**02-VBA-Scripting - Session 1 - Activity - 01-Ins\_HelloWorld a month ago**

**02-VBA-Scripting - Session 1 - Activity - 02-Stu\_HelloVBA a month ago**

**02-VBA-Scripting - Session 1 - Activity - 03-Ins\_ButtonClicks a month ago**

**02-VBA-Scripting - Session 1 - Activity - 04-Stu\_ChooseYourButton a month ago**

**02-VBA-Scripting - Session 1 - Activity - 05-Ins\_CellsAndRanges a month ago**

**02-VBA-Scripting - Session 1 - Activity - 06-Stu\_ChessBoard a month ago**

**02-VBA-Scripting - Session 1 - Activity - 07-Ins\_Variables a month ago**

**02-VBA-Scripting - Session 1 - Activity - 08-Stu\_TotalCalculator a month ago**

**02-VBA-Scripting - Session 1 - Activity - 09-Ins\_Arrays a month ago**

**02-VBA-Scripting - Session 1 - Activity - 10-Ins\_Splitting a month ago**

**02-VBA-Scripting - Session 1 - Activity - 11-Stu\_SentenceBreaker a month ago**

**02-VBA-Scripting - Session 1 - Activity - 12-Ins\_Conditionals a month ago**

**02-VBA-Scripting - Session 1 - Activity - 13-Stu\_ChooseYourStory**

**02-VBA-Scripting - Session 2 - Activity - 01-Stu\_Warmup a month ago**

**02-VBA-Scripting - Session 2 - Activity - 02-Ins\_ForLoops a month ago**

**02-VBA-Scripting - Session 2 - Activity - 03-Stu\_ChickenNuggets a month ago**

**02-VBA-Scripting - Session 2 - Activity - 04-Ins\_LoopConditionals a month ago**

**02-VBA-Scripting - Session 2 - Activity - 05-Stu\_FizzBuzz a month ago**

**02-VBA-Scripting - Session 2 - Activity - 06-Stu\_Lotto a month ago**

**02-VBA-Scripting - Session 2 - Activity - 07-Ins\_NestedForLoops a month ago**

**02-VBA-Scripting - Session 2 - Activity - 08-Stu\_HornetsNest**

**02-VBA-Scripting - Session 3 - Activity - 01-Stu\_StarsCounter 24 days ago**

**02-VBA-Scripting - Session 3 - Activity - 02-Ins\_Formatter 24 days ago**

**02-VBA-Scripting - Session 3 - Activity - 03-Stu\_Gradebook 24 days ago**

**02-VBA-Scripting - Session 3 - Activity - 04-Stu\_Checkerboard 24 days ago**

**02-VBA-Scripting - Session 3 - Activity - 05-Ins\_NextCells 24 days ago**

**02-VBA-Scripting - Session 3 - Activity - 06-Stu\_CreditCardChecker 24 days ago**

**02-VBA-Scripting - Session 3 - Activity - 07-Stu\_WellsFargo\_Pt1 24 days ago**

**02-VBA-Scripting - Session 3 - Activity - 08-Stu\_WellsFargo\_Pt2**

**03-Python - Session 1 - Activity - 01-Ins\_Terminal 17 days ago**

**03-Python - Session 1 - Activity - 02-Stu\_TerminalTest 17 days ago**

**03-Python - Session 1 - Activity - 03-Ins\_Variables 17 days ago**

**03-Python - Session 1 - Activity - 04-Stu\_HelloVariableWorld 17 days ago**

**03-Python - Session 1 - Activity - 05-Ins\_Prompts 17 days ago**

**03-Python - Session 1 - Activity - 06-Stu\_DownToInput 17 days ago**

**03-Python - Session 1 - Activity - 07-Ins\_Conditionals 17 days ago**

**03-Python - Session 1 - Activity - 08-Stu\_ConditionalConundrum 17 days ago**

**03-Python - Session 1 - Activity - 09-Ins\_List 17 days ago**

**03-Python - Session 1 - Activity - 10-Stu\_RockPaperScissors 17 days ago**

**03-Python - Session 1 - Activity - 11-Ins\_Loops 17 days ago**

**03-Python - Session 1 - Activity - 12-Stu\_NumberChain**

**03-Python - Session 2 - Activity - 01-Stu\_QuickCheckup 17 days ago**

**03-Python - Session 2 - Activity - 02-Ins\_SimpleLoops 17 days ago**

**03-Python - Session 2 - Activity - 03-Stu\_KidInCandyStore 17 days ago**

**03-Python - Session 2 - Activity - 04-Stu\_HouseOfPies 17 days ago**

**03-Python - Session 2 - Activity - 05-Ins\_BasicRead 17 days ago**

**03-Python - Session 2 - Activity - 06-Ins\_Modules 17 days ago**

**03-Python - Session 2 - Activity - 07-Ins\_ReadCSV 17 days ago**

**03-Python - Session 2 - Activity - 08-Stu\_ReadNetFlix 17 days ago**

**03-Python - Session 2 - Activity - 09-Ins\_WriteCSV 17 days ago**

**03-Python - Session 2 - Activity - 10-Ins\_Zip 17 days ago**

**03-Python - Session 2 - Activity - 11-Stu\_UdemyZip 17 days ago**

**03-Python - Session 2 - Activity - 12-Ins\_Functions**

**03-Python - Session 3 - Activity - 01-Stu\_CerealCleaner 17 days ago**

**03-Python - Session 3 - Activity - 02-Ins\_Dicts 17 days ago**

**03-Python - Session 3 - Activity - 03-Stu\_HobbyBook 17 days ago**

**03-Python - Session 3 - Activity - 04-Evr\_List\_Comprehensions 17 days ago**

**03-Python - Session 3 - Activity - 05-Stu\_List\_Comprehensions 17 days ago**

**03-Python - Session 3 - Activity - 06-Evr\_Functions 17 days ago**

**03-Python - Session 3 - Activity - 07-Stu\_Functions 17 days ago**

**03-Python - Session 3 - Activity - 08-Par\_WrestlingWithFunctions**

**03-Python - Session 1 - Activity - 01-Ins\_JupyterIntro 10 days ago**

**04-Pandas - Session 1 - Activity - 02-Stu\_NetflixRemix 10 days ago**

**04-Pandas - Session 1 - Activity - 03-Ins\_IntroToPandas 10 days ago**

**04-Pandas - Session 1 - Activity - 04-Stu\_DataFrameShop 10 days ago**

**04-Pandas - Session 1 - Activity - 05-Ins\_DataFunctions 10 days ago**

**04-Pandas - Session 1 - Activity - 06-Stu\_TrainingGrounds 10 days ago**

**04-Pandas - Session 1 - Activity - 07-Ins\_ColumnManipulation 10 days ago**

**04-Pandas - Session 1 - Activity - 08-Stu\_Hey\_Arnold 10 days ago**

**04-Pandas - Session 1 - Activity - 09-Ins\_ReadingWritingCSV 10 days ago**

**04-Pandas - Session 1 - Activity - 10-Stu\_GoodReads 10 days ago**

**04-Pandas - Session 1 - Activity - 11-Stu\_GoodReadsSummary**

**04-Pandas - Session 2 - Activity - 01-Ins\_LocAndIloc 10 days ago**

**04-Pandas - Session 2 - Activity - 02-Stu\_GoodMovies 10 days ago**

**04-Pandas - Session 2 - Activity - 03-Ins\_CleaningData 10 days ago**

**04-Pandas - Session 2 - Activity - 04-Par\_PortlandCrime 10 days ago**

**04-Pandas - Session 2 - Activity - 05-Evr\_PandasRecap 10 days ago**

**04-Pandas - Session 2 - Activity - 06-Ins\_GroupBy 10 days ago**

**04-Pandas - Session 2 - Activity - 07-Par\_Pokemon 10 days ago**

**04-Pandas - Session 2 - Activity - 08-Ins\_Sorting 10 days ago**

**04-Pandas - Session 2 - Activity - 09-Stu\_SearchForTheWorst**

**04-Pandas - Session 3 - Activity - 01-Ins\_Merging 10 days ago**

**04-Pandas - Session 3 - Activity - 02-Stu\_Cryptocurrency 7 days ago**

**04-Pandas - Session 3 - Activity - 03-Ins\_Binning 10 days ago**

**04-Pandas - Session 3 - Activity - 04-Stu\_TedTalks 7 days ago**

**04-Pandas - Session 3 - Activity - 05-Ins\_Mapping 10 days ago**

**04-Pandas - Session 3 - Activity - 06-Stu\_CleaningKickstarter 7 days ago**

**04-Pandas - Session 3 - Activity - 07-Ins\_IntroToBugfixing 10 days ago**

**04-Pandas - Session 3 - Activity - 08-Stu\_BugfixingBonanza**

**05-Matplotlib - Session 1 - Activity - 01-Ins\_BasicLineGraphs 3 days ago**

*Desktop/RUTJER201809DATA3-master/05-Matplotlib/Classwork/1/Activities/01 Ins\_BasicLineGraphs/Solved/exponential\_chart.ipynb*

# Import Numpy for calculations and matplotlib for charting

import numpy as np

import matplotlib.pyplot as plt

# Creates a list from 0 to 5 with each step being 0.1 higher than the last

x\_axis = np.arange(0, 5, 0.1)

x\_axis

# Creates an exponential series of values which we can then chart

e\_x = [np.exp(x) for x in x\_axis]

e\_x

# Create a graph based upon the two lists we have created

plt.plot(x\_axis, e\_x)

# Show the graph that we have created

plt.show()

# Give our graph axis labels

plt.xlabel("Time With MatPlotLib")

plt.ylabel("How Cool MatPlotLib Seems")

# Have to plot our chart once again as it doesn't stick after being shown

plt.plot(x\_axis, e\_x)

plt.show()

*Desktop/RUTJER201809DATA3-master/05-Matplotlib/Classwork/1/Activities/01 Ins\_BasicLineGraphs/Solved/ sin\_cos.ipynb*

# Import Numpy for calculations and matplotlib for charting

import numpy as np

import matplotlib.pyplot as plt

# Create our x\_axis list

x\_axis = np.arange(0, 6, 0.1)

# Creates a list based on the sin of our x\_axis values

sin = np.sin(x\_axis)

# Creates a list based on the cos of our x\_axis values

cos = np.cos(x\_axis)

# Plot both of these lines so that they will appear on our final chart

plt.plot(x\_axis, sin)

plt.plot(x\_axis, cos)

plt.show()

**05-Matplotlib - Session 1 - Activity - 02-Stu\_NJTemp 3 days ago**

*Desktop/RUTJER201809DATA3-master/05-Matplotlib/Classwork/1/Activities/02-Stu\_NJTemp/Solved/NJ\_temp.ipynb*

# Dependencies

import numpy as np

import matplotlib.pyplot as plt

# Set x axis to numerical value for month

x\_axis\_data = np.arange(1,13,1)

x\_axis\_data

# Average weather temp

points = [39, 42, 51, 62, 72, 82, 86, 84, 77, 65, 55, 44]

# Plot the line

plt.plot(x\_axis\_data, points)

plt.show()

# Convert to Celsius C = (F-32) \* 0.56

points\_C = [(x-32) \* 0.56 for x in points]

points\_C

# Plot using Celsius

plt.plot(x\_axis\_data, points\_C)

plt.show()

# Plot both on the same chart

plt.plot(x\_axis\_data, points)

plt.plot(x\_axis\_data, points\_C)

plt.show()

**05-Matplotlib - Session 1 - Activity - 03-Ins\_ConfiguringLinePlots 3 days ago**

*Desktop/RUTJER201809DATA3-master/05-Matplotlib/Classwork/1/Activities/03-Ins\_ConfiguringLinePlots/Solved*

%matplotlib notebook

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

# Set x axis and variables

x\_axis = np.arange(0, 10, 0.1)

sin = np.sin(x\_axis)

cos = np.cos(x\_axis)

# Draw a horizontal line with 0.25 transparency

plt.hlines(0, 0, 10, alpha=0.25)

# Assign plots to tuples that stores result of plot

# Each point on the sine chart is marked by a blue circle

sine\_handle, = plt.plot(x\_axis, sin, marker ='o', color='blue', label="Sine")

# Each point on the cosine chart is marked by a red triangle

cosine\_handle, = plt.plot(x\_axis, cos, marker='^', color='red', label="Cosine")

**05-Matplotlib - Session 1 - Activity - 04-Stu\_LegendaryTemperature 3 days ago**

# Include this line to make plots interactive

%matplotlib notebook

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

# Set x axis to numerical value for month

x\_axis = np.arange(1,13,1)

x\_axis

# Avearge weather temp

points\_F = [39, 42, 51, 62, 72, 82, 86, 84, 77, 65, 55, 44]

# Convert to Celsius C = (F-32) \* 0.56

points\_C = [(x-32) \* 0.56 for x in points\_F]

points\_C

# Create a handle for each plot

fahrenheit, = plt.plot(x\_axis, points\_F, marker="+",color="blue", linewidth=1, label="Fahreneit")

celcius, = plt.plot(x\_axis, points\_C, marker="s", color="Red", linewidth=1, label="Celcius")

# Set our legend to where the chart thinks is best

plt.legend(handles=[fahrenheit, celcius], loc="best")

# Create labels for the X and Y axis

plt.xlabel("Months")

plt.ylabel("Degrees")

# Save and display the chart

plt.savefig("../Images/avg\_temp.png")

plt.show()

**05-Matplotlib - Session 1 - Activity - 05-Ins\_Aesthetics 3 days ago**

%matplotlib notebook

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

# Generate the x values from 0 to 10 using a step of 0.1

x\_axis = np.arange(0, 10, 0.1)

sin = np.sin(x\_axis)

cos = np.cos(x\_axis)

# Add a semi-transparent horizontal line at y = 0

plt.hlines(0, 0, 10, alpha=0.25)

# Use dots or other markers for your plots, and change their colors

plt.plot(x\_axis, sin, linewidth=0, marker="o", color="blue")

plt.plot(x\_axis, cos, linewidth=0, marker="^", color="red")

# Add labels to the x and y axes

plt.title("Juxtaposed Sine and Cosine Curves")

plt.xlabel("Input (Sampled Real Numbers from 0 to 10)")

plt.ylabel("Value of Sine (blue) and Cosine (red)")

# Set your x and y limits

plt.xlim(0, 10)

plt.ylim(-1, 1)

# Set a grid on the plot

plt.grid()

# Save the plot and display it

plt.savefig("../Images/sin\_cos\_with\_markers.png")

plt.show()

**05-Matplotlib - Session 1 - Activity - 06-Stu\_RollerCoaster 3 days ago**

%matplotlib notebook

# Import Dependencies

import matplotlib.pyplot as plt

import numpy as np

# Create the X and Y axis lists

time = np.arange(0,130,10)

speed\_chain = [9, 8, 90, 85, 80, 70, 70, 65, 55, 60, 70, 65, 50]

speed\_launch = [75, 70, 60, 65, 60, 45, 55, 50, 40, 40, 35, 35, 30]

# Plot the charts and apply some styling

danger\_drop, = plt.plot(time, speed\_chain, color="red", label="Danger Drop")

railgun, = plt.plot(time, speed\_launch, color="blue", label="RailGun")

# Add labels to X and Y axes :: Add title

plt.title("Coaster Speed Over Time")

plt.xlabel("Coaster Runtime")

plt.ylabel("Speed (MPH)")

# Set the limits for the X and Y axes

plt.xlim(0,120)

plt.ylim(5,95)

# Create a legend for the chart

plt.legend(handles=[danger\_drop, railgun], loc="best")

# Add in a grid for the chart

plt.grid()

plt.show()

**05-Matplotlib - Session 1 - Activity - 07-Ins\_BarCharts 3 days ago**

%matplotlib notebook

import matplotlib.pyplot as plt

import numpy as np

# Create an array that contains the number of users each language has

users = [13000, 26000, 52000, 30000, 9000]

x\_axis = np.arange(len(users))

# Tell matplotlib that we will be making a bar chart

# Users is our y axis and x\_axis is, of course, our x axis

# We apply align="edge" to ensure our bars line up with our tick marks

plt.bar(x\_axis, users, color='r', alpha=0.5, align="center")

# Tell matplotlib where we would like to place each of our x axis headers

tick\_locations = [value for value in x\_axis]

plt.xticks(tick\_locations, ["Java", "C++", "Python", "Ruby", "Clojure"])

# Sets the x limits of the current chart

plt.xlim(-0.75, len(x\_axis)-0.25)

# Sets the y limits of the current chart

plt.ylim(0, max(users)+5000)

# Give our chart some labels and a tile

plt.title("Popularity of Programming Languages")

plt.xlabel("Programming Language")

plt.ylabel("Number of People Using Programming Languages")

**05-Matplotlib - Session 1 - Activity - 08-Stu\_PyBars 3 days ago**

%matplotlib notebook

import matplotlib.pyplot as plt

import numpy as np

cities = ["New Orleans", "Milwaukee", "Omaha", "Pittsburgh", "Toledo"]

bars\_in\_cities = [8.6, 8.5, 8.3, 7.9, 7.2]

x\_axis = np.arange(len(bars\_in\_cities))

# Create a bar chart based upon the above data

plt.bar(x\_axis, bars\_in\_cities, color="b", align="center")

# Create the ticks for our bar chart's x axis

tick\_locations = [value for value in x\_axis]

plt.xticks(tick\_locations, cities)

# Set the limits of the x axis

plt.xlim(-0.75, len(x\_axis)-0.25)

# Set the limits of the y axis

plt.ylim(0, max(bars\_in\_cities)+0.4)

# Give the chart a title, x label, and y label

plt.title("Density of Bars in Cities")

plt.xlabel("Cities")

plt.ylabel("Bars Per 10,000 Households")

# Save an image of the chart and print it to the screen

plt.savefig("../Images/BarDensity.png")

plt.show()

**05-Matplotlib - Session 1 - Activity - 09-Ins\_PieCharts 3 days ago**

%matplotlib notebook

# Import our dependencies

import matplotlib.pyplot as plt

import numpy as np

# Labels for the sections of our pie chart

labels = ["Humans", "Smurfs", "Hobbits", "Ninjas"]

# The values of each section of the pie chart

sizes = [220, 95, 80, 100]

# The colors of each section of the pie chart

colors = ["red", "orange", "lightcoral", "lightskyblue"]

# Tells matplotlib to seperate the "Python" section from the others

explode = (0.1, 0, 0, 0)

# Creates the pie chart based upon the values above

# Automatically finds the percentages of each part of the pie chart

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct="%1.1f%%", shadow=True, startangle=140)

# Tells matplotlib that we want a pie chart with equal axes

plt.axis("equal")

**05-Matplotlib - Session 1 - Activity - 10-Stu\_PyPies 3 days ago**

%matplotlib notebook

import matplotlib.pyplot as plt

import numpy as np

pies = ["Apple", "Pumpkin", "Chocolate Creme", "Cherry", "Apple Crumb", "Pecan", "Lemon Meringue", "Blueberry", "Key Lime", "Peach"]

pie\_votes = [47,37,32,27,25,24,24,21,18,16]

colors = ["yellow","green","lightblue","orange","red","purple","pink","yellowgreen","lightskyblue","lightcoral"]

explode = (0.1,0,0,0,0,0,0,0,0,0)

# Tell matplotlib to create a pie chart based upon the above data

plt.pie(pie\_votes, explode=explode, labels=pies, colors=colors,

autopct="%1.1f%%", shadow=True, startangle=140)

# Create axes which are equal so we have a perfect circle

plt.axis("equal")

# Save an image of our chart and print the final product to the screen

plt.savefig("../Images/PyPies.png")

plt.show()

**05-Matplotlib - Session 1 - Activity - 11-Ins\_ScatterPlots 3 days ago**

%matplotlib notebook

# Import Dependencies

import random

import matplotlib.pyplot as plt

import numpy as np

# The maximum x value for our chart will be 100

x\_limit = 100

# List of values from 0 to 100 each value being 1 greater than the last

x\_axis = np.arange(0, x\_limit, 1)

# Create a random array of data that we will use for our y values

data = [random.random() for value in x\_axis]

# Tells matplotlib that we want to make a scatter plot

# The size of each point on our plot is determined by their x value

plt.scatter(x\_axis, data, marker="o", facecolors="red", edgecolors="black",

s=x\_axis, alpha=0.75)

# The y limits of our scatter plot is 0 to 1

plt.ylim(0, 1)

# The x limits of our scatter plot is 0 to 100

plt.xlim(0, x\_limit)

# Prints the scatter plot to the screen

plt.show()

**05-Matplotlib - Session 1 - Activity - 12-Stu\_ScatterPy 3 days ago**

%matplotlib notebook

import matplotlib.pyplot as plt

import numpy as np

temp = [14.2, 16.4, 11.9, 15.2, 18.5, 22.1, 19.4, 25.1, 23.4, 18.1, 22.6, 17.2]

sales = [215, 325, 185, 332, 406, 522, 412, 614, 544, 421, 445, 408]

# Tell matplotlib to create a scatter plot based upon the above data

# Without scoop\_price

plt.scatter(temp, sales, marker="o", facecolors="red", edgecolors="black")

# BONUS: With scoop\_price set to the scalar value

# scoop\_price = [89, 18, 10, 28, 79, 46, 29, 38, 89, 26, 45, 62]

# plt.scatter(temp, sales, marker="o", facecolors="red", edgecolors="black", s=scoop\_price)

# Set the upper and lower limits of our y axis

plt.ylim(180,620)

# Set the upper and lower limits of our x axis

plt.xlim(11,26)

# Create a title, x label, and y label for our chart

plt.title("Ice Cream Sales v Temperature")

plt.xlabel("Temperature (Celsius)")

plt.ylabel("Sales (Dollars)")

# Save an image of the chart and print to screen

# NOTE: If your plot shrinks after saving an image,

# update matplotlib to 2.2 or higher,

# or simply run the above cells again.

plt.savefig("../Images/IceCreamSales.png")

plt.show()

**05-Matplotlib - Session 1 - Activity - 13-Stu\_AvgRain**

%matplotlib notebook

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# Load in csv

rain\_df = pd.read\_csv("../Resources/avg\_rain\_state.csv")

rain\_df.head()

# Set x axis and tick locations

x\_axis = np.arange(len(rain\_df))

tick\_locations = [value+0.4 for value in x\_axis]

# Create a list indicating where to write x labels and set figure size to adjust for space

plt.figure(figsize=(20,3))

plt.bar(x\_axis, rain\_df["Inches"], color='r', alpha=0.5, align="edge")

plt.xticks(tick\_locations, rain\_df["State"], rotation="vertical")

# Set x and y limits

plt.xlim(-0.25, len(x\_axis))

plt.ylim(0, max(rain\_df["Inches"])+10)

# Set a Title and labels

plt.title("Average Rain per State")

plt.xlabel("State")

plt.ylabel("Average Amount of Rainfall in Inches")

# Save our graph and show the grap

plt.tight\_layout()

plt.savefig("../Images/avg\_state\_rain.png")

plt.show()

**05-Matplotlib - Session 2 - Activity - 01-Stu\_PlotsReview 3 days ago**

# Import Dependencies

import numpy as np

import matplotlib.pyplot as plt

# DATASET 1

gyms = ["Crunch", "Planet Fitness", "NY Sports Club", "Rickie's Gym"]

members = [49, 92, 84, 53]

x\_axis = np.arange(0, len(gyms))

tick\_locations = []

for x in x\_axis:

tick\_locations.append(x)

plt.title("NYC Gym Popularity")

plt.xlabel("Gym Name")

plt.ylabel("Number of Members")

plt.xlim(-0.75, len(gyms)-.25)

plt.ylim(0, max(members) + 5)

plt.bar(x\_axis, members, facecolor="red", alpha=0.75, align="center")

plt.xticks(tick\_locations, gyms)

plt.show()

# DATASET 2

x\_lim = 2 \* np.pi

x\_axis = np.arange(0, x\_lim, 0.1)

sin = np.sin(x\_axis)

plt.title("Sin from 0 to 2$\pi$")

plt.xlabel("Real Numbers from 0 to 2$\pi$")

plt.ylabel("sin(x)")

plt.hlines(0, 0, x\_lim, alpha=0.2)

plt.xlim(0, x\_lim)

plt.ylim(-1.25, 1.25)

plt.plot(x\_axis, sin, marker="o", color="red", linewidth=1)

plt.show()

# DATASET 3

gyms = ["Crunch", "Planet Fitness", "NY Sports Club", "Rickie's Gym"]

members = [49, 92, 84, 53]

colors = ["yellowgreen", "red", "lightcoral", "lightskyblue"]

explode = (0, 0.05, 0, 0)

plt.title("NYC Gym Popularity")

plt.pie(members, explode=explode, labels=gyms, colors=colors,

autopct="%1.1f%%", shadow=True, startangle=90)

plt.axis("equal")

plt.show()

# DATASET 4

x\_axis = np.arange(0, 10, 0.1)

times = []

for x in x\_axis:

times.append(x \* x + np.random.randint(0, np.ceil(max(x\_axis))))

plt.title("Running Time of FakeSort for Sample Input Sizes")

plt.xlabel("Length of Input Array")

plt.ylabel("Time to Sort (s)")

plt.scatter(x\_axis, times, marker="o", color="red")

plt.show()

**05-Matplotlib - Session 2 - Activity - 02-Ins\_PandasPlot 3 days ago**

%matplotlib notebook

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# Load in csv

rain\_df = pd.read\_csv("../Resources/avg\_rain\_state.csv")

rain\_df.head()

# Set x axis and tick locations

x\_axis = np.arange(len(rain\_df))

tick\_locations = [value for value in x\_axis]

# Create a list indicating where to write x labels and set figure size to adjust for space

plt.figure(figsize=(20,3))

plt.bar(x\_axis, rain\_df["Inches"], color='r', alpha=0.5, align="center")

plt.xticks(tick\_locations, rain\_df["State"], rotation="vertical")

# Set x and y limits

plt.xlim(-0.75, len(x\_axis))

plt.ylim(0, max(rain\_df["Inches"])+10)

# Set a Title and labels

plt.title("Average Rain per State")

plt.xlabel("State")

plt.ylabel("Average Amount of Rainfall in Inches")

# Save our graph and show the grap

plt.tight\_layout()

plt.savefig("../Images/avg\_state\_rain.png")

plt.show()

# Filter the DataFrame down only to those columns to chart

state\_and\_inches = rain\_df[["State","Inches"]]

# Set the index to be "State" so they will be used as labels

state\_and\_inches = state\_and\_inches.set\_index("State")

state\_and\_inches.head()

# Use DataFrame.plot() in order to create a bar chart of the data

state\_and\_inches.plot(kind="bar", figsize=(20,3))

# Set a title for the chart

plt.title("Average Rain Per State")

plt.show()

plt.tight\_layout()

# Pandas can also plot multiple columns if the DataFrame includes them

multi\_plot = rain\_df.plot(kind="bar", figsize=(20,5))

# PandasPlot.set\_xticklabels() can be used to set the tick labels as well

multi\_plot.set\_xticklabels(rain\_df["State"], rotation=45)

plt.show()

plt.tight\_layout()

**05-Matplotlib - Session 2 - Activity - 03-Stu\_BattlingKings 3 days ago**

%matplotlib notebook

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# Read CSV

got\_data = pd.read\_csv("Resources/got.csv")

got\_data

# Get attacker and defender data

attacker\_data = got\_data["attacker\_king"].value\_counts()

defender\_data = got\_data["defender\_king"].value\_counts()

# Get total battle data

battle\_data = attacker\_data.add(defender\_data, fill\_value=0)

battle\_data

# Configure plot and ticks

battle\_data.plot(kind="bar", facecolor="red")

# Set textual properties

plt.title("The Bloodthirst of Kings")

plt.ylabel("Number of Battles Participated In")

plt.xlabel("King")

# Show plot

plt.show()

# Resize plot to display labels

plt.tight\_layout()

**05-Matplotlib - Session 2 - Activity - 04-Ins\_GroupPlots 3 days ago**

%matplotlib notebook

# Import Dependencies

import matplotlib.pyplot as plt

import pandas as pd

# Import our data into pandas from CSV

used\_string = '../Resources/used\_cars.csv'

used\_car\_df = pd.read\_csv(used\_string)

used\_car\_df

# Create a group based on the values in the 'maker' column

maker\_group = used\_car\_df.groupby('maker')

# Count how many times each maker appears in our group

count\_makers = maker\_group['maker'].count()

count\_makers

# Create a bar chart based off of the group series from before

count\_chart = count\_makers.plot(kind='bar')

# Set the xlabel and ylabel using class methods

count\_chart.set\_xlabel("Car Manufacturer")

count\_chart.set\_ylabel("Number of Cars")

plt.show()

plt.tight\_layout()

**05-Matplotlib - Session 2 - Activity - 05-Stu\_BikeTrippin 3 days ago**

%matplotlib notebook

# Import Dependencies

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# Import our data into pandas from CSV

string\_thing = '../Resources/trip.csv'

bike\_trips\_df = pd.read\_csv(string\_thing, low\_memory=False)

bike\_trips\_df

# Split up our data into groups based upon 'gender'

gender\_groups = bike\_trips\_df.groupby('gender')

# Find out how many of each gender took bike trips

gender\_trips = gender\_groups['tripduration'].count()

gender\_trips.drop("stoptime")

# Drop the 'stoptime' row that is contained within our group

gender\_trips = gender\_trips.drop(gender\_trips.index[3])

# Chart our data, give it a title, and label the axes

gender\_chart = gender\_trips.plot(kind="bar", title="Bike Trips by Gender")

gender\_chart.set\_xlabel("Gender")

gender\_chart.set\_ylabel("Number of Trips Taken")

plt.show()

plt.tight\_layout()

# Split up our data into groups based upon 'bikeid' and 'gender'

bike\_groups = bike\_trips\_df.groupby(['bikeid','gender'])

# Create a new variable that holds the sum of our groups

sum\_it\_up = bike\_groups.sum()

sum\_it\_up.head(12)

# Make a variable called bike\_id and store a 'bikeid' in it

bike\_id = "SEA00001"

# Collect the trips of the 'bikeid' above

just\_one\_bike = sum\_it\_up.loc[bike\_id]

# Place the gender keys for that single bike into a list

gender\_list = just\_one\_bike.keys()

# Create a pie chart based upon the trip duration of that single bike

bike\_pie = just\_one\_bike.plot(kind="pie", y=gender\_list, title=("Trips of " + bike\_id))

bike\_pie.set\_ylabel("Trip Duration")

plt.show()

plt.tight\_layout()

plt.axis("equal")

**05-Matplotlib - Session 2 - Activity - 06-Stu\_MilesPerGallon 3 days ago**

%matplotlib notebook

# Dependencies and Setup

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

car\_data = pd.read\_csv('../Resources/mpg.csv')

car\_data.head()

# Remove the rows with missing values in horsepower

car\_data = car\_data.loc[car\_data['horsepower'] != "?"]

car\_data.head()

# Set the 'car name' as our index

car\_data = car\_data.set\_index('car name')

# Remove the 'origin' column

del car\_data['origin']

car\_data.head()

# Convert the "horsepower" column to numeric so the data can be used

car\_data['horsepower'] = pd.to\_numeric(car\_data['horsepower'])

# Create a scatter plot which compares MPG to horsepower

car\_data.plot(kind="scatter", x="horsepower", y="mpg", grid=True, figsize=(20,10),

title="Horsepower Vs. MPG")

plt.show()

**05-Matplotlib - Session 2 - Activity - 07-Ins\_PandasMultiLine 3 days ago**

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# Read CSV

unemployed\_data\_one = pd.read\_csv("../Resources/unemployment\_2010-2011.csv")

unemployed\_data\_two = pd.read\_csv("../Resources/unemployment\_2012-2014.csv")

# Merge our two data frames together

combined\_unemployed\_data = pd.merge(unemployed\_data\_one, unemployed\_data\_two, on="Country Name")

combined\_unemployed\_data.head()

# Delete the duplicate 'Country Code' column and rename the first one back to 'Country Code'

del combined\_unemployed\_data['Country Code\_y']

combined\_unemployed\_data = combined\_unemployed\_data.rename(columns={"Country Code\_x":"Country Code"})

combined\_unemployed\_data.head()

# Set the 'Country Code' to be our index for easy referencing of rows

combined\_unemployed\_data = combined\_unemployed\_data.set\_index("Country Code")

# Collect the mean unemployment rates for the world

average\_unemployment = combined\_unemployed\_data.mean()

# Collect the years where data was collected

years = average\_unemployment.keys()

# Plot the world average as a line chart

world\_avg, = plt.plot(years, average\_unemployment, color="blue", label="World Average" )

# Plot the unemployment values for a single country

country\_one, = plt.plot(years, combined\_unemployed\_data.loc['USA',["2010","2011","2012","2013","2014"]],

color="green",label=combined\_unemployed\_data.loc['USA',"Country Name"])

# Create a legend for our chart

plt.legend(handles=[world\_avg, country\_one], loc="best")

# Show the chart

plt.show()

average\_unemployment.plot(label="World Average")

combined\_unemployed\_data.loc['USA', "2010":"2014"].plot(label="United States")

plt.legend()

plt.show()

**05-Matplotlib - Session 2 - Activity - 08-Stu\_WinnerWrestling-Part1 3 days ago**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# Take in all of our wrestling data and read it into pandas

wrestling\_2013 = "../Resources/WWE-Data-2013.csv"

wrestling\_2014 = "../Resources/WWE-Data-2014.csv"

wrestling\_2015 = "../Resources/WWE-Data-2015.csv"

wrestling\_2016 = "../Resources/WWE-Data-2016.csv"

wrestlers\_2013\_df = pd.read\_csv(wrestling\_2013)

wrestlers\_2014\_df = pd.read\_csv(wrestling\_2014)

wrestlers\_2015\_df = pd.read\_csv(wrestling\_2015)

wrestlers\_2016\_df = pd.read\_csv(wrestling\_2016)

# Merge the first two datasets on "Wrestler" so that no data is lost (should be 182 rows)

combined\_wrestlers\_df = pd.merge(wrestlers\_2013\_df, wrestlers\_2014\_df,

how='outer', on='Wrestler')

combined\_wrestlers\_df.head()

# Rename our \_x columns to "2013 Wins", "2013 Losses", and "2013 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins\_x":"2013 Wins",

"Losses\_x":"2013 Losses",

"Draws\_x":"2013 Draws"})

# Rename our \_y columns to "2014 Wins", "2014 Losses", and "2014 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins\_y":"2014 Wins",

"Losses\_y":"2014 Losses",

"Draws\_y":"2014 Draws"})

combined\_wrestlers\_df.head()

# Merge our newly combined dataframe with the 2015 dataframe

combined\_wrestlers\_df = pd.merge(combined\_wrestlers\_df, wrestlers\_2015\_df, how="outer", on="Wrestler")

combined\_wrestlers\_df

# Rename "wins", "losses", and "draws" to "2015 Wins", "2015 Losses", and "2015 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins":"2015 Wins","Losses":"2015 Losses","Draws":"2015 Draws"})

combined\_wrestlers\_df.head()

# Merge our newly combined dataframe with the 2016 dataframe

combined\_wrestlers\_df = pd.merge(combined\_wrestlers\_df, wrestlers\_2016\_df, how="outer", on="Wrestler")

combined\_wrestlers\_df

# Rename "wins", "losses", and "draws" to "2016 Wins", "2016 Losses", and "2016 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins":"2016 Wins","Losses":"2016 Losses","Draws":"2016 Draws"})

combined\_wrestlers\_df.head(10)

**05-Matplotlib - Session 2 - Activity - 09-Stu\_WinnerWrestling-Part2 3 days ago**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# Take in all of our wrestling data and read it into pandas

wrestling\_2013 = "../Resources/WWE-Data-2013.csv"

wrestling\_2014 = "../Resources/WWE-Data-2014.csv"

wrestling\_2015 = "../Resources/WWE-Data-2015.csv"

wrestling\_2016 = "../Resources/WWE-Data-2016.csv"

wrestlers\_2013\_df = pd.read\_csv(wrestling\_2013)

wrestlers\_2014\_df = pd.read\_csv(wrestling\_2014)

wrestlers\_2015\_df = pd.read\_csv(wrestling\_2015)

wrestlers\_2016\_df = pd.read\_csv(wrestling\_2016)

# Merge the first two datasets on "Wrestler" so that no data is lost (should be 182 rows)

combined\_wrestlers\_df = pd.merge(wrestlers\_2013\_df, wrestlers\_2014\_df, how='outer', on='Wrestler')

combined\_wrestlers\_df

# Rename our \_x columns to "2013 Wins", "2013 Losses", and "2013 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins\_x":"2013 Wins", "Losses\_x":"2013 Losses", "Draws\_x":"2013 Draws"})

# Rename our \_y columns to "2014 Wins", "2014 Losses", and "2014 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins\_y":"2014 Wins","Losses\_y":"2014 Losses","Draws\_y":"2014 Draws"})

combined\_wrestlers\_df.head()

# Merge our newly combined dataframe with the 2015 dataframe

combined\_wrestlers\_df = pd.merge(combined\_wrestlers\_df, wrestlers\_2015\_df, how="outer", on="Wrestler")

combined\_wrestlers\_df

# Rename "wins", "losses", and "draws" to "2015 Wins", "2015 Losses", and "2015 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins":"2015 Wins","Losses":"2015 Losses","Draws":"2015 Draws"})

combined\_wrestlers\_df.head()

# Merge our newly combined dataframe with the 2016 dataframe

combined\_wrestlers\_df = pd.merge(combined\_wrestlers\_df, wrestlers\_2016\_df, how="outer", on="Wrestler")

combined\_wrestlers\_df

# Rename "wins", "losses", and "draws" to "2016 Wins", "2016 Losses", and "2016 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins":"2016 Wins","Losses":"2016 Losses","Draws":"2016 Draws"})

combined\_wrestlers\_df.head()

# Replace all NaN values with 0

combined\_wrestlers\_df = combined\_wrestlers\_df.fillna(0)

# Create a new column called "Total Wins" and add up each wrestler's wins per year to fill in the values

combined\_wrestlers\_df["Total Wins"] = combined\_wrestlers\_df["2013 Wins"] + combined\_wrestlers\_df["2014 Wins"] + combined\_wrestlers\_df["2015 Wins"] + combined\_wrestlers\_df["2016 Wins"]

# Create a new column called "Total Losses" and add up each wrestler's losses per year to fill in the values

combined\_wrestlers\_df["Total Losses"] = combined\_wrestlers\_df["2013 Losses"] + combined\_wrestlers\_df["2014 Losses"] + combined\_wrestlers\_df["2015 Losses"] + combined\_wrestlers\_df["2016 Losses"]

# Create a new column called "Total Draws" and add up each wrestler's draws per year to fill in the values

combined\_wrestlers\_df["Total Draws"] = combined\_wrestlers\_df["2013 Draws"] + combined\_wrestlers\_df["2014 Draws"] + combined\_wrestlers\_df["2015 Draws"] + combined\_wrestlers\_df["2016 Draws"]

# Create a new column called "Total Matches" and add up the total wins, losses, and draws for each wrestler to fill in the values

combined\_wrestlers\_df["Total Matches"] = combined\_wrestlers\_df["Total Wins"] + combined\_wrestlers\_df["Total Losses"] + combined\_wrestlers\_df["Total Draws"]

combined\_wrestlers\_df

# Create a new dataframe for those wrestlers who have wrestled at least 100 matches,

# have at least one win in 2013,

# and have at least one win in 2016

wrestled\_over\_hundred = combined\_wrestlers\_df.loc[(combined\_wrestlers\_df["Total Matches"] >= 100) &

(combined\_wrestlers\_df["2013 Wins"] > 0) &

(combined\_wrestlers\_df["2016 Wins"] > 0)]

# Set the index of this new dataframe to be the wrestlers names

wrestled\_over\_hundred = wrestled\_over\_hundred.set\_index("Wrestler")

wrestled\_over\_hundred.head()

**05-Matplotlib - Session 2 - Activity - 10-Stu\_WinnerWrestling-Part3**

%matplotlib notebook

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# Take in all of our wrestling data and read it into pandas

wrestling\_2013 = "../Resources/WWE-Data-2013.csv"

wrestling\_2014 = "../Resources/WWE-Data-2014.csv"

wrestling\_2015 = "../Resources/WWE-Data-2015.csv"

wrestling\_2016 = "../Resources/WWE-Data-2016.csv"

wrestlers\_2013\_df = pd.read\_csv(wrestling\_2013)

wrestlers\_2014\_df = pd.read\_csv(wrestling\_2014)

wrestlers\_2015\_df = pd.read\_csv(wrestling\_2015)

wrestlers\_2016\_df = pd.read\_csv(wrestling\_2016)

# Merge the first two datasets on "Wrestler" so that no data is lost (should be 182 rows)

combined\_wrestlers\_df = pd.merge(wrestlers\_2013\_df, wrestlers\_2014\_df, how='outer', on='Wrestler')

combined\_wrestlers\_df

# Rename our \_x columns to "2013 Wins", "2013 Losses", and "2013 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins\_x":"2013 Wins", "Losses\_x":"2013 Losses", "Draws\_x":"2013 Draws"})

# Rename our \_y columns to "2014 Wins", "2014 Losses", and "2014 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins\_y":"2014 Wins","Losses\_y":"2014 Losses","Draws\_y":"2014 Draws"})

combined\_wrestlers\_df.head()

# Merge our newly combined dataframe with the 2015 dataframe

combined\_wrestlers\_df = pd.merge(combined\_wrestlers\_df, wrestlers\_2015\_df, how="outer", on="Wrestler")

combined\_wrestlers\_df

# Rename "wins", "losses", and "draws" to "2015 Wins", "2015 Losses", and "2015 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins":"2015 Wins","Losses":"2015 Losses","Draws":"2015 Draws"})

combined\_wrestlers\_df.head()

# Merge our newly combined dataframe with the 2016 dataframe

combined\_wrestlers\_df = pd.merge(combined\_wrestlers\_df, wrestlers\_2016\_df, how="outer", on="Wrestler")

combined\_wrestlers\_df

# Rename "wins", "losses", and "draws" to "2016 Wins", "2016 Losses", and "2016 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins":"2016 Wins","Losses":"2016 Losses","Draws":"2016 Draws"})

combined\_wrestlers\_df.head()

# Replace all NaN values with 0

combined\_wrestlers\_df = combined\_wrestlers\_df.fillna(0)

# Create a new column called "Total Wins" and add up each wrestler's wins per year to fill in the values

combined\_wrestlers\_df["Total Wins"] = combined\_wrestlers\_df["2013 Wins"] + combined\_wrestlers\_df["2014 Wins"] + combined\_wrestlers\_df["2015 Wins"] + combined\_wrestlers\_df["2016 Wins"]

# Create a new column called "Total Losses" and add up each wrestler's losses per year to fill in the values

combined\_wrestlers\_df["Total Losses"] = combined\_wrestlers\_df["2013 Losses"] + combined\_wrestlers\_df["2014 Losses"] + combined\_wrestlers\_df["2015 Losses"] + combined\_wrestlers\_df["2016 Losses"]

# Create a new column called "Total Draws" and add up each wrestler's draws per year to fill in the values

combined\_wrestlers\_df["Total Draws"] = combined\_wrestlers\_df["2013 Draws"] + combined\_wrestlers\_df["2014 Draws"] + combined\_wrestlers\_df["2015 Draws"] + combined\_wrestlers\_df["2016 Draws"]

# Create a new column called "Total Matches" and add up the total wins, losses, and draws for each wrestler to fill in the values

combined\_wrestlers\_df["Total Matches"] = combined\_wrestlers\_df["Total Wins"] + combined\_wrestlers\_df["Total Losses"] + combined\_wrestlers\_df["Total Draws"]

combined\_wrestlers\_df

# Create a new dataframe for those wrestlers who have wrestled at least 100 matches,

# have at least one win in 2013,

# and have at least one win in 2016

wrestled\_over\_hundred = combined\_wrestlers\_df.loc[(combined\_wrestlers\_df["Total Matches"] >= 100) &

(combined\_wrestlers\_df["2013 Wins"] > 0) &

(combined\_wrestlers\_df["2016 Wins"] > 0)]

# Set the index of this new dataframe to be the wrestlers names

wrestled\_over\_hundred = wrestled\_over\_hundred.set\_index("Wrestler")

wrestled\_over\_hundred.head()

# Collect the user's input to search through our data frame

wrestler\_name = input("What wrestler's career would you like to look at?")

# Create a series that looks for a wrestler by name and then traces their wins from 2013 to 2016

wins\_over\_time = wrestled\_over\_hundred.loc[wrestler\_name,["2013 Wins","2014 Wins", "2015 Wins", "2016 Wins"]]

# Create a series that looks for a wrestler by name and then traces their losses from 2013 to 2016

losses\_over\_time = wrestled\_over\_hundred.loc[wrestler\_name,["2013 Losses","2014 Losses",

"2015 Losses", "2016 Losses"]]

# Create a list of the years that we will use as our x axis

years = [2013,2014,2015,2016]

# Plot our line that will be used to track a wrestler's wins over the years

plt.plot(years, wins\_over\_time, color="green", label="Wins")

# Plot our line that will be used to track a wrestler's losses over the years

plt.plot(years, losses\_over\_time, color="blue", label="Losses")

# Place a legend on the chart in what matplotlib believes to be the "best" location

plt.legend(loc="best")

plt.title(wrestler\_name + "'s Recent Career")

plt.xlabel("Years")

plt.ylabel("Number of Wins/Losses")

# Print our chart to the screen

plt.show()

**05-Matplotlib - Session 3 - Activity - 01-Ins\_Mean\_Median\_Mode 3 days ago**

# Dependencies

from stats import mean, median, mode, multi\_mode

# Prices of random electronics at Best Buy

prices = [4, 425, 984, 2932, 49]

print(f"Median Price: {median(prices)}")

# Ages of students in bootcamp

bootcamp\_classroom\_ages = [27, 35, 42, 52, 36, 28]

print(f"Mean Bootcamp Age: {mean(bootcamp\_classroom\_ages)}")

print(f"Median Bootcamp Age: {median(bootcamp\_classroom\_ages)}")

# Ages of children and parents at child's party

birthday\_party\_ages = [6, 5, 6, 6, 35, 42, 34]

print(f"Mode of Birthday Party Ages: {mean(birthday\_party\_ages)}")

# Test score from a 2nd grade geography test

geo\_grades = [87, 89, 91, 93, 95]

print(f"Mean of Geography Test Scores: {mean(geo\_grades)}")

# Test scores from a graduate quantum mechanics midterm

quantum\_grades = [63, 63, 98, 13, 58, 13, 8]

print(f"Median of QM Grades: {median(quantum\_grades)}")

print(f"Modes of QM Grades: {multi\_mode(quantum\_grades)}")

print(mean(quantum\_grades))

**05-Matplotlib - Session 3 - Activity - 02-Ins\_Variance\_and\_Z\_Score 3 days ago**

# Dependencies

from spread import variance, standard\_deviation, zipped\_z\_scores

def summarize(title, arr):

print(f"Summarizing {title}")

print(f"Variance: {variance(arr)}")

print(f"Standard Deviation: {tandard\_deviation(arr)}")

print("Z-Scores: {zipped\_z\_scores(arr)}")

print("======")

# Prices of random electronics at Best Buy

prices = [4, 425, 984, 2932, 49]

summarize("Prices", prices)

# Ages of students in bootcamp

bootcamp\_classroom\_ages = [27, 35, 42, 52, 36, 28]

summarize("Bootcamp Ages", bootcamp\_classroom\_ages)

# Ages of children and parents at child's party

birthday\_party\_ages = [6, 5, 6, 6, 35, 34, 42]

summarize("Birthday Party Ages", birthday\_party\_ages)

# Test score from a 2nd grade geography test

geo\_grades = [87, 89, 91, 93, 95]

summarize("Geograph Grades", geo\_grades)

# Test scores from a graduate quantum mechanics midterm

quantum\_grades = [63, 63, 98, 13, 58, 13, 8]

summarize("Quantum Mechanics Grades", quantum\_grades)

# Prices

summarize("Prices", [30, 31, 31, 32, 32, 40, 41, 41, 1000])

**05-Matplotlib - Session 3 - Activity - 03-Ins\_Quartiles\_and\_Outliers 3 days ago**

# Dependencies

import numpy as np

numbers = [3, 3, 4, 5, 5, 6, 7, 7, 8, 8, 9]

median = 6

lower\_quartile = 4

upper\_quartile = 8

**05-Matplotlib - Session 3 - Activity - 04-Stu\_Quartiles\_and\_Outliers 3 days ago**

%matplotlib notebook

# Dependencies

import matplotlib.pyplot as plt

from stats import median

import numpy as np

### Data Points

arr = np.array([2.3, 10.2,11.2, 12.3, 14.5, 14.6, 15.0, 15.1, 19.0, 24.0])

arr

# Find the median

mid = median(arr)

mid

# Use numpy to create quartiles

q1 = np.percentile(arr, 25)

q3 = np.percentile(arr,75)

# Print the quartiles

print(f"Q1 is {q1}")

print(f"Q3 is {q3}")

# Calculate the interquartile range

iqr = (q3 - q1)

print("interquartile range:", iqr)

# Find lower boundary

# Q1 - 1.5 \* IQR

lower\_boundary = q1 - (1.5 \* iqr)

lower\_boundary

# Find upper boundary

# Q3 + 1.5 \* IQR

upper\_boundary = q3 + (1.5 \* iqr)

upper\_boundary

# Check for any lower outliers

arr[arr <= lower\_boundary]

# Check for any upper outliers

arr[arr >= upper\_boundary]

# Create box plot

plt.boxplot(arr, showmeans=True)

plt.grid()

plt.show()

**05-Matplotlib - Session 3 - Activity - 05-Ins\_Standard\_Error 3 days ago**

# Dependencies

from random import random

import matplotlib.pyplot as plt

import numpy as np

from scipy.stats import sem

# "Will you vote for a republican in this election?"

sample\_size = 100

samples = [[True if random() < 0.5 else False for x in range(0, sample\_size)]

for y in range(0, 10)]

x\_axis = np.arange(0, len(samples), 1)

means = [np.mean(s) for s in samples]

standard\_errors = [sem(s) for s in samples]

# Setting up the plot

fig, ax = plt.subplots()

ax.errorbar(x\_axis, means, standard\_errors, fmt="o")

ax.set\_xlim(-1, len(samples) + 1)

ax.set\_xlabel("Sample Number")

ax.set\_ylabel("Proportion of People Voting Republican")

plt.show()

**05-Matplotlib - Session 3 - Activity - 06-Stu\_Standard\_Error 3 days ago**

%matplotlib notebook

# Dependencies

from matplotlib import pyplot as plt

import numpy as np

import pandas as pd

# Read data

housing\_data = pd.read\_csv("../Resources/housing\_data.csv")

housing\_data = housing\_data.sample(frac=1).reset\_index(drop=True)

# Create a bunch of samples, each with div items

div = 20

lim = len(housing\_data) // div

samples = [housing\_data.iloc[(i \* div):(i \* div + div), 13]

for i in range(0, lim)]

# Calculate means

means = [s.mean() for s in samples]

# Calculate standard error on means

sem = [s.sem() for s in samples]

# Plot sample means with error bars

fig, ax = plt.subplots()

ax.errorbar(np.arange(0, len(means)), means, yerr=sem, fmt="o", color="b",

alpha=0.5, label="Mean of House Prices")

ax.set\_xlim(-0.5, len(means))

ax.set\_xlabel("Sample Number")

ax.set\_ylabel("Mean of Median House Prices")

plt.legend(loc="best", fontsize="small", fancybox=True)

plt.show()

**05-Matplotlib - Session 3 - Activity - 07-Ins\_Students\_t\_test 3 days ago**

%matplotlib notebook

# Dependencies

from random import randint

import matplotlib.pyplot as plt

import numpy as np

from scipy.stats import sem, ttest\_ind

# Generate

high\_prices = [randint(1, 5) \* 1000 for x in range(1, 10)]

high\_means = np.mean(high\_prices)

high\_sem = sem(high\_prices)

low\_prices = [randint(1, 5) \* 200 for x in range(1, 10)]

low\_means = np.mean(low\_prices)

low\_sem = sem(low\_prices)

means = [high\_means, low\_means]

sems = [high\_sem, low\_sem]

labels = ["High Prices", "Low Prices"]

# Plot

fig, ax = plt.subplots()

ax.errorbar(np.arange(0, len(means)), means, yerr=sems, fmt="o")

ax.set\_xlim(-0.5, 2.5)

ax.set\_xticklabels(labels)

ax.set\_xticks([0, 1, 2])

ax.set\_ylabel("Mean House Price")

plt.show()

# t-test

(t\_stat, p) = ttest\_ind(high\_prices, low\_prices, equal\_var=False)

if p < 0.05:

print("The differences between the high and low prices are significant.")

else:

print("The differences between high and low prices are due to chance.")

**05-Matplotlib - Session 3 - Activity - 08-Stu\_Students\_t\_test 3 days ago**

%matplotlib notebook

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from scipy import stats

# Read in data

general\_heights = pd.read\_csv("../Resources/general\_heights.csv")

wba\_data = pd.read\_csv("../Resources/wba\_data.csv")

wba\_heights = wba\_data.iloc[:, -1]

# Run the t-test

(t\_stat, p) = stats.ttest\_ind(general\_heights, wba\_heights, equal\_var=False)

# Report the data

print("The mean height of WBA players is {}.".format(wba\_heights.mean()))

print("The mean height of women sampled is {}.".format(

general\_heights.values.mean()))

print("p is {}.".format(p[0]))

if p < 0.05:

print("The difference in sample means is significant.")

else:

print("The difference in sample means is not significant.")

# Plot sample means with error bars

tick\_labels = ["General Public", "WBA Players"]

means = [general\_heights.mean().values[0], wba\_heights.mean()]

x\_axis = np.arange(0, len(means))

sem = [general\_heights.sem().values[0], wba\_heights.sem()]

# Plot mean height of players

fig, ax = plt.subplots()

fig.suptitle("Mean Height of Women in General Population and WBA Players",

fontsize=12, fontweight="bold")

ax.errorbar(x\_axis, means, yerr=sem, fmt="o")

ax.set\_xlim(-0.5, 1.5)

ax.set\_ylim(64, 73)

ax.set\_xticklabels(tick\_labels)

ax.set\_xticks([0, 1])

ax.set\_ylabel("Height (Inches)")

plt.show()

**05-Matplotlib - Session 3 - Activity - 09-Ins\_Fits\_and\_Regression 3 days ago**

# Dependencies

from matplotlib import pyplot as plt

from scipy.stats import linregress

import numpy as np

# Set data

x\_axis = np.arange(0, 10, 1)

fake = [1, 2.5, 2.75, 4.25, 5.5, 6, 7.25, 8, 8.75, 9.8]

# Set line

(slope, intercept, \_, \_, \_) = linregress(x\_axis, fake)

fit = slope \* x\_axis + intercept

# Plot data

fig, ax = plt.subplots()

fig.suptitle("Fake Banana Data!", fontsize=16, fontweight="bold")

ax.set\_xlim(0, 10)

ax.set\_ylim(0, 10)

ax.set\_xlabel("Fake Banana Ages (in days)")

ax.set\_ylabel("Fake Banana Weights (in Hundres of Kilograms)")

ax.plot(x\_axis, fake, linewidth=0, marker='o')

ax.plot(x\_axis, fit, 'b--')

plt.show()

**05-Matplotlib - Session 3 - Activity - 10-Stu\_Fits\_and\_Regression**

%matplotlib notebook

# Dependencies

from matplotlib import pyplot as plt

from scipy import stats

import numpy as np

import pandas as pd

# Load data

crime\_data = pd.read\_csv("../Resources/crime\_data.csv")

year = crime\_data.iloc[:, 0]

# Grab violent crime rates

violent\_crime\_rate = crime\_data.iloc[:, 3]

vc\_slope, vc\_int, vc\_r, vc\_p, vc\_std\_err = stats.linregress(

year, violent\_crime\_rate)

vc\_fit = vc\_slope \* year + vc\_int

# Grab murder rate

murder\_rate = crime\_data.iloc[:, 5]

m\_slope, m\_int, m\_r, m\_p, m\_std\_err = stats.linregress(year, murder\_rate)

m\_fit = m\_slope \* year + m\_int

# Grab aggravated assault rate

aggravated\_assault\_rate = crime\_data.iloc[:, 9]

aa\_slope, aa\_int, aa\_r, aa\_p, aa\_std\_err = stats.linregress(

year, aggravated\_assault\_rate)

aa\_fit = aa\_slope \* year + aa\_int

# Plot

fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True)

fig.suptitle("Crime Rates Over Time", fontsize=16, fontweight="bold")

ax1.set\_xlim(min(year), max(year))

ax1.plot(year, violent\_crime\_rate, linewidth=1, marker="o")

ax1.plot(year, vc\_fit, "b--", linewidth=1)

ax1.set\_ylabel("Violent Crime Rate")

ax2.plot(year, murder\_rate, linewidth=1, marker="o", color="r")

ax2.plot(year, m\_fit, "r--", linewidth=1)

ax2.set\_ylabel("Murder Rate")

ax3.plot(year, aggravated\_assault\_rate, linewidth=1, marker="o", color="g")

ax3.plot(year, aa\_fit, "g--", linewidth=1)

ax3.set\_ylabel("Aggravated Assault Rate")

ax3.set\_xlabel("Year")

# Print results and save image

year = 2019

print("The violent crime rate in 2019 will be " +

str(vc\_slope \* year + vc\_int) + ".")

print("The murder rate in 2019 will be " + str(m\_slope \* year + m\_int) + ".")

print("The aggravated assault rate in 2019 will be " +

str(aa\_slope \* year + aa\_int) + ".")

plt.savefig("../Images/18-final-plot.png")