""" Cody Valle, Andrew Burke, and Katreina Carpenter

CPSC 310

hw1.py """

from collections import defaultdict

import matplotlib.pylab as plt

import numpy as np

""" Reads a csv file and returns a table (list of lists) """

def read\_csv(filename):

import csv

the\_file = open(filename, 'r')

the\_reader = csv.reader(the\_file, dialect='excel')

table = []

for row in the\_reader:

if len(row) > 0:

table.append(row)

the\_file.close()

return table

""" Makes floats """

def get\_column\_as\_floats(table, index):

column = []

for row in table:

if row[index] != 'NA':

column.append(float(row[index]))

return column

""" Helper function for step1 """

def doPlot1(column, strings, labels = None):

dict = {} # Initialize a dictionary

for i in column: # Count the number of cars with each number of cylinders

dict[i] = dict.get(i, 0) + 1 # Increase the count or create and increase

xs = [] # Will hold the x values

ys = [] # Will hold the y vlaues

for key, val in dict.items(): # Iterate through, and create values for the plot

xs.append(key) # Keys are x values

ys.append(val) # Items are y values

plt.figure() # Reset the figure

plt.suptitle(strings['title'], fontsize=12, fontweight='bold') # Title for the graph we are making

plt.grid(True) # Turn on the grid

plt.bar(xs, ys, 1/1.5, color='blue', align='center') # Plot the data

if labels == None: # We are generating our own labels

labels = [int(i) for i in xs] # Prepare the tick labels

elif labels == 'append': # We are generating our own labels

labels = ['19' + str(int(i)) for i in xs]

plt.xticks(xs, labels) # Set the tick labels

plt.xlim(min(xs) - 1, max(xs) + 1) # Set the x limits

plt.ylim(0, max(ys) \* 1.05) # Set the y limits

plt.xlabel(strings['xlabel']) # Set the x axis label

plt.ylabel(strings['ylabel']) # Set the y axis label

plt.savefig('step-1-' + strings['filename'] + '.pdf') # Save the figure

if show\_graphs: # Show the graph if preferred

plt.show()

plt.close() # Clean up

""" Performs step 1 of the homework """

def step1(table):

column = get\_column\_as\_floats(table, 1) # Get the cylinders

strings = {'title' : 'Number of Cars versus Number of Cylinders',

'xlabel' : 'Number of Cylinders',

'ylabel' : 'Number of Cars',

'filename' : 'cylinders'}

doPlot1(column, strings) # Plot the cylinders

column = get\_column\_as\_floats(table, 6) # Get the model year

strings = {'title' : 'Number of Cars versus Model Year',

'xlabel' : 'Model Year',

'ylabel' : 'Number of Cars',

'filename' : 'model\_year'}

doPlot1(column, strings, labels='append') # Plot the model year

column = get\_column\_as\_floats(table, 7) # Get the origin

strings = {'title' : 'Number of Cars versus Origin',

'xlabel' : 'Origin',

'ylabel' : 'Number of Cars',

'filename' : 'origin'}

doPlot1(column, strings, labels=['USA','EUR','JAP']) # Plot the origin

""" Step 2: Creates a pie chart showing the frequency of cars for each

of the categorical attributes of the auto-data.txt dataset. """

def step2(table):

"""Origin Chart"""

# Reset the figure

fig = plt.figure()

#cars

xs = ['US Cars', 'EU Cars', 'JP Cars']

ys = [0] \* 3

#creating frequencies

for row in table:

if row[7] == '1':

ys[0] += 1

if row[7] == '2':

ys[1] += 1

if row[7] == '3':

ys[2] += 1

#make chart

plt.pie(ys, labels=xs, autopct='%1.1f%%')

plt.axis('equal')

plt.suptitle('Origin of Cars', fontsize=12, fontweight='bold')

if show\_graphs: # Show the graph if preferred

plt.show()

fig.savefig('step-2-origins.pdf')

plt.close()

"""Cylinders Chart"""

# Reset the figure

fig = plt.figure()

#cars

xs = ['3', '4', '5', '6', '8']

ys = [0,0,0,0,0]

#creating frequencies

for row in table:

if row[1] == '3':

ys[0] += 1

if row[1] == '4':

ys[1] += 1

if row[1] == '5':

ys[2] += 1

if row[1] == '6':

ys[3] += 1

if row[1] == '8':

ys[4] += 1

#make chart

plt.pie(ys, labels=xs, autopct='%1.1f%%')

plt.axis('equal')

plt.suptitle('Cars Per Cylinders', fontsize=12, fontweight='bold')

if show\_graphs: # Show the graph if preferred

plt.show()

fig.savefig('step-2-cylinders.pdf')

plt.close()

"""Model Year Chart"""

# Reset the figure

fig = plt.figure()

#cars

xs = list(range(70, 80))

ys = [0] \* len(xs)

#creating frequencies

for row in table:

if row[6] == '70':

ys[0] += 1

if row[6] == '71':

ys[1] += 1

if row[6] == '72':

ys[2] += 1

if row[6] == '73':

ys[3] += 1

if row[6] == '74':

ys[4] += 1

if row[6] == '75':

ys[5] += 1

if row[6] == '76':

ys[6] += 1

if row[6] == '77':

ys[7] += 1

if row[6] == '78':

ys[8] += 1

if row[6] == '79':

ys[9] += 1

#make chart

labels = ['19' + str(x) for x in xs]

plt.pie(ys, labels=labels, colors = ['blue', 'green', 'red', 'yellow', 'white'], autopct='%1.1f%%')

plt.axis('equal')

plt.suptitle('Cars Produced Per Year', fontsize=12, fontweight='bold')

if show\_graphs: # Show the graph if preferred

plt.show()

fig.savefig('step-2-modelYear.pdf')

plt.close()

def help3(xs, name):

fig = plt.figure() # Reset figure

#create dot chart

plt.plot(xs, [1] \* len(xs), 'b.', alpha = 0.2, markersize = 16)

#get rid of y-axis

plt.gca().get\_yaxis().set\_visible(False)

#label graph

plt.xlabel(name)

plt.title(name + " Frequency")

#save result to a pdf file

fig.savefig('step-3-' + name + '.pdf')

if show\_graphs: # Show the graph if preferred

plt.show()

plt.close()

""" Step 3: Create a dot chart showing the values for each of the continuous attributes. """

def step3(table):

"""MPG Chart"""

xs = get\_column\_as\_floats(table, 0)

help3(xs, 'MPG')

"""Displacement Chart"""

xs = get\_column\_as\_floats(table, 2)

help3(xs, 'Displacement')

"""Horsepower Chart"""

xs = get\_column\_as\_floats(table, 3)

help3(xs, 'Horsepower')

"""Weight Chart"""

xs = get\_column\_as\_floats(table, 4)

help3(xs, 'Weight')

"""Acceleration"""

xs = get\_column\_as\_floats(table, 5)

help3(xs, 'Acceleration')

"""MSRP Chart"""

xs = get\_column\_as\_floats(table, 9)

help3(xs, 'MSRP')

""" Performs step 4-1 of the homework """

def step4\_1(table):

mpgs = get\_column\_as\_floats(table, 0) # Get mpg values

ratings = [0] \* 10 # Initialize list to count cars for each rating

maxes = [13, 14, 16, 19, 23, 26, 30, 36, 44]

for value in mpgs:

found = False

for i in range(len(maxes)):

if value <= maxes[i]:

ratings[i] += 1

found = True

break

if not found:

ratings[-1] += 1

# Set up conditions for the graph

labels = ["<=13", "14", "15-16", "17-19", "20-23", "24-26", "27-30", "31-36", "37-44", ">=45"]

xs = range(10)

ys = ratings

plt.figure()

plt.suptitle("Number of Cars by MPG", fontsize=12, fontweight='bold')

plt.grid(True)

plt.bar(xs, ys, 1/1.5, color='blue', align='center')

plt.xticks(xs, labels)

plt.xlim(min(xs) - 1, max(xs) + 1)

plt.ylim(0, max(ys) + 10)

plt.xlabel("MPG")

plt.ylabel("Count")

plt.savefig("step-4-ratings.pdf") # Save the figure

if show\_graphs: # Show the graph if preferred

plt.show()

plt.close() # Clean up

""" Performs step 4-2 of the homework """

def step4\_2(table):

mpgs = get\_column\_as\_floats(table, 0) # Get MPG values

ratings = [0] \* 5 # Create list for the 5 bins (ranges)

maxes = [17, 23, 29, 35]

for value in mpgs:

found = False

for i in range(len(maxes)):

if value <= maxes[i]:

ratings[i] += 1

found = True

break

if not found:

ratings[-1] += 1

# Set up conditions for the graph

labels = ["<=17", "18-23", "24-29", "30-35", ">=36"]

xs = range(5)

ys = ratings

plt.figure()

plt.suptitle("Number of Cars by Equal Width Rankings of MPG", fontsize=12, fontweight='bold')

plt.grid(True)

plt.bar(xs, ys, 1/1.5, color='blue', align='center')

plt.xticks(xs, labels)

plt.xlim(min(xs) - 1, max(xs) + 1)

plt.ylim(0, max(ys) + 10)

plt.xlabel("MPG")

plt.ylabel("Count")

plt.savefig("step-4-equal.pdf") # Save the figure

if show\_graphs: # Show the graph if preferred

plt.show()

plt.close() # Clean up

""" Helper function for step5 """

def doPlot5(column, strings):

plt.figure() # Reset the figure

plt.suptitle(strings['title'], fontsize=12, fontweight='bold') # Title for the graph we are making

(n, bins, pathces) = plt.hist(column, color='green', align='right') # Plot the data

#binwidth = bins[1] - bins[0]

#plt.xticks(bins - binwidth / 2.0)

plt.xlabel(strings['xlabel']) # Set the x axis label

plt.ylabel(strings['ylabel']) # Set the y axis label

plt.savefig('step-5-' + strings['filename'] + '.pdf') # Save the figure

if show\_graphs: # Show the graph if preferred

plt.show()

plt.close() # Clean up

""" Performs step 5 of the homework """

def step5(table):

column = get\_column\_as\_floats(table, 0) # Get the MPG attribute

strings = {'title' : 'Number of Cars versus Miles Per Gallon',

'xlabel' : 'Miles Per Gallon',

'ylabel' : 'Number of Cars',

'filename' : 'mpg'}

doPlot5(column, strings)

column = get\_column\_as\_floats(table, 2) # Get the displacement attribute

strings = {'title' : 'Number of Cars versus Displacement',

'xlabel' : 'Displacement',

'ylabel' : 'Number of Cars',

'filename' : 'disp'}

doPlot5(column, strings)

column = get\_column\_as\_floats(table, 3) # Get the horsepower attribute

strings = {'title' : 'Number of Cars versus Horsepower',

'xlabel' : 'Horsepower',

'ylabel' : 'Number of Cars',

'filename' : 'hp'}

doPlot5(column, strings)

column = get\_column\_as\_floats(table, 4) # Get the weight attribute

strings = {'title' : 'Number of Cars versus Weight',

'xlabel' : 'Weight',

'ylabel' : 'Number of Cars',

'filename' : 'weight'}

doPlot5(column, strings)

column = get\_column\_as\_floats(table, 5) # Get the acceleration attribute

strings = {'title' : 'Number of Cars versus Acceleration',

'xlabel' : 'Acceleration',

'ylabel' : 'Number of Cars',

'filename' : 'accel'}

doPlot5(column, strings)

column = get\_column\_as\_floats(table, 9) # Get the MSRP attribute

strings = {'title' : 'Number of Cars versus MSRP',

'xlabel' : 'MSRP',

'ylabel' : 'Number of Cars',

'filename' : 'msrp'}

doPlot5(column, strings)

""" Gets points from the table """

def points(table, xindex, yindex):

xs = []

ys = []

for row in table:

if row[xindex] == 'NA' or row[yindex] == 'NA':

continue

xs.append(row[xindex])

ys.append(row[yindex])

xs = [float(x) for x in xs]

ys = [float(y) for y in ys]

return (xs,ys)

""" Reusable function for step 6 """

def help6(table, index, attribute):

(xs, ys) = points(table, index, 0) # y values will always be mpg

plt.figure()

plt.suptitle("MPG vs" + attribute, fontsize=12, fontweight='bold')

plt.xlabel(attribute)

plt.ylabel("MPG")

plt.grid(True)

plt.xlim(0, max(xs) + 10)

plt.ylim(0, max(ys) + 10)

plt.plot(xs, ys, 'b.')

if show\_graphs: # Show the graph if preferred

plt.show()

plt.savefig("step-6-" + attribute + ".pdf")

plt.close()

""" Performs step 6 of the homework """

def step6(table):

help6(table, 2, "Displacement")

help6(table, 3, "Horsepower")

help6(table, 4, "Weight")

help6(table, 5, "Acceleration")

help6(table, 9, "MSRP")

""" Gets the slope and intercept of the group of points """

def slope\_intercept(xs, ys):

ybar = sum(ys) / float(len(ys)) # average y value

xbar = sum(xs) / float(len(xs)) # average x value

nume = sum([(xs[i] - xbar)\*(ys[i] - ybar) for i in range(len(xs))])

denom = sum([(xs[i] - xbar)\*\*2 for i in range(len(xs))])

m = nume / denom

b = ybar - m\*xbar

return (m, b)

""" Helps step7 """

def help7(table, index, name):

(xs, ys) = points(table, index, 0)

(m, b) = slope\_intercept(xs, ys)

# Graph it

plt.figure()

plt.suptitle("MPG vs " + name, fontsize=12, fontweight='bold')

plt.xlabel(name)

plt.ylabel("MPG")

plt.plot(xs, ys, '.') # Plot the scatter plot

lxs = range(int(min(xs)), int(max(xs)) + 1)

lys = [x \* m + b for x in lxs]

plt.plot(lxs, lys, '-')

plt.savefig("step-7-" + name + ".pdf")

if show\_graphs: # Show the graph if preferred

plt.show()

plt.close()

""" Performs step 7 of the homework """

def step7(table):

help7(table, 2, 'Displacement')

help7(table, 3, 'Horsepower')

help7(table, 4, 'Weight')

help7(table, 9, 'MSRP')

""" Performs step 8 of the homework """

def step8(table):

table.sort(key=lambda x: x[6]) # Sort on model year

""" Draw the first chart, a box plot of the

MPG of cars produced by year. """

data = [] # List of list of data points

years = [] # Years that align with the data table

element = [] # Will hold the data points, then be appnded to the data table

cur\_year = table[0][6] # The current year we are counting for.

for row in table:

if row[0] == 'NA' or row[6] == 'NA': # The data must exist or we cannot use this row

continue

if row[6] != cur\_year: # Check if we are in a new year

data.append(element) # Add the found data points to the data table

element = [] # Clear the data points

years.append(cur\_year) # Add the old year to the labels

cur\_year = row[6] # Keep track of the current year

element.append(float(row[0])) # Add MPG data

years.append(cur\_year) # Add the final year as a label

fig = plt.figure() # Reset the figure

plt.suptitle('Miles Per Gallon vs Model Year', fontsize=12, fontweight='bold') # Title for the graph we are making

ax = fig.add\_subplot(111) # We are plotting multiple things

bp = ax.boxplot(data) # Add the data as a box plot

# Modify the colors and styles

for box in bp['boxes']:

box.set(color='blue', linewidth=2)

for whisker in bp['whiskers']:

whisker.set(color='green', linewidth=2, linestyle='solid')

for cap in bp['caps']:

cap.set(color='blue', linewidth=2)

for median in bp['medians']:

median.set(color='red', linewidth=2)

for flier in bp['fliers']:

flier.set(marker='o', color='blue')

labels = ['19' + year for year in years] # '19 in front of every label for a four digit year

ticklabels = ax.set\_xticklabels(labels) # Set the tick labels

plt.setp(ticklabels, rotation=35) # Rotate the labels a little for readability

plt.xlabel('Model Year') # Set x axis label

plt.ylabel('Miles Per Gallon') # Set y axis label

plt.subplots\_adjust(bottom=0.2) # Adjust the spacing to prevent clipping of labels

fig.savefig('step-8-chart1.pdf') # Save the figure

if show\_graphs: # Show the graph if preferred

plt.show()

plt.close() # Clean up

""" Draw the second chart, a multi-frequency graph of

the number of cars produced by year and country. """

years = [] # Will hold the year labels that align with the following data lists

USAdata = [0] # Will hold the number of cars produced by USA each year

EURdata = [0] # " " " ... ... EUR " "

JAPdata = [0] # " " " ... ... JAP " "

cur\_year = table[0][6] # The current year we are counting for. Note that the table is still sorted by year

for row in table:

if row[7] == 'NA' or row[6] == 'NA': # The data must exist or we cannot use this row

continue

if row[6] != cur\_year: # Check if we are in a new year

USAdata.append(0) # Add a new element to increment

EURdata.append(0) # ...for each dataset

JAPdata.append(0)

years.append(cur\_year) # Add the old year as a label

cur\_year = row[6] # Update to the new current year

if row[7] == '1': # Then increment the USA count

USAdata[len(years)] += 1

continue

if row[7] == '2': # Then increment the EUR count

EURdata[len(years)] += 1

continue

if row[7] == '3': # Then increment the JAP count

JAPdata[len(years)] += 1

years.append(cur\_year) # Add the final year as a label

fig = plt.figure() # Reset the figure

plt.suptitle('Origin of Cars by Year', fontsize=12, fontweight='bold') # Title for the graph we are making

ax = fig.add\_subplot(111) # We are plotting multiple things

xdist = plt.arange(len(years)) # Distance between the xticks

width = 0.25 # The width of each bar per xtick. We have three so there will be whitespace between the groupings

USAbars = ax.bar(xdist, USAdata, width, color = 'blue') # Blue USA bars

EURbars = ax.bar(xdist + width, EURdata, width, color = 'green') # Green EUR bars

JAPbars = ax.bar(xdist + width \* 2, JAPdata, 0.3, color = 'red') # Red JAP bars

ax.set\_xticks(xdist + width) # Set the tick label locations

labels = ['19' + year for year in years] # '19 in front of every label for a four digit year

ticklabels = ax.set\_xticklabels(labels) # Set the tick labels

plt.setp(ticklabels, rotation=25) # Rotate the labels a little for readability

ax.legend((USAbars[0],EURbars[0],JAPbars[0]), ('USA','EUR','JAP')) # Create the legend

plt.xlabel('Model Year') # Set x axis label

plt.ylabel('Number of Cars Produced') # Set y axis label

plt.subplots\_adjust(bottom=0.2) # Adjust the spacing to prevent clipping of labels

fig.savefig('step-8-chart2.pdf') # Save the figure

if show\_graphs: # Show the graph if preferred

plt.show()

plt.close() # Clean up

""" The main function """

def main():

global show\_graphs

show\_graphs = False

table = read\_csv('auto-data.txt') # Read in the automotive data

step1(table) # Perform step 1

step2(table)

step3(table)

step4\_1(table)

step4\_2(table)

step5(table) # Perform step 5

step6(table)

step7(table)

step8(table) # Perform step 8

""" Entry point """

if \_\_name\_\_ == '\_\_main\_\_':

main()