fig.savefig(sys.argv[2])


if __name__ == '__main__':
    main()
```

2.2.2 ./visuals/fig_2_4/caldwellc1/fig_2_4.png



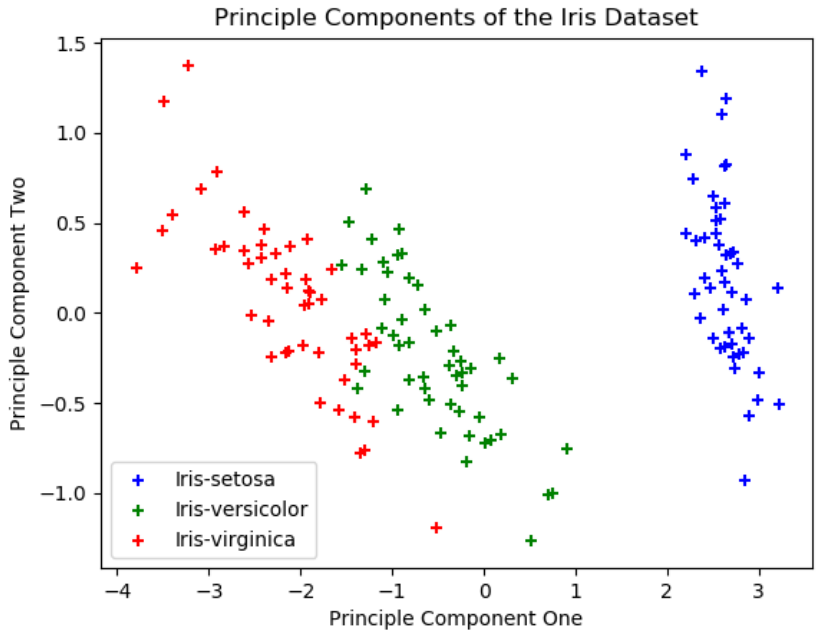
2.3 ./visuals/fig_2_4/campellcl

2.3.1 ./visuals/fig_2_4/campellcl/Fig_2_4.py

```
"""
Fig_2_4.py
Implementation of Programming Assignment Two for CS 5720.
"""

__author__ = "Chris Campell"
__version__ = "1/31/2018"

import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn import decomposition
from matplotlib import pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.lines as lines
import sys


def main():
    # Load file:
    with open(sys.argv[1], 'r') as fp:
        iris = pd.read_csv(fp, header=0)
    # Search for NANs:
    print(iris.describe())
    print(iris.head())
    print(iris.info())

    # Remove Y (target):
    Y = iris['class']
    X = np.delete(iris.values, 4, axis=1)

    # Resource:
↪   https://stats.stackexchange.com/questions/235882/pca-in-numpy-and-sklearn-produces-different-resu
    # Resource:
↪   http://sebastianraschka.com/Articles/2014_pca_step_by_step.html

    # Standardize:
    X_std = StandardScaler().fit_transform(X=X)

    # Add back target labels for vis.
    # Perform PCA:
    pca = decomposition.PCA(n_components=4)
    sklearn_pca = pca.fit_transform(X=iris.values[:, 0:4], y=None)
    # Multiply transformed data by -1 to mirror image:
    sklearn_pca[:,0] = sklearn_pca[:,0] * (-1)
    # sklearn_pca[:,1] = sklearn_pca[:,1] * (-1)
```

```
    # Plot Results:
    fig, ax = plt.subplots()
    # plt.scatter(x=sklearn_pca[:,0], y=sklearn_pca[:,1], marker=(4, 1,
↪   90))
    plt.scatter(x=sklearn_pca[0:50,0], y=sklearn_pca[0:50,1], marker='+',
↪   color='blue', label='Iris-setosa')
    plt.scatter(x=sklearn_pca[50:100,0], y=sklearn_pca[50:100,1],
↪   marker='+', color='green', label='Iris-versicolor')
    plt.scatter(x=sklearn_pca[100:,0], y=sklearn_pca[100:,1], marker='+',
↪   color='red', label='Iris-virginica')
    plt.title('Principle Components of the Iris Dataset')
    plt.xlabel('Principle Component One')
    plt.ylabel('Principle Component Two')
    plt.legend()
    plt.savefig(sys.argv[2])
    plt.show()


if __name__ == '__main__':
    main()
```

2.3.2 ./visuals/fig_2_4/campellcl/fig_2_4.png



2.4 ./visuals/fig_2_4/wascherb

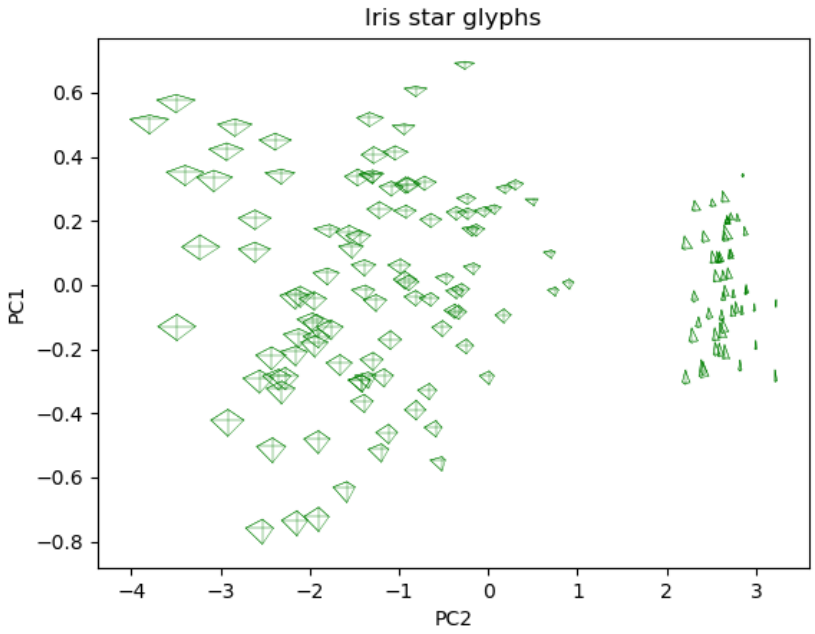
2.4.1 ./visuals/fig_2_4/wascherb/fig_2_4.py

```
import numpy as np
import matplotlib.pylab as plt
import sys

from sklearn import decomposition


def results(file_name):
    data = np.empty((0, 4), float)
    file = open(file_name, 'r')
    line_cnt = 0
    min_data = [100 for i in range(0, 4)]
    max_data = [-100 for i in range(0, 4)]
    list = []

    for line in file:
        if line_cnt == 0:
            line_cnt += 1
            continue
        l = line.split(',')
        b = [float(l[i]) for i in range(0, 4)]
        list.append(b)

        for i in range(0, 4):
            if b[i] < min_data[i]:
                min_data[i] = b[i]
            if b[i] > max_data[i]:
                max_data[i] = b[i]

        data = np.append(data, np.array([b]), axis=0)

    pca = decomposition.PCA(n_components=4)
    pca.fit(data)
    x = pca.transform(data)

    scale = [0.22, 0.05]
    # print(max_data)
    # print(min_data)
    for e in range(len(list)):
        for i in range(0, 4):
            list[e][i] = (list[e][i] - min_data[i]) / (max_data[i] -
↪  min_data[i]) * scale[i % 2]

        # print(list[e])

    plot(x, list)
```

```
def plot(x, data):
    plt.title('Iris star glyphs')

    d1 = []
    d2 = []
    for i in x:
        d1.append(i[0]*-1)
        d2.append(i[2]*-1)

    for i in range(0, len(d1)):
        drawStar(d1[i], d2[i], data[i])

    plt.xlabel('PC2')
    plt.ylabel('PC1')

    # plt.show()
    plt.savefig(sys.argv[2])
    return


def connectpoints(x, y, p1, p2):
    x1, x2 = x[p1], x[p2]
    y1, y2 = y[p1], y[p2]
    plt.plot([x1, x2], [y1, y2], color='green', linewidth=0.3)


def drawStar(px, py, data):
    lx = []
    ly = []

    # x changed y unchanged
    lx.append(px + data[0])
    ly.append(py)

    ly.append(py + data[1])
    lx.append(px)

    # x changed y unchanged
    lx.append(px - data[2])
    ly.append(py)

    ly.append(py - data[3])
    lx.append(px)

    plt.plot(lx, ly, color='green', linewidth=0.3)
    for i in range(0, len(lx)):
        for j in range(i + 1, len(lx)):
            connectpoints(lx, ly, i, j)
```

```
if __name__ == '__main__':
    if sys.argv == 1:
        print('Accept one argument: No input file for data')
        exit(0)

    results(sys.argv[1])
```

2.4.2 ./visuals/fig_2_4/wascherb/fig_2_4.png

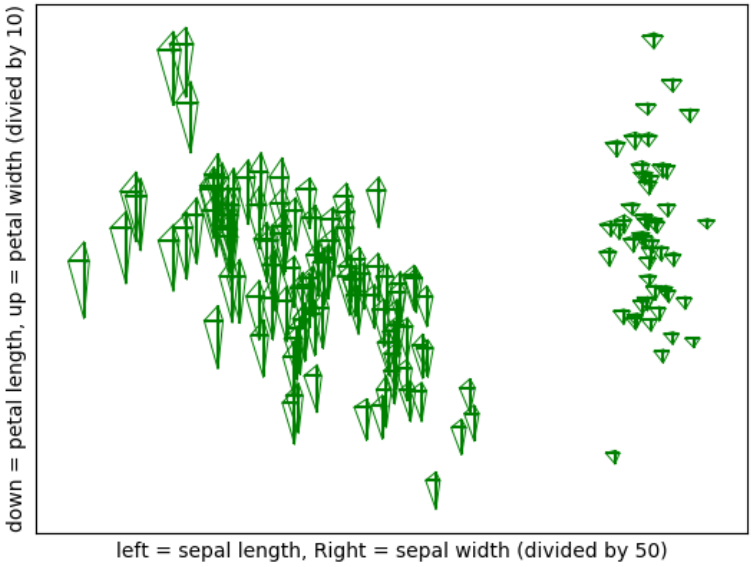


2.5 ./visuals/fig_2_4/stokesnl

2.5.1 ./visuals/fig_2_4/stokesnl/fig_2_4.py

```
#!/usr/bin/env python
import sys
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.mlab import PCA

data = pd.read_csv(sys.argv[1], usecols=["sepal length", "sepal width",
↪  "petal length", "petal width"])
results = np.array(data)
pca = PCA(results)
pca.Y[:,0] = pca.Y[:,0] * -1
pca.Y[:,1] = pca.Y[:,1] * -1
plt.scatter(pca.Y[:,0], pca.Y[:,1], s=1)
i = 0
while i < len(pca.Y[:,0]):
        x1 =[pca.Y[i,0] - (results[i,0]/50), pca.Y[i,0] +
↪  (results[i,1]/50)]
        y1 =[pca.Y[i,1] - (results[i,2]/10), pca.Y[i,1] +
↪  (results[i,3]/10)]
        plt.hlines(y=pca.Y[i,1], xmin=x1[0], xmax=x1[1], color='green')
        plt.vlines(x=pca.Y[i,0], ymin=y1[0], ymax=y1[1], color='green')
        plt.plot([x1[0], pca.Y[i,0]], [pca.Y[i,1], y1[0]], color='green',
↪  linewidth=.7)
        plt.plot([x1[0], pca.Y[i,0]], [pca.Y[i,1], y1[1]], color='green',
↪  linewidth=.7)
        plt.plot([x1[1], pca.Y[i,0]], [pca.Y[i,1], y1[0]], color='green',
↪  linewidth=.7)
        plt.plot([x1[1], pca.Y[i,0]], [pca.Y[i,1], y1[1]], color='green',
↪  linewidth=.7)
        #plt.plot(x1[0], pca.Y[i,1], pca.Y[i,0], y1[0])
        i+=1

#fig,ax = plt.subplots(1)

#ax.set_yticklabels([])
#ax.set_xticklabels([])
plt.xlabel("left = sepal length, Right = sepal width (divided by 50)")
plt.ylabel("down = petal length, up = petal width (divied by 10)")
plt.xticks([])
plt.yticks([])

plt.savefig(sys.argv[2])
```

2.5.2 ./visuals/fig_2_4/stokesnl/fig_2_4.png

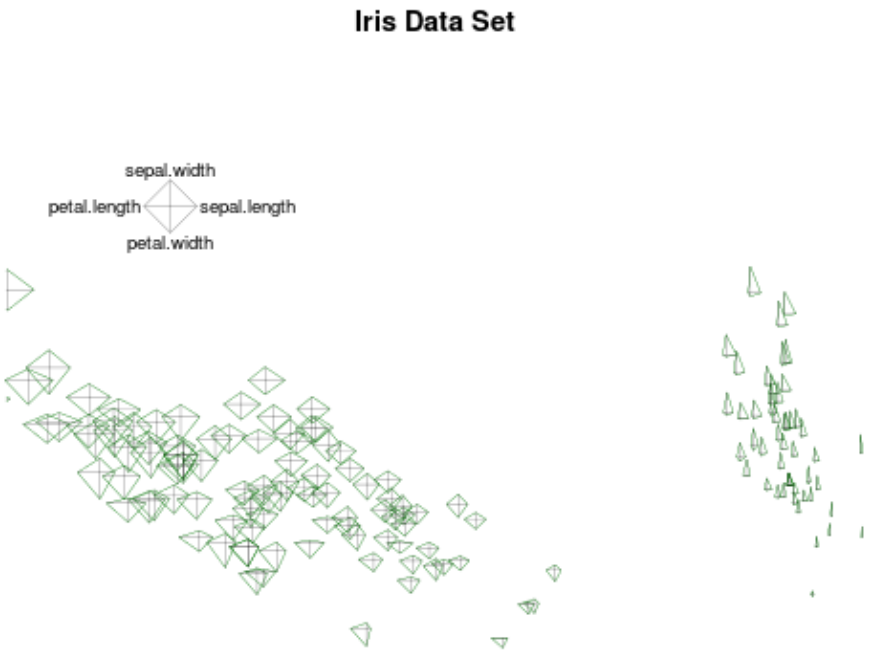


2.6 ./visuals/fig_2_4/davisjb2

2.6.1 ./visuals/fig_2_4/davisjb2/fig_2_4.R

```
require(grDevices)
require(stats)
library(ggplot2)
args <- commandArgs()
x <- read.table(args[6],header=T,sep=",")
PCA_data <- princomp(x[1:4],cor="False")
pc.comp <- PCA_data$scores
pc.comp1 <- -1*pc.comp[,1]
pc.comp2 <- -1*pc.comp[,2]
m <- matrix(c(pc.comp1,pc.comp2),ncol=2)
colors <- c(rep(3,150))
png(filename=args[7])
stars(x[, 1:4], locations=m,len=.2,xlim=c(-3,3),col.lines = colors,
↪  main="Iris Data Set", key.loc = c(-2,2))
dev.off()
```

2.6.2 ./visuals/fig_2_4/davisjb2/fig_2_4.png



2.7 ./visuals/fig_2_4/zhengn

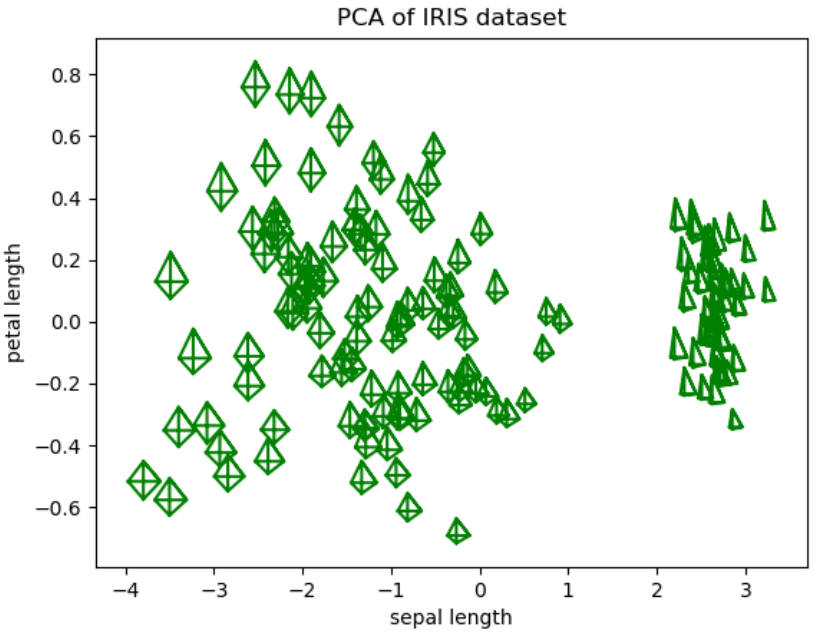
2.7.1 ./visuals/fig_2_4/zhengn/fig_2_4.py

```
__author__ = 'Naibin Zheng'
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import pandas as pd
from PIL import Image
import sys

def main():

    #iris = datasets.load_iris()
    iris = pd.read_csv(sys.argv[1])
    sl = iris['sepal length']/40
    sw = iris['sepal width']/40
    pl = iris['petal length']/40
    pw = iris['petal width']/40
    #print (sl)
    x = iris.as_matrix(columns=iris.columns[0:4])
    #print (x)

    m = x.mean(axis=0)
    #print (m)

    for i in range (len(x)):
        for j in range (len(x[i])):
            x[i][j] -= m[j]


    #print (x)


    pca = PCA(n_components=4)
    p = pca.fit_transform(x)
    #print (p)
    #print (pca.explained_variance_ratio_)
    #print (p[:, 0])
    fig =plt.figure()

    m = []
    n = []

    for x, y, sle, swi, ple, pwi in zip(p[:, 0], p[:, 2], sl, sw, pl,
↪   pw):

        #plt.scatter(-x, -y)
        #print (x)
```

```
    #print (y)
    x = - (x)
    #y = - (y)
    sle = -(sle)
    #swi = - (swi)
    ple = - (ple)
    #pwi = - (pwi)
    m.append(x+ple)
    m.append(x)
    m.append(x-sle)
    m.append(x)
    m.append(x+ple)
    m.append(x)
    m.append(x)
    m.append(x)
    m.append(x-sle)
    m.append(x)
    n.append(y)
    n.append(y)
    n.append(y)
    n.append(y+swi)
    n.append(y)
    n.append(y-pwi)
    n.append(y)
    n.append(y+swi)
    n.append(y)
    n.append(y-pwi)
    plt.plot(m, n, 'g')
    del m[:]
    del n[:]

#plt.legend()
plt.title('PCA of IRIS dataset')
plt.xlabel('sepal length')
plt.ylabel('petal length')


#a=[-1, -1]
#b=[-1, 1]
#c=[1, 1]
#d=[1, -1]
#x=[a[0], b[0], c[0], d[0], a[0]]
#y=[a[1], b[1], c[1], d[1], a[1]]
#plt.plot(x, y)


plt.show()
fig.savefig(sys.argv[2])
```

```
if __name__ == '__main__':
    main()
```

2.7.2 ./visuals/fig_2_4/zhengn/fig_2_4.png

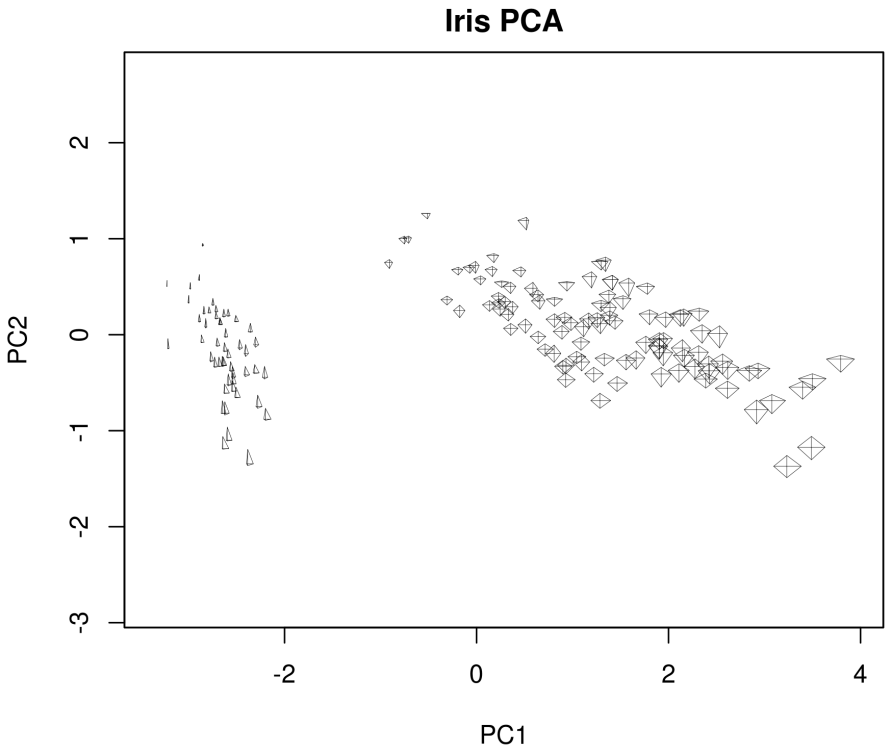


2.8 ./visuals/fig_2_4/smithkj2

2.8.1 ./visuals/fig_2_4/smithkj2/Program2.R

```
#Program2
t <- proc.time()

#iris <- datasets::iris
library(readr)
args <- commandArgs(T)
print(args)
name <- args[1]
iris <- read_csv(name)

iris_mat <- as.matrix(iris[,c(1:4)]/100)
pca <- prcomp(x = iris[,c(1:4)], 2, center=T)
comps <- pca$x



symbols(x=comps[,1],
        y=comps[,2],
        stars=(iris_mat[,c(1:4)]),
        inches=.15,
        fg='darkgreen',
        xlab = 'PC 1',
        ylab = 'PC 2',
        main='Iris PCA')

iris_loc = matrix(c(comps[,1],comps[,2]), ncol=2, nrow=150, byrow=F)

png(filename=args[2],
    width = 6,
    height = 5,
    units = 'in',
    res = 300)

stars(iris_mat[,c(1:4)],
      locations=iris_loc,
      len=.15,
      axes = T,
      xlab='PC1',
      ylab='PC2',
      main='Iris PCA')



print(t-proc.time())
```

2.8.2 ./visuals/fig_2_4/smithkj2/fig_2_4.png



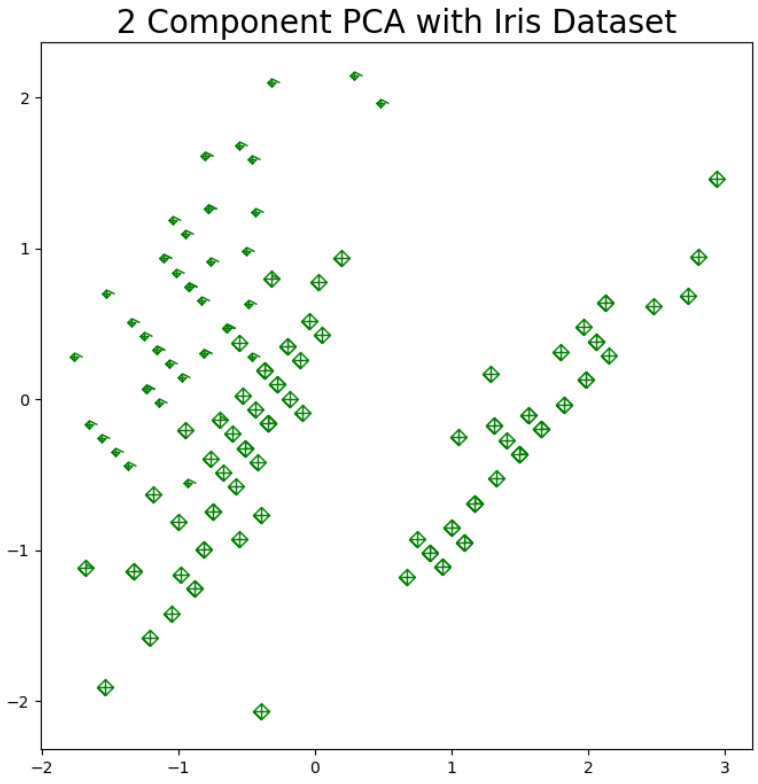
2.9 ./visuals/fig_2_4/carnsds

2.9.1 ./visuals/fig_2_4/carnsds/fig_2_4.py

```
#Dillon Carns
#2/4/2018
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
import numpy as np
import random as r
import sys

df = pd.read_csv(sys.argv[1],skiprows=[0], names=['sepal length','sepal
↪  width','petal length','petal width','class'])

#print(df['petal length'])

#standardize the data set
from sklearn.preprocessing import StandardScaler

features = ['sepal length', 'sepal width', 'petal length', 'petal width']

#separate out features
x = df.loc[:,['sepal width', 'petal width']].values

# Separating out the target
y = df.loc[:,['class']].values

# Standardizing the features
x = StandardScaler().fit_transform(x)

from sklearn.decomposition import PCA

pca = PCA(n_components=2)

principalComponents = pca.fit_transform(x)

principalDf = pd.DataFrame(data = principalComponents
             , columns = ['PC1', 'PC2'])

finalDf = pd.concat([principalDf, df[['class']]], axis = 1)

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
#ax.set_xlabel('Principal Component 1', fontsize = 15)
#ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 Component PCA with Iris Dataset', fontsize = 20)

classes = ['Iris-virginica', 'Iris-versicolor', 'Iris-setosa']
```

```
colors = 'g', 'g', 'g'
df.drop('class', axis=1, inplace=True)
for target, color in zip(classes,colors):
        indicesToKeep = finalDf['class'] == target
        verts = [[r.randint(1, 2), 0], [0,1], [-1, 0], [0, -1], [1, 0],
↪   [-1, 0], [0, -1], [0, 1]]
        #print(verts)
        plt.scatter(finalDf.loc[indicesToKeep,'PC1']
                ,finalDf.loc[indicesToKeep, 'PC2']
                 ,c = None, edgecolor=color, facecolor='w'
                 ,s = 100, verts=verts)
#plt.show()
plt.savefig(sys.argv[2])
```

2.9.2 ./visuals/fig_2_4/carnsds/fig_2_4.png



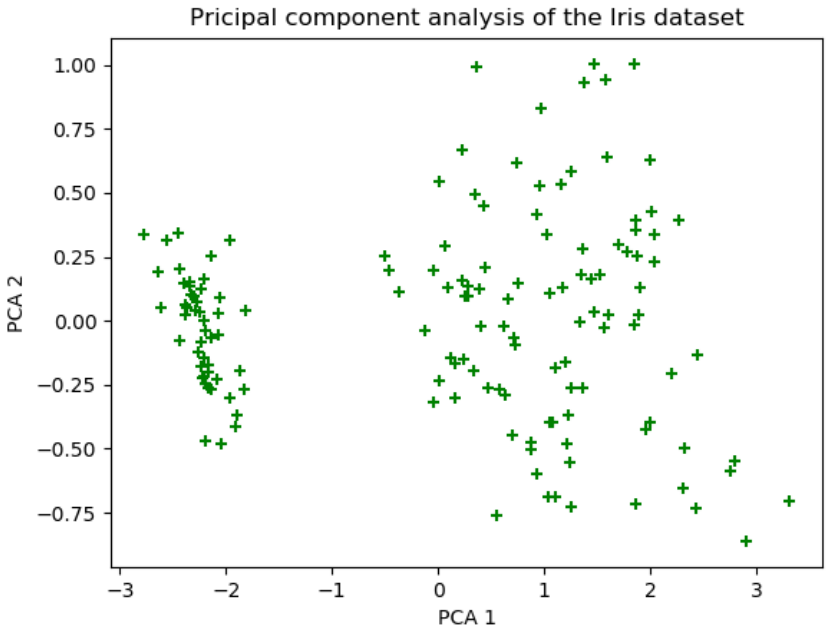
2.10 ./visuals/fig_2_4/halvorsenca

2.10.1 ./visuals/fig_2_4/halvorsenca/iris.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.path as mpath
import matplotlib.patches as mpatches
import sys


file = sys.argv[1]
data = pd.read_csv(file, header=None, sep=',')
#get the data of just the values not the species names
X = data.ix[:,0:3].values

#get the species names
y = data.ix[:,4:].values

#remove the first element which is a list of the header names
columns = X[0]
X = X[1:]

#create a standardized
X_std = StandardScaler().fit_transform(X)

x0 = []
x1 = []
x2 = []
x3 = []

for e in X:
    x0.append(float(e[0]))
    x1.append(float(e[1]))
    x2.append(float(e[2]))
    x3.append(float(e[3]))

cov_mat=np.cov(X_std.T)

eig_vals, eig_vecs = np.linalg.eig(cov_mat)

# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in
↪  range(len(eig_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort()
eig_pairs.reverse()
```

```
matrix_w = np.hstack((eig_pairs[0][1].reshape(4,1),
                      eig_pairs[2][1].reshape(4,1)))


Y = X_std.dot(matrix_w)

xs = []
ys = []
for i in Y:
    xs.append(i[0])
    ys.append(i[1])


# verticies = []
# for i in range(len(xs)):
#     vert = [
#         (xs[i], ys[i]),
#         (x0[i], 0),
#         (xs[i], ys[i]),
#         (0, x1[i]),
#         (xs[i], ys[i]),
#         (x2[i],0),
#         (xs[i], ys[i]),
#         (0,-x3[i]),
#     ]
#     verticies.append(vert)
#
# Path = mpath.Path
# codes = [Path.MOVETO, Path.LINETO, Path.MOVETO, Path.LINETO,
↪  Path.MOVETO, Path.LINETO, Path.MOVETO, Path.LINETO]
# path = mpath.Path(verticies[0], codes)
# patch = mpatches.PathPatch(path, facecolor='g')
# fig, ax = plt.subplots()
# ax.add_patch(patch)

plt.scatter(xs, ys, color='green',marker='+')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('Pricipal component analysis of the Iris dataset')
plt.savefig(sys.argv[2])
```

2.10.2 ./visuals/fig_2_4/halvorsenca/fig_2_4.png



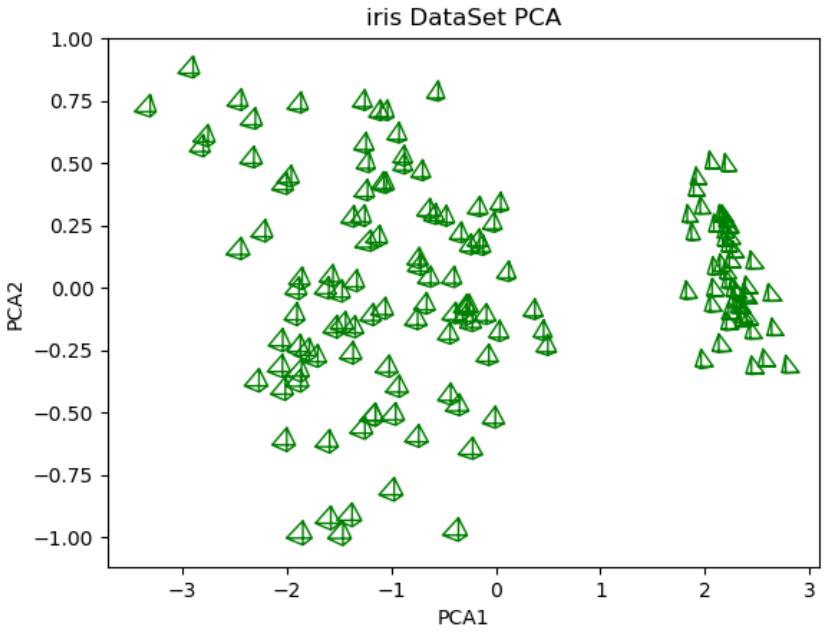
2.11 ./visuals/fig_2_4/laresaguilared

2.11.1 ./visuals/fig_2_4/laresaguilared/Plot.py

```
import csv
import matplotlib.patches as mpatches
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from matplotlib.patches import Polygon
import matplotlib.patches as patches
import numpy as np
import matplotlib.path as mpath
import matplotlib.patches as mpatches
import sys


def read_file():
   table_list = []
   with open( sys.argv[1]) as csvfile:
           readCSV = csv.reader(csvfile, delimiter=',')
           for row in readCSV:
               if len(row) > 0:
                   table_list.append(row)
   return table_list


def main():
   final = []
   table = read_file()
   table_clean = table[1:]
   table_clean = [i[0:4] for i in table_clean]
   for row in table_clean:
       final.append([float(i) for i in row])

   x = StandardScaler().fit_transform(final)
   pca = PCA(n_components=4)
   X_r = pca.fit_transform(x)


   for i in range(0, len(X_r)):
       X = X_r[i][0] * -1
       Y = X_r[i][2] * -1
       four = final[i][0]
       tree = final[i][1]
       two = final[i][2]
       one = final[i][3]
       Path = mpath.Path
       path_data = [
       (Path.MOVETO, (-two, 0)),
```

```
        (Path.LINETO, (0, four)),
        (Path.LINETO, (tree, 0)),
        (Path.LINETO, (0, -one)),
        (Path.LINETO, (-two, 0)),
        (Path.LINETO, (tree, 0)),
        (Path.MOVETO, (0, -one)),
        (Path.LINETO, (0, four)),
        ]
        codes, verts = zip(*path_data)
        markerp = mpath.Path(verts, codes)
        plt.scatter(X, Y, marker=markerp, s=300, facecolors='none',
↪   edgecolors='g')

        plt.xlabel("PCA1")
        plt.ylabel("PCA2")
        plt.title("iris DataSet PCA")




    plt.savefig(sys.argv[2])



if __name__ == '__main__':
    main()
```

2.11.2 ./visuals/fig_2_4/laresaguilared/fig_2_4.png

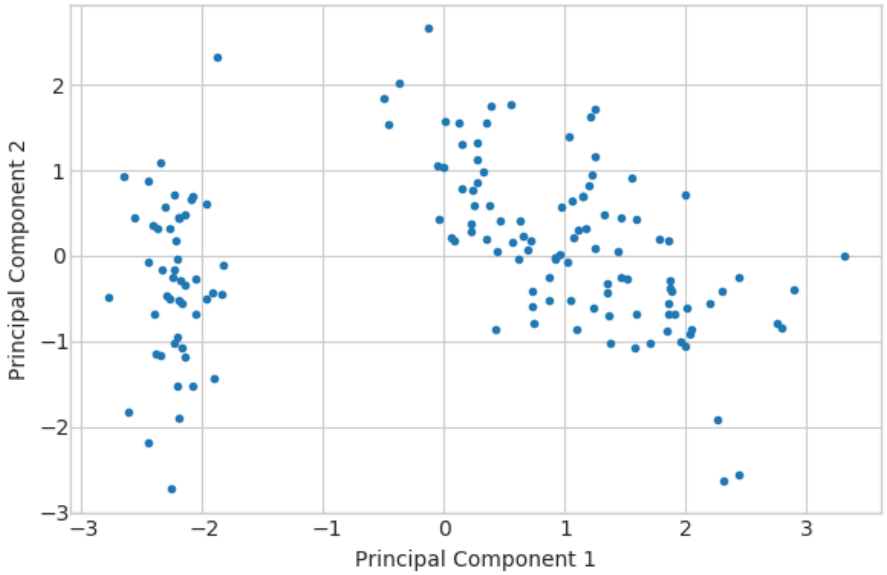


2.12 ./visuals/fig_2_4/beasonke

2.12.1 ./visuals/fig_2_4/beasonke/fig_2_4.py

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot as plt
# import bokeh.plotting, bokeh.models
# bokeh.plotting.output_notebook()
import sys

df= pd.read_csv(sys.argv[1])
# df = df[["sepal length", "sepal width", "petal length", "petal
 ↪  width"]]
x = df.ix[:,0:4].values
y = df.ix[:,4].values
X_std = StandardScaler().fit_transform(x)
c = np.cov(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(c)

#create and sort eigenvalue, eigenvector pairs
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in
 ↪  range(len(eig_vals))]
eig_pairs.sort(key=lambda x: x[0], reverse=True)
eig_mat = np.column_stack((eig_pairs[0][1], eig_pairs[1][1]))
# print(eig_mat)
Y = X_std.dot(eig_mat)

with plt.style.context('seaborn-whitegrid'):
    x = Y[:,0]
    y = Y[:,1]
    plt.figure(figsize=(6, 4))
    plt.scatter(x, y, marker='.')

    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.legend(loc='lower center')
    plt.tight_layout()
    plt.plot()
    #plt.show()
plt.savefig(sys.argv[2])
```

2.12.2 ./visuals/fig_2_4/beasonke/fig_2_4.png

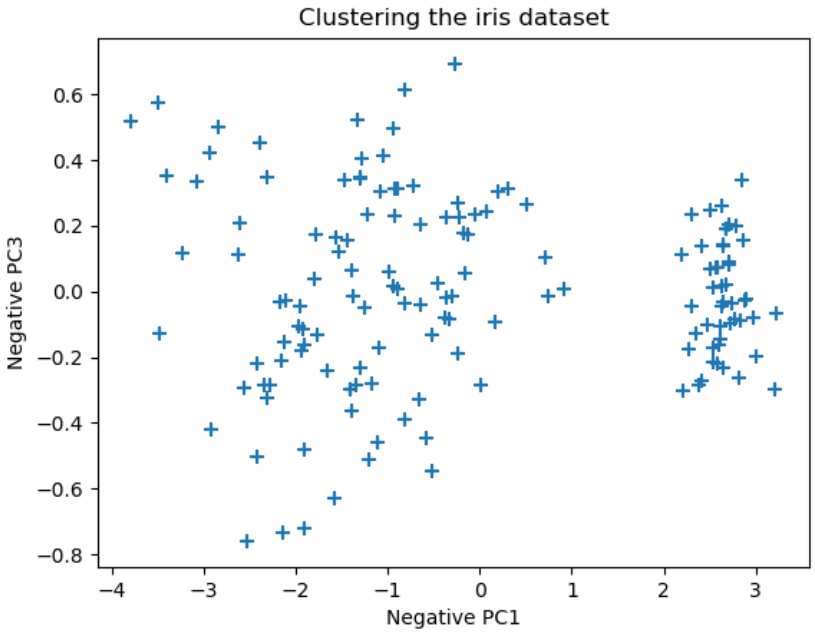


2.13 ./visuals/fig_2_4/beekmanpc

2.13.1 ./visuals/fig_2_4/beekmanpc/p02.py

```
import sys
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import normalize
import pandas as pd
import numpy as np

def main():
    iris = pd.read_csv(sys.argv[1], header=0)
    X = iris.loc[:,:'petal width']
    X_norm = normalize(X)
    Y = iris.loc[:,'class']

    pca = PCA(n_components=4)
    pca.fit(X)
    X = pca.transform(X)

    plt.scatter(-X[:,0], -X[:,2], marker=(4,2,0))
    plt.xlabel("Negative PC1")
    plt.ylabel("Negative PC3")
    plt.title("Clustering the iris dataset")
    plt.savefig(sys.argv[2])
    #plt.show()


if __name__ == "__main__":
    main()
```

2.13.2 ./visuals/fig_2_4/beekmanpc/fig_2_4.png

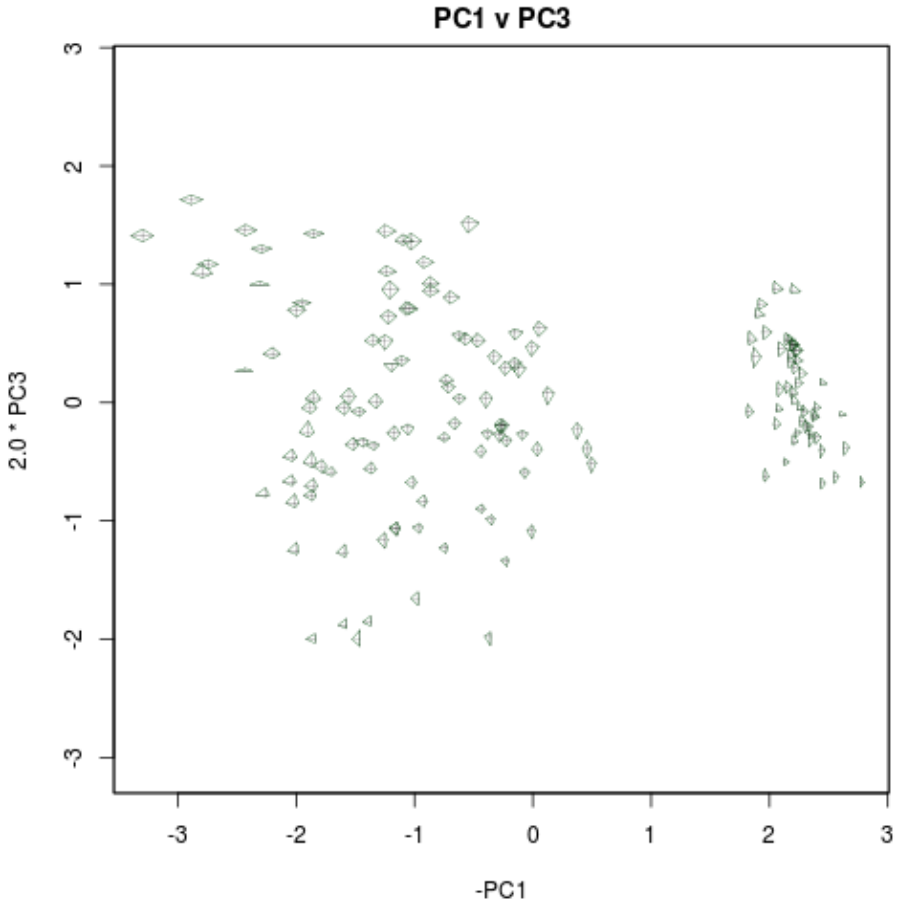


2.14 ./visuals/fig_2_4/emeryde

2.14.1 ./visuals/fig_2_4/emeryde/fig_2_4.R

```
args = commandArgs(trailingOnly=TRUE)
df <- read.csv(args[1])
##Iris is pre loaded in R

ir.pca <- prcomp(x = df[,c(1:4)], center = T, scale = T)
df <- as.data.frame(ir.pca$x)

png(args[2])

stars(df,
      len = -.1,
      locations = matrix(c(-df$PC1, 2.0*df$PC3), ncol = 2, nrow = 150,
 ↪  byrow = F),
      xlab = '-PC1',
      ylab = '2.0 * PC3',
      main = 'PC1 v PC3',
      axes = T,
      col.lines = rep('#1DA42F',150))

dev.off()

##No clue why my stars are rotated, have tried just about every type of
 ↪  PCA
##manipulation but it doesn't seem to compute it the same way the book
 ↪  does.
##Had to just set the len parameter to a negative, which worked.
##Also, the base graphics don't allow me to change the cross color.
```

2.14.2 ./visuals/fig_2_4/emeryde/fig_2_4.png



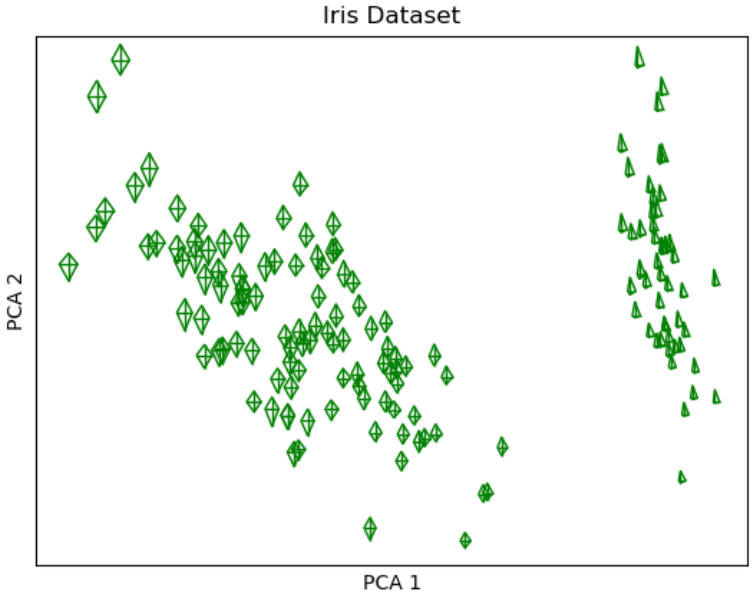
2.15 ./visuals/fig_2_4/parkerat2

2.15.1 ./visuals/fig_2_4/parkerat2/fig_2_4.py

```
import pandas as pd
import sys
from matplotlib.patches import Polygon
from matplotlib.collections import PatchCollection
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

args = sys.argv
df = pd.read_csv(args[1])
# df = pd.read_csv("iris.csv")
headers = list(df)
X = df.as_matrix(columns=headers[0:-1])

pca = PCA(n_components=2)
Xpca = pca.fit_transform(X)

x = [i[0] for i in Xpca]
y = [i[1] for i in Xpca]

Xn = X / X.max(0)
patches = []
for i in range(len(x)):
    poly = Polygon([[x[i] + Xn[i][2] / 10, y[i]], [x[i], y[i] + Xn[i][1]
↪  / 10], [x[i] - Xn[i][0] / 10, y[i]], [x[i], y[i] - Xn[i][3] / 10]])
    horz = Polygon([[x[i] + Xn[i][2] / 10, y[i]], [x[i] - Xn[i][0] / 10,
↪  y[i]]])
    vert = Polygon([[x[i], y[i] + Xn[i][1] / 10], [x[i], y[i] - Xn[i][3]
↪  / 10]])
    patches.append(poly)
    patches.append(horz)
    patches.append(vert)
    plt.scatter(x[i], y[i], s=1/2, marker='+', color='g')
p = PatchCollection(patches)
p.set_color('g')
p.set_facecolor('none')
ax = plt.gca()
ax.add_collection(p)
ax.invert_xaxis()
# ax.invert_yaxis()
plt.xticks([])
plt.yticks([])
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('Iris Dataset')

# plt.show()
```

```
plt.savefig(args[2])
plt.clf()
```

2.15.2 ./visuals/fig_2_4/parkerat2/fig_2_4.png



Chapter 3

./visuals/music_som

3.1 Description

```
# Self-organizing Map of Major and Minor Triads

"Music and Schema Theory"
by Marc Leman
Section 5.3: SAM: A Simple Model
Section 6.3-6.6: SOM: The Self-Organizing Map
Section 7.1: SAMSOM

Data created following the process in Table 5.1 for major
and minor triads, in this repository:
[../../data/chords.csv]
```