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")
```