# Module 5 – Data Visualisation

**Day 1, Activity 1 🡪 Exponential Chart**

# Import Numpy for calculations and matplotlib for charting

import numpy as np

import matplotlib.pyplot as plt

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

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

**A black background with white numbers

Description automatically generated**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

**A screenshot of a computer

Description automatically generated**

**A blue line graph with numbers

Description automatically generated**

# Create a graph based upon the list and array 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")

**A graph with a blue line

Description automatically generated**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()



**Day 1, Activity 1 🡪 sin\_cos**

# Import Numpy for calculations and matplotlib for charting

import numpy as np

import matplotlib.pyplot as plt

# Create our x\_axis numpy array

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

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

sin = np.sin(x\_axis)

# Creates a numpy array 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()

**A graph of a function

Description automatically generated with medium confidence**

**Day 1, Activity 2 🡪 Melb Temperature**

# Set x axis to numerical value for month

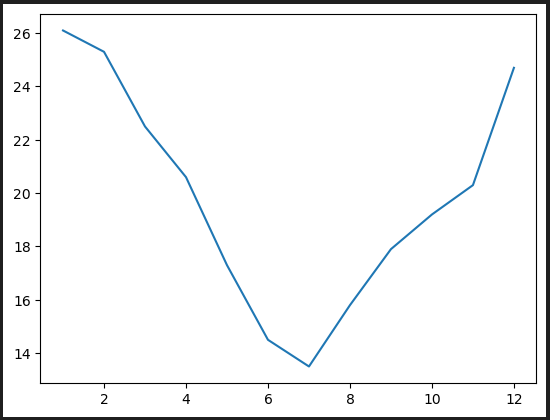
x\_axis\_data = np.arange(1,13,2)

x\_axis\_data

****

# Average weather temp

points = [26.1, 25.3, 22.5, 20.6, 17.3, 14.5, 13.5, 15.8, 17.9, 19.2, 20.3, 24.7]

****

# Plot the line

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

plt.show()

# Convert to Fahrenheit F = C \* 9/5 + 32

points\_F = [round(x \* 9 / 5 + 32,1) for x in points]

****points\_F

**A graph with a line

Description automatically generated**# Plot using Fahrenheit

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

plt.show()



**A graph showing the difference between two different colored lines

Description automatically generated**



# Plot both on the same chart

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

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

plt.show()

**Day 1, Activity 3 🡪 Line Config**

# 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

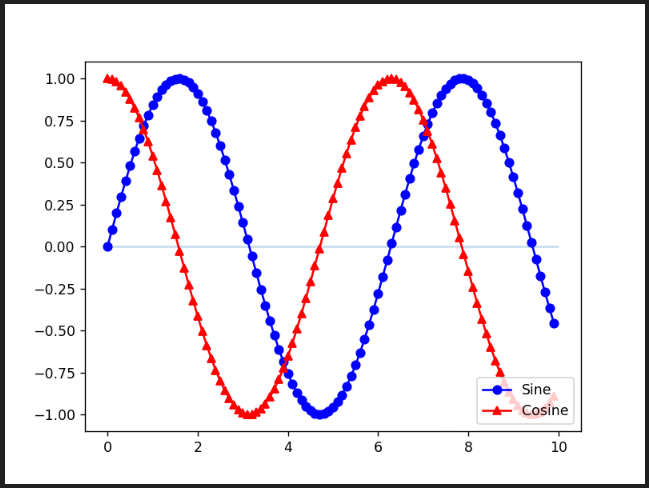
cosine\_handle, = plt.plot(x\_axis, cos, marker='^', color='red', label="Cosine")

plt.show()

# Adds a legend and sets its location to the lower right

plt.legend(loc="lower right")

plt.show()

****

# Saves an image of our chart so that we can view it in a folder

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

plt.show()

**Day 1, Activity 4 🡪 Legendary Temp**

# Include this line to make plots interactive

%matplotlib widget

# 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\_C = [26.1, 25.3, 22.5, 20.6, 17.3, 14.5, 13.5, 15.8, 17.9, 19.2, 20.3, 24.7]

# Convert to Fahrenheit F = C \* 9/5 + 32

points\_F = [round(x \* 9 / 5 + 32,1) for x in points\_C]

points\_F

****

# Create a handle for each plot

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

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

plt.show()

**A graph with a blue line and red dots

Description automatically generated**

# Set our legend to where the chart thinks is best

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

plt.show()

**A graph with numbers and symbols

Description automatically generated with medium confidence**

# Create labels for the X and Y axis

plt.xlabel("Months")

plt.ylabel("Degrees")

plt.show()

**A graph with numbers and a line

Description automatically generated with medium confidence**

# Save and display the chart

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

plt.show()

**Day 1, Activity 5 🡪 Aesthetics**

# 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)

plt.show()

# 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")

plt.show()

**A graph of a function

Description automatically generated**

**A graph of a function

Description automatically generated**# 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)")

plt.show()

# Set your x and y limits

plt.xlim(0, 10)

plt.ylim(-1, 1)

plt.show()

**A white rectangular object with black lines

Description automatically generated**

# Set a grid on the plot

plt.grid()

plt.show()

**A graph with lines in the center

Description automatically generated**

# Save the plot and display it

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

plt.show()

**Day 1, Activity 6 🡪 Coaster Speed**

# Import Dependencies

import matplotlib.pyplot as plt

import numpy as np

# Create the X and Y axis lists

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

danger\_drop\_speeds = [15, 13, 145, 137, 129, 113, 113, 105, 89, 97, 113, 105, 81]

railgun\_speeds = [121, 113, 97, 105, 97, 73, 89, 81, 64, 64, 56, 56, 48]

# Plot the charts and apply some styling

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

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

# Set the limits for the X and Y axes

plt.xlim(0,120)

plt.ylim(5,150)

# Add in a grid for the chart

plt.grid()

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

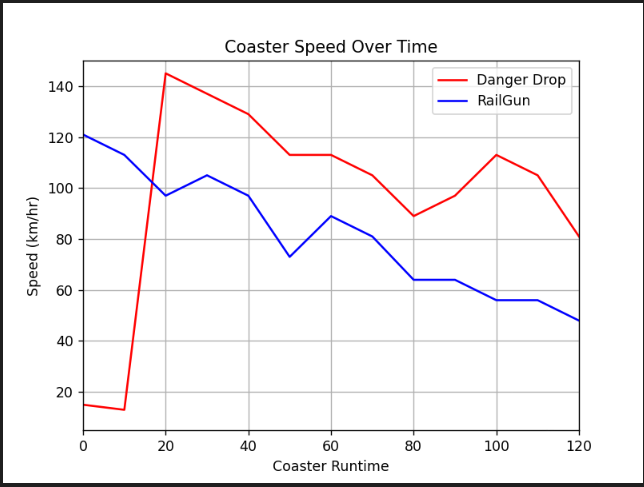
plt.title("Coaster Speed Over Time")

plt.xlabel("Coaster Runtime")

plt.ylabel("Speed (km/hr)")

# Create a legend for the chart

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

plt.show()****

# Save figure

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

plt.show()

**Day 1, Activity 7 🡪 Bar Chart**

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="center" 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")

plt.show()

**A graph of pink bars

Description automatically generated**

**Day 1, Activity 8 🡪 Bar Chart**

import matplotlib.pyplot as plt

import numpy as np

cities = ["San Francisco", "Omaha", "New Orleans", "Cincinnati", "Pittsburgh"]

cars\_in\_cities = [214.7, 564.4, 416.5, 466.7, 350.6]

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

# Create a bar chart based upon the above data

plt.bar(x\_axis, cars\_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(cars\_in\_cities)+10)

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

plt.title("Density of Commuting Cars in Cities")

plt.xlabel("Cities")

plt.ylabel("Commuting Cars Per 1,000 Population Age 16+")

plt.show()

**A graph of blue rectangular bars

Description automatically generated with medium confidence**

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

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

plt.show()

**Day 1, Activity 9 🡪 Pie Charts**

# 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 colours of each section of the pie chart

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

# Tells matplotlib to separate the "Humans" 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=colours,

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

plt.show()

**A pie chart with different colored circles

Description automatically generated**

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

plt.axis("equal")

plt.show()

**Day 1, Activity 10 🡪 Pie Charts**

import matplotlib.pyplot as plt

import numpy as np

pies = ["Aussie Beef", "Apple", "Chicken", "Lemon Meringue", "Shepherds", "Steak and Kidney", "Curry", "Lamb and Rosemary", "Steak and Mushroom", "Spinach and Feta"]

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

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

# Add explode levels for each value

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=colours,

        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()

**A pie chart with different colored circles

Description automatically generated**

**Day 1, Activity 11 🡪 Scatter Plot**

# 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)

plt.show()

**A diagram of red dots

Description automatically generated**

**Day 1, Activity 12 🡪 Scatter Plot – Ice Cream Sales**

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)

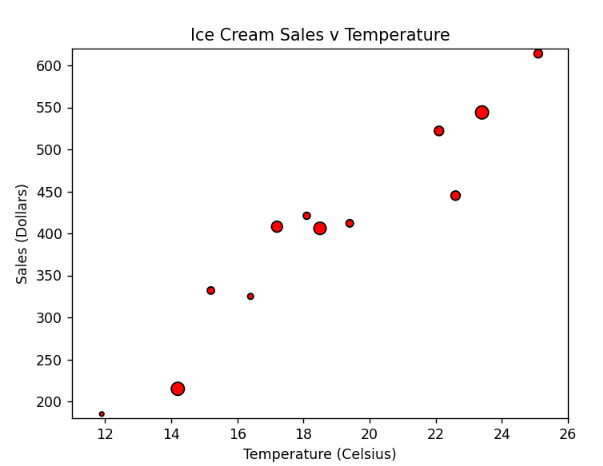
# Create a title, x label, and y label for our chart

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

plt.xlabel("Temperature (Celsius)")

plt.ylabel("Sales (Dollars)")

plt.show()

****

# 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()

**Day 1, Activity 13 🡪 Avg Cities Rain (using Pandas)**

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

**A black and white screen with white text

Description automatically generated**import pandas as pd

# Load in csv

rain\_df = pd.read\_csv("../Resources/avg\_rain\_cities.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=(8,5))

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

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

# Set x and y limits

plt.xlim(-0.15, len(x\_axis)-0.05)

plt.ylim(0, max(rain\_df["ANNUAL (mm)"])+50)

# Set a Title and labels

plt.title("Average Rain in Major Cities")

plt.xlabel("City")

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

plt.show()

**A graph of rain in major cities

Description automatically generated**

# Save our graph and show the grap

plt.tight\_layout()

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

plt.show()

**Day 2, Activity 1 🡪 Plot Drills - Gym**

# Import Dependencies

import numpy as np

import matplotlib.pyplot as plt

# DATA SET 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")

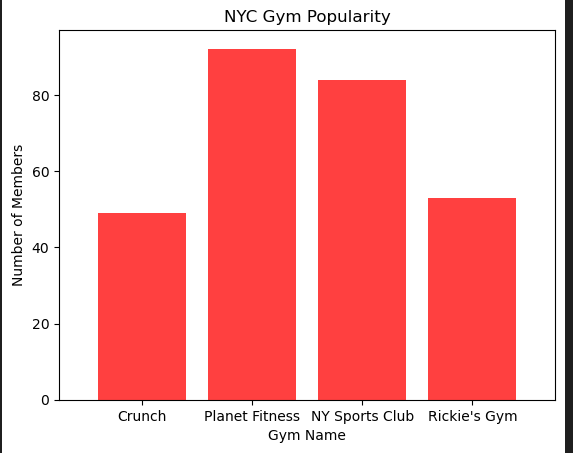
plt.xlim(-0.75, len(gyms)-.25)

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()

****

# DATA SET 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()

**A graph of a function

Description automatically generated**

# DATA SET 3

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

members = [49, 92, 84, 53]

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

explode = (0, 0.05, 0, 0)

**A pie chart with different colored circles with Crust in the background

Description automatically generated**

plt.title("NYC Gym Popularity")

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

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

plt.axis("equal")

plt.show()

# DATA SET 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))))

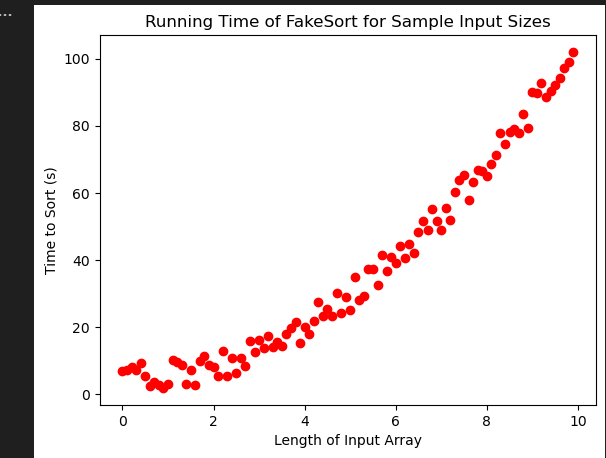
plt.title("Running Time of FakeSort for Sample Input Sizes")

plt.xlabel("Length of Input Array")

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

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

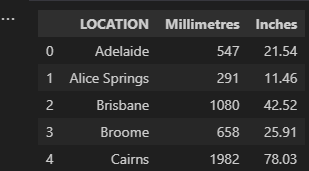
plt.show()

****

**Day 2, Activity 2 – Avg State Rain**

# Dependencies

import matplotlib.pyplot as plt

****import numpy as np

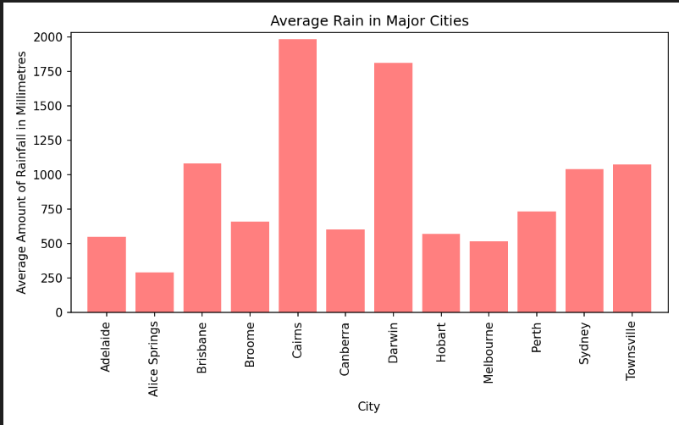
import pandas as pd

#Using MatpltLib to Chart a DataFrame

# Load in csv

rain\_df = pd.read\_csv("../Resources/avg\_rain\_cities.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=(8,5))

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

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

# Set x and y limits

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

plt.ylim(0, max(rain\_df["Millimetres"])+50)

… (0.0, 2032.0)

# Set a Title and labels

plt.title("Average Rain in Major Cities")

plt.xlabel("City")

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

**…** Text(0, 0.5, 'Average Amount of Rainfall in Millimetres')

# Save our graph and show the grap

plt.tight\_layout()

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

plt.show()

… <Figure size 640X180 with 0 Axes>

**Using Pandas to Chart a DataFrame**

# Filter the DataFrame down only to those columns to chart

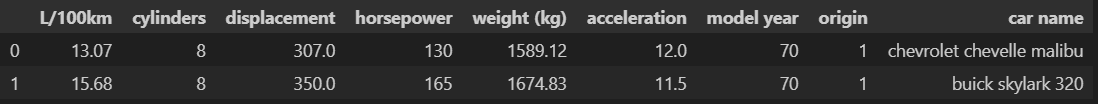
city\_and\_mm = rain\_df[["LOCATION","Millimetres"]]

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

city\_and\_mm = city\_and\_mm.set\_index("LOCATION")

**A screenshot of a computer

Description automatically generated**city\_and\_mm.head()

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

city\_and\_mm.plot(kind="bar", figsize=(6,5))

**A graph of rain in a city

Description automatically generated with medium confidence**

# Set a title for the chart

plt.title("Average Rain Per City")

plt.show()

plt.tight\_layout()

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

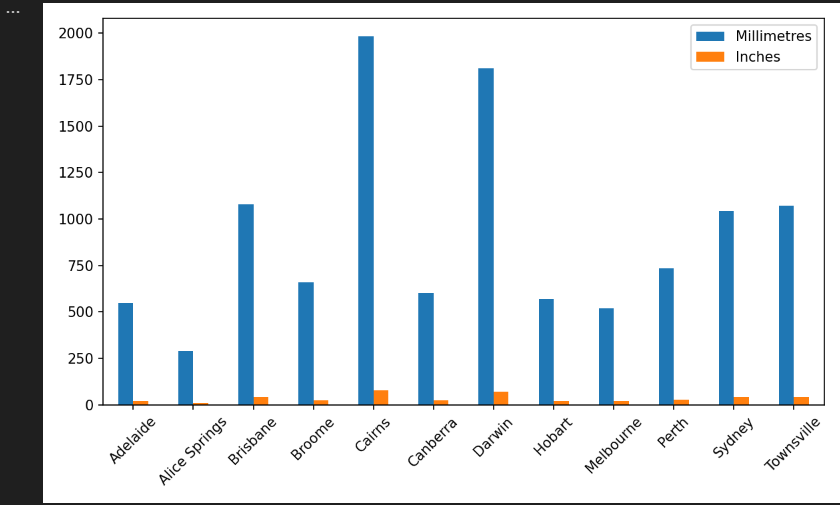
multi\_plot = rain\_df.plot(kind="bar", figsize=(8,5))

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

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

plt.show()

plt.tight\_layout()

****

**Day 2, Activity 3**

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

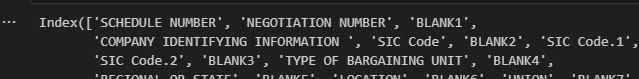
import pandas as pd

# Read CSV

settlement\_data = pd.read\_csv("Resources/union\_settlements\_1995.csv")

settlement\_data.head()

# List columns

****settlement\_data.columns

# Get total settlements by union

union\_data = settlement\_data["UNION"].value\_counts()

# Configure plot, figsize, title, and axis labels

figure1 = union\_data.plot(kind="bar", facecolor="red", figsize=(8,6),

                                title="Major Collective Bargaining Settlements (1995)")

figure1.set\_xlabel("Union")

figure1.set\_ylabel("Settlements")

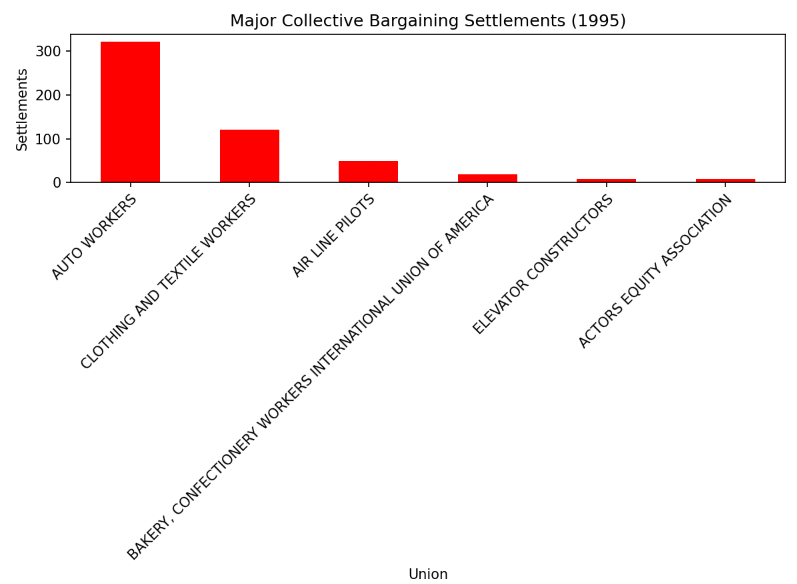
# Configure x-tick rotation

xticklabels = union\_data.index

figure1.set\_xticklabels(xticklabels, rotation=45, rotation\_mode="anchor", ha="right", wrap=True)

# Show plot

plt.show()

****

# Resize plot to display labels

plt.tight\_layout()

**BONUS Question**

# Get total settlements by union

union\_data = settlement\_data["UNION"].value\_counts()

# Filter data to national settlements

national\_settlements = settlement\_data.loc[settlement\_data["LOCATION"]=="NATIONAL",:]

national\_settlements

# Get national settlements by union

national\_union\_data = national\_settlements["UNION"].value\_counts()

# Create new dataframe for data to chart (has NaN)

compare\_all\_national\_df = pd.DataFrame({"National Settlements": national\_union\_data,

                                       "All Settlements": union\_data})

compare\_all\_national\_df

# Fill NA

compare\_all\_national\_df = compare\_all\_national\_df.fillna(0)

compare\_all\_national\_df

# Configure plot, figsize, title, and axis labels

figure1 = compare\_all\_national\_df.plot(kind="bar", color=["red", "blue"], figsize=(8,6),

                                   title="Major Collective Bargaining Settlements (1995)")

figure1.set\_xlabel("Union")

figure1.set\_ylabel("Settlements")

# Configure x-tick rotation

xticklabels = compare\_all\_national\_df.index

figure1.set\_xticklabels(xticklabels, rotation=45, rotation\_mode="anchor", ha="right", wrap=True)

# Show plot

plt.show()

**A graph with red and blue bars

Description automatically generated**

**Activity 4 – Ins GroupPlots**

# Import Dependencies

import matplotlib.pyplot as plt

import pandas as pd

# Import our data into pandas from CSV

accident\_string = '../Resources/accidents.csv'

accidents\_df = pd.read\_csv(accident\_string, low\_memory=False)

accidents\_df

# Create a group based on the values in the 'FUNC\_SYSNAME' column

# 'FUNC\_SYSNAME' stores the type of road the accident occurred

accident\_road\_type = accidents\_df.groupby('FUNC\_SYSNAME')

# Count how many times each road type appears in our group

count\_road\_types = accident\_road\_type['FUNC\_SYSNAME'].count()

**A screen shot of a computer

Description automatically generated**count\_road\_types

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

count\_chart = count\_road\_types.plot(kind='bar', figsize=(6,8))

# Set the xlabel and ylabel using class methods

count\_chart.set\_xlabel("Road Type")

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

plt.show()

plt.tight\_layout()

**A graph with blue bars

Description automatically generated**

**Day 2, Activity 5 – Student Library Usage – Groupby (Pie chart)**

# 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/library\_usage.csv'

library\_usage\_df = pd.read\_csv(string\_thing, low\_memory=False)

library\_usage\_df.head()

# Filter data so it only includes patrons who checked out at least one item

library\_loans\_df = pd.DataFrame(library\_usage\_df.loc[library\_usage\_df['Total Checkouts']>0,:])

# Split up our data into groups based upon 'Patron Type Definition'

patron\_groups = library\_loans\_df.groupby('Patron Type Definition')

# Find out how many of each patron type borrowed library items

patron\_borrows = patron\_groups['Total Checkouts'].count()

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

patron\_chart = patron\_borrows.plot(kind="bar", title="Library Usage by Patron Type")

patron\_chart.set\_xlabel("Patron Type")

patron\_chart.set\_ylabel("Number of Patrons Borrowing Items")

plt.show()

plt.tight\_layout()

**A graph of a number of users

Description automatically generated**

# Split up our data into groups based upon 'Home Library Definition' and 'Patron Type Definition'

library\_groups = library\_usage\_df.groupby(['Home Library Definition','Patron Type Definition'])

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

sum\_it\_up = library\_groups[['Total Checkouts']].sum()

sum\_it\_up.head(20)

# Make a variable called branch and store a 'Home Library Definition' in it

branch = "Anza"

# Make a variable called min\_checkouts that you can change depending on how busy the library branch you've chosen is

min\_checkouts = 5000

# Collect the loans of the branch above

just\_one\_branch = sum\_it\_up.loc[branch]

# filter the data to patron types with greater than the value set for min\_checkouts

just\_one\_branch = just\_one\_branch.loc[just\_one\_branch['Total Checkouts']>min\_checkouts,:]

# Create a pie chart based upon the total checkouts (or loans) of that single branch

branch\_pie = just\_one\_branch.plot(kind="pie", y='Total Checkouts', title=("Loans of " + branch +

                                                                          " Branch for Patron Types Over "

                                                                         + str(min\_checkouts) + " Loaned Items"))

branch\_pie.set\_ylabel("Branch Checkouts")

plt.axis("equal")

**A pie chart with different colors and text

Description automatically generated**plt.show()

**Day 2, Activity 6 – Student L Per 100km-ScatterPlot**

# Dependencies and Setup

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# Import our data into pandas from CSV

car\_data = pd.read\_csv('../Resources/Lp100km.csv')

car\_data.head()

A screenshot of a black screen

Description automatically generated

# Remove the rows with missing values in horsepower

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

**A screenshot of a black screen

Description automatically generated**car\_data.head()

**index 2 got  
 removed**

# 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()

**A screenshot of a computer

Description automatically generated**

# 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 L/100km to horsepower

**A graph of blue dots

Description automatically generated**car\_data.plot(kind="scatter", x="horsepower", y="L/100km", grid=True, figsize=(8,8),

              title="L/100km Vs. Horsepower")

plt.show()

**Day 2, Activity 7 – Instructor Pandas Multi Line**

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# Read CSV

unemployed\_data\_one = pd.read\_csv("../Resources/unemployment\_2010-2015.csv")

unemployed\_data\_two = pd.read\_csv("../Resources/unemployment\_2016-2020.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")

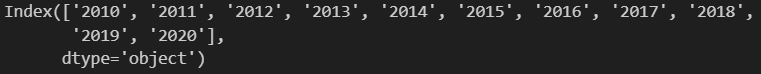
# Collect the mean unemployment rates for the world

average\_unemployment = combined\_unemployed\_data.select\_dtypes(include=np.number).mean()

# Collect the years where data was collected

years = average\_unemployment.keys()

years

****

# 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['AUS',["2010","2011","2012","2013","2014","2015",

                                                                  "2016","2017","2018","2019","2020"]],

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

# Create a legend for our chart

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

# Show the chart

plt.show()

**A graph with a line and a line

Description automatically generated**

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

combined\_unemployed\_data.loc['AUS', "2010":"2020"].plot(label="Australia")

plt.legend()

plt.show()

**A graph with blue and orange lines

Description automatically generated**

**Day 2, Activity 8 – Student Travel Part 1**

**# Travelling Companions**

In this activity you will be taking three separate csvs that were gathered by Tourism Malaysia, merging them together, and then creating charts to visualise a country's change of travelling companions to Malaysia over the course of three years.

**### Part 1 - Merging Companions**

\* You will likely need to perform two different merges over the course of this activity, changing the names of your columns as you go along.

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

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

travel\_2016 = "../Resources/2016\_travellers.csv"

travel\_2017 = "../Resources/2017\_travellers.csv"

travel\_2018 = "../Resources/2018\_travellers.csv"

travel\_2016\_df = pd.read\_csv(travel\_2016)

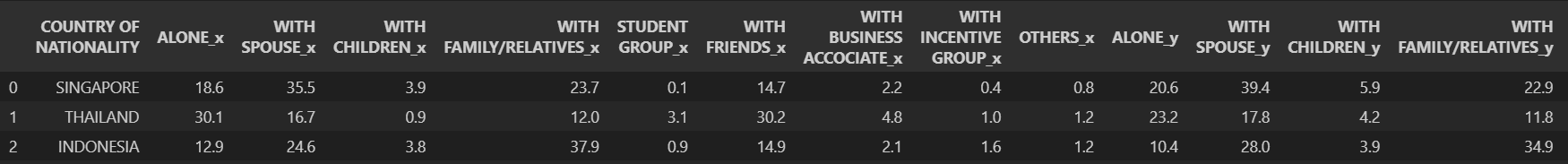
travel\_2017\_df = pd.read\_csv(travel\_2017)

travel\_2018\_df = pd.read\_csv(travel\_2018)

# Merge the first two datasets on "COUNTRY OF NATIONALITY" so that no data is lost (should be 44 rows)

combined\_travel\_df = pd.merge(travel\_2016\_df, travel\_2017\_df,

                                 how='outer', on='COUNTRY OF NATIONALITY')

****combined\_travel\_df.head()

# Rename our \_x columns to "2016 Alone", "2016 With Spouse", "2016 With Children", "2016 With Family/Relatives",

# "2016 Student Group", "2016 With Friends", "2016 With Business Associate", "2016 With Incentive Group",

# and "2016 Others"

combined\_travel\_df = combined\_travel\_df.rename(columns={"ALONE\_x":"2016 Alone",

                                                        "WITH SPOUSE\_x":"2016 With Spouse",

                                                        "WITH CHILDREN\_x":"2016 With Children",

                                                        "WITH FAMILY/RELATIVES\_x":"2016 With Family/Relatives",

                                                        "STUDENT GROUP\_x":"2016 Student Group",

                                                        "WITH FRIENDS\_x":"2016 With Friends",

                                                        "WITH BUSINESS ACCOCIATE\_x":"2016 With Business Associate",

                                                        "WITH INCENTIVE GROUP\_x":"2016 With Incentive Group",

                                                        "OTHERS\_x":"2016 Others"})

# Rename our \_y columns to "2016 Alone", "2016 With Spouse", "2016 With Children", "2016 With Family/Relatives",

# "2016 Student Group", "2016 With Friends", "2016 With Business Associate", "2016 With Incentive Group",

# and "2016 Others"

combined\_travel\_df = combined\_travel\_df.rename(columns={"ALONE\_y":"2017 Alone",

                                                        "WITH SPOUSE\_y":"2017 With Spouse",

                                                        "WITH CHILDREN\_y":"2017 With Children",

                                                        "WITH FAMILY/RELATIVES\_y":"2017 With Family/Relatives",

                                                        "STUDENT GROUP\_y":"2017 Student Group",

                                                        "WITH FRIENDS\_y":"2017 With Friends",

                                                        "WITH BUSINESS ACCOCIATE\_y":"2017 With Business Associate",

                                                        "WITH INCENTIVE GROUP\_y":"2017 With Incentive Group",

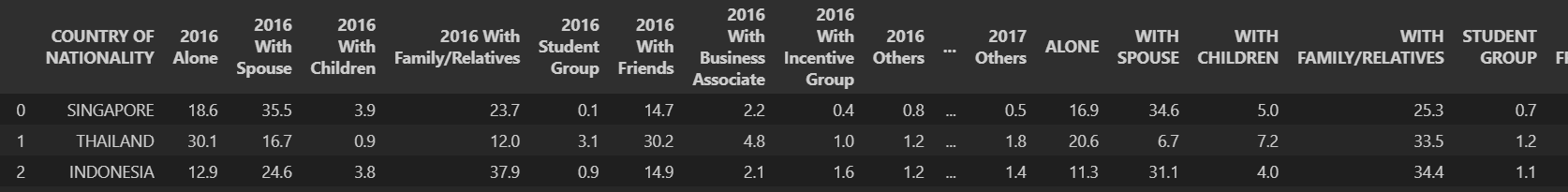
                                                        "OTHERS\_y":"2017 Others"})

**A screen shot of a black and white screen

Description automatically generated**combined\_travel\_df.head()

# Merge our newly combined dataframe with the 2018 dataframe

combined\_travel\_df = pd.merge(combined\_travel\_df, travel\_2018\_df, how="outer", on="COUNTRY OF NATIONALITY")

****combined\_travel\_df

# Rename "ALONE", "WITH SPOUSE", "WITH CHILDREN", "WITH FAMILY/RELATIVES", "STUDENT GROUP", "WITH FRIENDS",

# "WITH BUSINESS ACCOCIATE","WITH INCENTIVE GROUP", "OTHERS" to

# "2018 Alone", "2018 With Spouse", "2018 With Children", "2018 With Family/Relatives", "2018 Student Group",

# "2018 With Friends", "2018 With Business Associate", "2018 With Incentive Group", and "2018 Others"

combined\_travel\_df = combined\_travel\_df.rename(columns={"ALONE":"2018 Alone",

                                                        "WITH SPOUSE":"2018 With Spouse",

                                                        "WITH CHILDREN":"2018 With Children",

                                                        "WITH FAMILY/RELATIVES":"2018 With Family/Relatives",

                                                        "STUDENT GROUP":"2018 Student Group",

                                                        "WITH FRIENDS":"2018 With Friends",

                                                        "WITH BUSINESS ACCOCIATE":"2018 With Business Associate",

                                                        "WITH INCENTIVE GROUP":"2018 With Incentive Group",

                                                        "OTHERS":"2018 Others"})

**A screenshot of a black screen

Description automatically generated**combined\_travel\_df.head()

**### Part 2**

# Check the mean of the columns

combined\_travel\_df.select\_dtypes(include=np.number).mean()

**A screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generated**

# Reduce columns where mean of travelling companions is > 1 across all years

travel\_reduced = pd.DataFrame(combined\_travel\_df[["COUNTRY OF NATIONALITY",

                                                  "2016 Alone","2016 With Spouse","2016 With Children",

                                                  "2016 With Family/Relatives","2016 With Friends",

                                                  "2016 With Business Associate","2017 Alone",

                                                  "2017 With Spouse","2017 With Children",

                                                  "2017 With Family/Relatives","2017 With Friends",

                                                  "2017 With Business Associate","2018 Alone",

                                                  "2018 With Spouse","2018 With Children",

                                                  "2018 With Family/Relatives","2018 With Friends",

                                                  "2018 With Business Associate"]])

# Set index to "Country of Nationality"

travel\_reduced = travel\_reduced.set\_index("COUNTRY OF NATIONALITY")

**A screenshot of a computer

Description automatically generated**travel\_reduced

**Part 3**

# Create a variable for each country to chart

country1 = "AUSTRALIA"

country2 = "THAILAND"

country3 = "PHILIPPINES"

# Set type of travelling companion

columns\_to\_compare = "With Spouse"

# Create a Series for each chosen country that looks for the chosen travel companion from 2016 to 2018

country1\_traveller\_over\_time = travel\_reduced.loc[country1,

                                                [f"2016 {columns\_to\_compare}",

                                                 f"2017 {columns\_to\_compare}",

                                                 f"2018 {columns\_to\_compare}"]]

country2\_traveller\_over\_time = travel\_reduced.loc[country2,

                                                [f"2016 {columns\_to\_compare}",

                                                 f"2017 {columns\_to\_compare}",

                                                 f"2018 {columns\_to\_compare}"]]

country3\_traveller\_over\_time = travel\_reduced.loc[country3,

                                                [f"2016 {columns\_to\_compare}",

                                                 f"2017 {columns\_to\_compare}",

                                                 f"2018 {columns\_to\_compare}"]]

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

years = [2016,2017,2018]

# Plot our line that will be used to track the first country's travelling companion percentage over the years

plt.plot(years, country1\_traveller\_over\_time, color="green", label=country1)

# Plot our line that will be used to track the second country's travelling companion percentage over the years

plt.plot(years, country2\_traveller\_over\_time, color="blue", label=country2)

# Plot our line that will be used to track the third country's travelling companion percentage over the years

plt.plot(years, country3\_traveller\_over\_time, color="orange", label=country3)

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

plt.legend(loc="best")

plt.title("Travelling " + columns\_to\_compare + " Country Comparison")

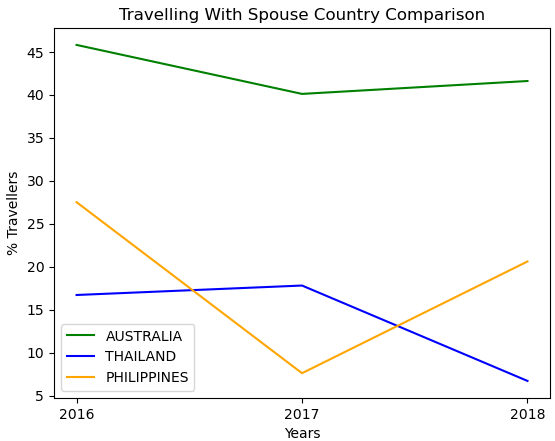
plt.xlabel("Years")

plt.xticks(np.arange(min(years), max(years)+1, 1.0))

plt.ylabel("% Travellers")

# Print our chart to the screen

plt.show()

****

**### Part 3 - Charting Travelling Companions --bonus**

\* Create 3 variables with inputs that asks the user what country they would like to chart.

\* Ask the user what type of travelling companion they would like to compare for their chosen countries.

\* Store each country's percentage of travellers for the chosen travelling companion over time in 3 variables (one for each country)

\* Create a line chart that will plot the comparison of each country's percentage of travellers with the chosen traveling companion from 2016 to 2018

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

country1 = input("What country would you like to chart 1st? ")

country2 = input("What country would you like to chart 2nd? ")

country3 = input("What country would you like to chart 3rd? ")

**A black background with white text

Description automatically generated**

Manual entry for input data! – Belgium / France / Australia

# Ask type of travelling companion

select\_options = "1 - Alone\n2 - With Spouse\n3 - With Children\n4 - With Family/Relatives\n5 - With Friends\n6 - With Business Associate"

print("What type of travelling companion would you like to chart? ")

print(select\_options)

traveller\_type = int(input("Choose a number:  "))

# Boolean to check if number was correctly chosen

selected = False

# Loop through options to set column name according to chosen number

while selected == False:

    if traveller\_type == 1:

        columns\_to\_compare = "Alone"

        selected = True

    elif traveller\_type == 2:

        columns\_to\_compare = "With Spouse"

        selected = True

    elif traveller\_type == 3:

        columns\_to\_compare = "With Children"

        selected = True

    elif traveller\_type == 4:

        columns\_to\_compare = "With Family/Relatives"

        selected = True

    elif traveller\_type == 5:

        columns\_to\_compare = "With Friends"

        selected = True

    elif traveller\_type == 6:

        columns\_to\_compare = "With Business Associate"

        selected = True

    else:

        # Incorrect input, try again

        print("Please make your selection again.")

        print(select\_options)

        traveller\_type = int(input("Choose a number:  "))

print("You selected option " + str(traveller\_type) + " - " + columns\_to\_compare)

**A screen shot of a computer

Description automatically generated**

Manual entry for input data! 1,2,3,4,5,6

# Create a Series for each chosen country that looks for the chosen travel companion from 2016 to 2018

country1\_traveller\_over\_time = travel\_reduced.loc[country1.upper(),

                                                [f"2016 {columns\_to\_compare}",f"2017 {columns\_to\_compare}", f"2018 {columns\_to\_compare}"]]

country2\_traveller\_over\_time = travel\_reduced.loc[country2.upper(),

                                                [f"2016 {columns\_to\_compare}",f"2017 {columns\_to\_compare}", f"2018 {columns\_to\_compare}"]]

country3\_traveller\_over\_time = travel\_reduced.loc[country3.upper(),

                                                [f"2016 {columns\_to\_compare}",f"2017 {columns\_to\_compare}", f"2018 {columns\_to\_compare}"]]

**A screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated**

print information for each country

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

years = [2016,2017,2018]

# Plot our line that will be used to track the first country's travelling companion percentage over the years

plt.plot(years, country1\_traveller\_over\_time, color="green", label=country1)

# Plot our line that will be used to track the second country's travelling companion percentage over the years

plt.plot(years, country2\_traveller\_over\_time, color="blue", label=country2)

# Plot our line that will be used to track the third country's travelling companion percentage over the years

plt.plot(years, country3\_traveller\_over\_time, color="orange", label=country3)

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

plt.legend(loc="best")

plt.title("Travelling " + columns\_to\_compare + " Country Comparison")

plt.xlabel("Years")

plt.xticks(np.arange(min(years), max(years)+1, 1.0))

plt.ylabel("% Travellers")

# Print our chart to the screen

plt.show()

**A graph with lines and numbers

Description automatically generated**

**Day 3 , Activity 1 🡪 Samples Solution**

# Dependencies

import pandas as pd

import matplotlib.pyplot as plt

import scipy.stats as st

import numpy as np

# Read in the LAX temperature data

temperature\_df = pd.read\_csv('../Resources/lax\_temperature.csv')

temperatures = temperature\_df['HourlyDryBulbTemperature (C)']

# Demonstrate calculating measures of central tendency

mean\_numpy = np.mean(temperatures)

print(f"The mean temperature at the LAX airport is {mean\_numpy}")

median\_numpy = np.median(temperatures)

print(f"The median temperature at the LAX airport is {median\_numpy}")

mode\_scipy = st.mode(temperatures, keepdims=False)

print(f"The mode temperature at the LAX airport is {mode\_scipy[0]}")

The mean temperature at the LAX airport is 14.252394446018743

The median temperature at the LAX airport is 13.9

The mode temperature at the LAX airport is 13.9

# Characterize the data set using matplotlib and stats.normaltest

plt.hist(temperatures)

plt.xlabel('Temperature (°C)')

plt.ylabel('Counts')

plt.show()

print(st.normaltest(temperatures.sample(50)))

**A graph of a temperature

Description automatically generated**

# Run the normality test on the whole temperatures DataFrame.

print(st.normaltest(temperatures.sample(3529)))

NormaltestResult(statistic=203.70256859680273, pvalue=5.841834570719453e-45)

# Demonstrate calculating the variance and standard deviation using the different modules (“ddof” mean ‘Delta degree of freedom’ used in stastical calcuiations, particularly used in Numpy and Panda

var\_numpy = np.var(temperatures,ddof = 0)

print(f"The population variance using the NumPy module is {var\_numpy}")

sd\_numpy = np.std(temperatures,ddof = 0)

print(f"The population standard deviation using the NumPy module is {sd\_numpy}")

The population variance using the NumPy module is 9.985231585982183

The population standard deviation using the NumPy module is 3.159941706105064

# Calculate the 68-95-99.7 rule using the standard deviation

print(f"Roughly 68% of the data is between {round(mean\_numpy-sd\_numpy,3)} and {round(mean\_numpy+sd\_numpy,3)}")

print(f"Roughly 95% of the data is between {round(mean\_numpy-2\*sd\_numpy,3)} and {round(mean\_numpy+2\*sd\_numpy,3)}")

print(f"Roughly 99.7% of the data is between {round(mean\_numpy-3\*sd\_numpy,3)} and {round(mean\_numpy+3\*sd\_numpy,3)}")

Roughly 68% of the data is between 11.092 and 17.412

Roughly 95% of the data is between 7.933 and 20.572

Roughly 99.7% of the data is between 4.773 and 23.732

# Demonstrate calculating the z-scores using SciPy

z\_scipy = st.zscore(temperatures)

print(f"The z-scores using the SciPy module are {z\_scipy}")

The z-scores using the SciPy module are 0 -0.997612

1 -1.155842

2 -0.997612

3 -1.535596

4 -2.579919

...

3524 0.426465

3525 0.236588

3526 0.046711

3527 0.046711

3528 0.046711

Name: HourlyDryBulbTemperature (C), Length: 3529, dtype: float64

**Day 3 , Activity 2 🡪 Quartile and Outliners (Box Plot)**

# Dependencies

import pandas as pd

A graph with lines and numbers

Description automatically generatedimport numpy as np

import matplotlib.pyplot as plt

# Example outlier plot of reaction times

times = [96,98,100,105,85,88,95,100,101,102,97,98,5]

fig1, ax1 = plt.subplots()

ax1.set\_title('Reaction Times at Tennis Batting Cage')

ax1.set\_ylabel('Reaction Time (ms)')

ax1.boxplot(times)

plt.show()

# We need to sort the data to determine which could be outliers

times.sort()

print(times)

[5, 85, 88, 95, 96, 97, 98, 98, 100, 100, 101, 102, 105]

# The second example again looks at the LAX temperature data set and computes quantiles

A diagram of a diagram

Description automatically generatedtemperature\_df = pd.read\_csv('../Resources/lax\_temperature.csv')

temperatures = temperature\_df['HourlyDryBulbTemperature (C)']

fig1, ax1 = plt.subplots()

ax1.set\_title('Temperatures at LAX')

ax1.set\_ylabel('Temperature (°C)')

ax1.boxplot(temperatures)

plt.show()

# If the data is in a dataframe, we use pandas to give quartile calculations

quartiles = temperatures.quantile([.25,.5,.75])

lowerq = quartiles[0.25]

upperq = quartiles[0.75]

iqr = upperq-lowerq

print(f"The lower quartile of temperatures is: {lowerq}")

print(f"The upper quartile of temperatures is: {upperq}")

print(f"The interquartile range of temperatures is: {iqr}")

print(f"The the median of temperatures is: {quartiles[0.5]} ")

lower\_bound = lowerq - (1.5\*iqr)

upper\_bound = upperq + (1.5\*iqr)

print(f"Values below {lower\_bound} could be outliers.")

print(f"Values above {upper\_bound} could be outliers.")

The lower quartile of temperatures is: 12.2

The upper quartile of temperatures is: 15.6

The interquartile range of temperatures is: 3.4000000000000004

The the median of temperatures is: 13.9

Values below 7.099999999999999 could be outliers.

Values above 20.7 could be outliers.

**Day 3 , Activity 3 🡪 Summary Stats (Student to do)**

# Dependencies

import pandas as pd

import matplotlib.pyplot as plt

import scipy.stats as st

# Read in the california housing data set

california\_data = pd.read\_csv('../Resources/California\_Housing.csv')

california\_data.head()

|  | **MedInc** | **HouseAge** | **AveRooms** | **AveBedrms** | **Population** | **AveOccup** | **Latitude** | **Longitude** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 5.2742 | 17.0 | 8.908654 | 1.884615 | 351.0 | 1.687500 | 32.98 | -117.27 |
| **1** | 1.8438 | 52.0 | 3.069847 | 1.044293 | 2382.0 | 4.057922 | 34.07 | -118.26 |
| **2** | 7.0691 | 13.0 | 6.689697 | 1.009697 | 2444.0 | 2.962424 | 37.57 | -121.87 |
| **3** | 5.4719 | 38.0 | 5.890080 | 0.957105 | 1228.0 | 3.292225 | 37.52 | -122.28 |
| **4** | 4.0156 | 35.0 | 6.195312 | 1.171875 | 669.0 | 5.226562 | 33.93 | -117.41 |

# Get the information on the DataFrame

california\_data.info()

RangeIndex: 3000 entries, 0 to 2999

Data columns (total 8 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 MedInc 3000 non-null float64

1 HouseAge 3000 non-null float64

2 AveRooms 3000 non-null float64

3 AveBedrms 3000 non-null float64

4 Population 3000 non-null float64

5 AveOccup 3000 non-null float64

6 Latitude 3000 non-null float64

7 Longitude 3000 non-null float64

dtypes: float64(8)

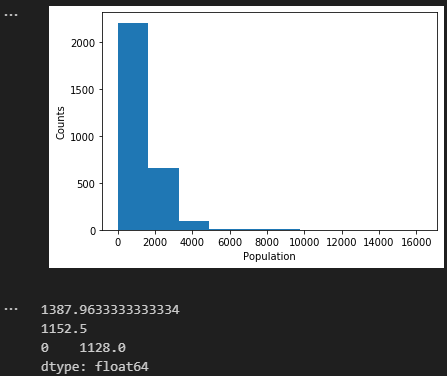
memory usage: 187.6 KB

# Determine which measure of central tendency is most appropriate to describe the Population

plt.hist(california\_data['Population'])

plt.xlabel('Population')

plt.ylabel('Counts')

plt.show()

print(california\_data['Population'].mean())

print(california\_data['Population'].median())

print(california\_data['Population'].mode())

# Determine if the house age in California is normally distributed using a small and large sample size.

plt.hist(california\_data['HouseAge'])

plt.xlabel('House Age (years)')

plt.ylabel('Counts')

plt.show()

print(st.normaltest(california\_data["HouseAge"].sample(100)))

A screenshot of a computer screen

Description automatically generatedprint(st.normaltest(california\_data["HouseAge"].sample(2000)))

# Determine if there are any potential outliers in the average occupancy in California

quartiles = california\_data['AveOccup'].quantile([.25,.5,.75])

lowerq = quartiles[0.25]

upperq = quartiles[0.75]

iqr = upperq-lowerq

print(f"The lower quartile of occupancy is: {lowerq}")

print(f"The upper quartile of occupancy is: {upperq}")

print(f"The interquartile range of occupancy is: {iqr}")

print(f"The the median of occupancy is: {quartiles[0.5]} ")

lower\_bound = lowerq - (1.5\*iqr)

upper\_bound = upperq + (1.5\*iqr)

print(f"Values below {lower\_bound} could be outliers.")

print(f"Values above {upper\_bound} could be outliers.")

outlier\_occupancy = california\_data.loc[(california\_data['AveOccup'] < lower\_bound) | (california\_data['AveOccup'] > upper\_bound)]

outlier\_occupancy

A screenshot of a computer screen

Description automatically generated

# With the potential outliers, what is the lowest and highest median income (in $1000s) observed?

print(f"The minimum median income of the potential outliers is {outlier\_occupancy['MedInc'].min()}")

print(f"The maximum median income of the potential outliers is {outlier\_occupancy['MedInc'].max()}")



# Bonus - plot the latitude and longitude of the California housing data using Matplotlib, color the data points using the median income of the block.

plt.scatter(california\_data['Longitude'],california\_data['Latitude'],c=california\_data['MedInc'])

clb = plt.colorbar()

plt.xlabel("Longitude")

plt.ylabel("Latitude")

clb.set\_label("Median Income")

plt.show()

A diagram of a number of dots

Description automatically generated

**Day 3, Activity 4 – Standard\_error\_solution**

# Dependencies

import pandas as pd

import random

import matplotlib.pyplot as plt

import numpy as np

from scipy.stats import sem

# Set the seed so our data is reproducible

random.seed(42)

# Sample versus population example fuel economy

fuel\_economy = pd.read\_csv('../Resources/2019\_fuel\_economy.csv')

# First overview the data set - how many factors, etc.

print(fuel\_economy.head())

Type Combined\_LP100KM Make\_Model

0 Two Seaters 11.2007 Acura\_NSX

1 Two Seaters 8.4184 ALFA ROMEO\_4C

2 Two Seaters 11.2299 Aston Martin Lagonda Ltd\_Vantage V8

3 Two Seaters 9.0484 Audi\_TT Roadster quattro

4 Two Seaters 8.5130 BMW\_Z4 sDrive30i

# Calculate the summary statistics and plot the histogram of the entire population data

print(f"The mean L/100km of all vehicles is: {round(fuel\_economy['Combined\_LP100KM'].mean(),2)}")

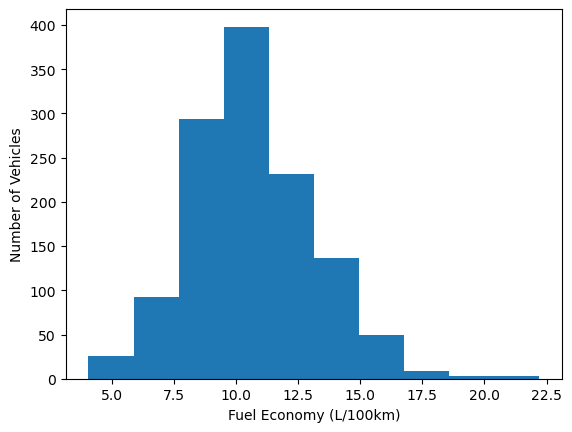
print(f"The standard deviation of all vehicle's L/100km is: {round(fuel\_economy['Combined\_LP100KM'].std(),2)}")

plt.hist(fuel\_economy['Combined\_LP100KM'])

plt.xlabel("Fuel Economy (L/100km)")

plt.ylabel("Number of Vehicles")

plt.show()

The mean L/100km of all vehicles is: 10.67

The standard deviation of all vehicle's L/100km is: 2.47

# Calculate the summary statistics and plot the histogram of the sample data using iloc

subset = fuel\_economy.iloc[range(766,856)]

print(f"The mean L/100km of all vehicles is: {round(subset['Combined\_LP100KM'].mean(),2)}")

print(f"The standard deviation of all vehicle's L/100km is: {round(subset['Combined\_LP100KM'].std(),2)}")

plt.hist(subset['Combined\_LP100KM'])

plt.xlabel("Fuel Economy (L/100km)")

plt.ylabel("Number of Vehicles")

plt.show()

The mean L/100km of all vehicles is: 12.95

A graph of fuel economy

Description automatically generatedThe standard deviation of all vehicle's L/100km is: 1.74

# Calculate the summary statistics and plot the histogram of the sample data using random sampling

subset = fuel\_economy.sample(90)

print(f"The mean L/100km of all vehicles is: {round(subset['Combined\_LP100KM'].mean(),2)}")

print(f"The standard deviation of all vehicle's L/100km is: {round(subset['Combined\_LP100KM'].std(),2)}")

plt.hist(subset['Combined\_LP100KM'])

plt.xlabel("Fuel Economy (L/100km)")

A graph of fuel economy

Description automatically generatedplt.ylabel("Number of Vehicles")

plt.show()

The mean L/100km of all vehicles is: 11.01

The standard deviation of all vehicle's L/100km is: 2.83

# Generate a new 30 vehicle sample and calculate the SEM of the sample

sample = fuel\_economy.sample(30)

print(f"The SEM value for the sample fuel economy data is {sem(sample['Combined\_LP100KM'])}")

The SEM value for the sample fuel economy data is 0.49228586996640716

# Create a sample set of 10, each with 30 vehicles

vehicle\_sample\_set = [fuel\_economy.sample(30) for x in range(0,10)]

# Generate the plot data for each sample (‘sem’ = Std Error of Mean)

means = [sample['Combined\_LP100KM'].mean() for sample in vehicle\_sample\_set]

standard\_errors = [sem(sample['Combined\_LP100KM']) for sample in vehicle\_sample\_set]

x\_axis = np.arange(0, len(vehicle\_sample\_set), 1) + 1

# Setting up the plot

fig, ax = plt.subplots()

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

ax.set\_xlim(0, len(vehicle\_sample\_set) + 1)

ax.set\_ylim(9,13)

ax.set\_xlabel("Sample Number")

ax.set\_ylabel("Mean L/100km")

plt.show()

A graph with blue dots and numbers

Description automatically generated

**Day 3, Activity 5 – Student to do – California Housing**

# Import dependencies

from matplotlib import pyplot as plt

import numpy as np

import pandas as pd

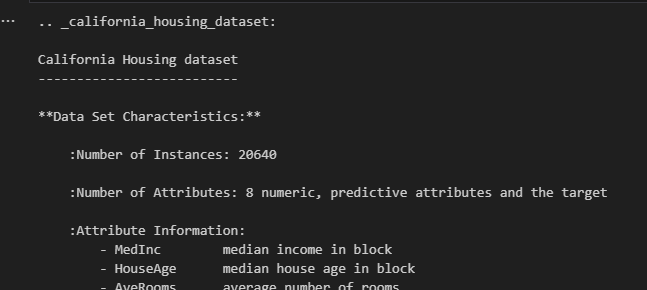
from sklearn.datasets import fetch\_california\_housing

from scipy.stats import sem

# Import the California housing data set and get description

california\_dataset = fetch\_california\_housing()

print(california\_dataset.DESCR)

****

# Read California housing data into a Pandas dataframe

housing\_data = pd.DataFrame(data=california\_dataset.data,columns=california\_dataset.feature\_names)

housing\_data['MEDV'] = california\_dataset.target

housing\_data.head()

**A screenshot of a black screen

Description automatically generated**

# Create a bunch of samples, each with sample size of 20

nsamples = 25

div = 20

samples = [housing\_data.sample(div) for x in range(0,nsamples)]

# Calculate means

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

# Calculate standard error on means

sems = [sem(s['MEDV']) for s in samples]

# Plot sample means with error bars

fig, ax = plt.subplots()

ax.errorbar(np.arange(0, len(samples), 1)+1,means, yerr=sems, fmt="o", color="b",

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

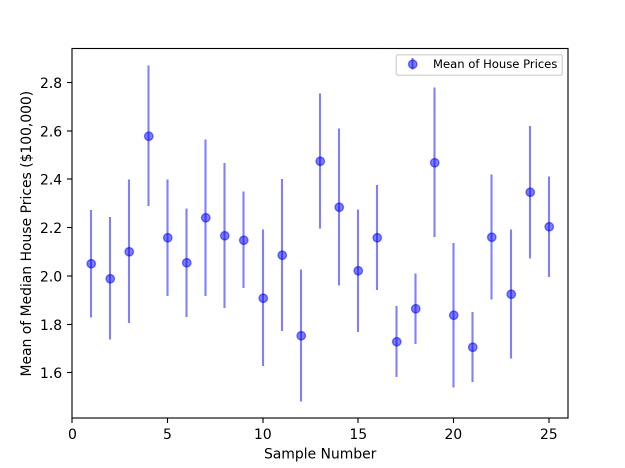
ax.set\_xlim(0, len(means)+1)

ax.set\_xlabel("Sample Number")

ax.set\_ylabel("Mean of Median House Prices ($100,000)")

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

plt.show()

****

**Day 3, Activity 6 – Correlation Conundrum Solution**

# Dependencies

import pandas as pd

import matplotlib.pyplot as plt

import scipy.stats as st

# Import the WDI dataset, drop missing data

wdi\_data = pd.read\_csv('../Resources/WDI\_2018.csv')

wdi\_data = wdi\_data.dropna()

wdi\_data.head()

**A screenshot of a computer screen

Description automatically generated**

# For the first example, determine which pairs of factors are correlated.

plt.scatter(wdi\_data.iloc[:,1],wdi\_data.iloc[:,8])

plt.xlabel('Income Per Capita')

plt.ylabel('Average Alcohol Consumed Per Person Per Year (L)')

**A graph of blue dots

Description automatically generated**plt.show()

plt.scatter(wdi\_data.iloc[:,3],wdi\_data.iloc[:,10])

plt.xlabel('Population Median Age')

plt.ylabel('Cell Phones Per 100 People')

plt.show()

plt.scatter(wdi\_data.iloc[:,9],wdi\_data.iloc[:,7])

plt.xlabel('% Population with Access to Clean Water')

plt.ylabel('Male Life Expectancy')

plt.show()

plt.scatter(wdi\_data.iloc[:,1],wdi\_data.iloc[:,12])

plt.xlabel('Income Per Capita')

plt.ylabel('% Measles Immunisation')

plt.show()

# Compare the calcualted Pearson's r to the plots

plt.scatter(income,alcohol)

plt.xlabel('Income Per Capita')

plt.ylabel('Average Alcohol Consumed Per Person Per Year (L)')

print(f"The correlation between both factors is {round(correlation[0],2)}")

plt.show()

age = wdi\_data.iloc[:,3]

cell\_phones = wdi\_data.iloc[:,10]

correlation = st.pearsonr(age,cell\_phones)

plt.scatter(age,cell\_phones)

plt.xlabel('Population Median Age')

plt.ylabel('Cell Phones Per 100 People')

print(f"The correlation between both factors is {round(correlation[0],2)}")

plt.show()

…The correlation between both factors is 0.53

water = wdi\_data.iloc[:,9]

life = wdi\_data.iloc[:,7]

correlation = st.pearsonr(water,life)

plt.scatter(water,life)

plt.xlabel('% Population with Access to Clean Water')

plt.ylabel('Male Life Expectancy')

print(f"The correlation between both factors is {round(correlation[0],2)}")

plt.show()

…The correlation between both factors is 0.83

income = wdi\_data.iloc[:,1]

measles = wdi\_data.iloc[:,12]

correlation = st.pearsonr(income,measles)

plt.scatter(income,measles)

plt.xlabel('Income Per Capita')

plt.ylabel('% Measles Immunisation')

print(f"The correlation between both factors is {round(correlation[0],2)}")

plt.show()

…The correlation between both factors is 0.34

**Day 3, activity 7 - Wine**

# Dependencies

import pandas as pd

import sklearn.datasets as dta

import scipy.stats as st

import matplotlib.pyplot as plt

# Read in the wine recognition data set from sklearn and load into Pandas

data = dta.load\_wine()

wine\_data = pd.DataFrame(data.data,columns=data.feature\_names)

print(data.DESCR)

# Plot flavanoids versus malic\_acid on a scatterplot

flavanoids = wine\_data['flavanoids']

malic\_acid = wine\_data['malic\_acid']

plt.scatter(malic\_acid,flavanoids)

plt.xlabel("Amount of Malic Acid")

plt.ylabel("Amount of Flavanoids")

plt.show()

…Graph

# Calculate the correlation coefficient between malic\_acid and flavanoids

print(f"The correlation coefficient between malic acid and flavanoids is {round(st.pearsonr(malic\_acid,flavanoids)[0],2)}")

…The correlation coefficient between malic acid and flavanoids is -0.41

# Plot colour\_intensity versus alcohol on a scatterplot

colour\_intensity = wine\_data['color\_intensity']

alcohol = wine\_data['alcohol']

plt.scatter(alcohol,colour\_intensity)

plt.xlabel("Amount of Alcohol")

plt.ylabel("Intensity of Colour")

plt.show()

# Calculate the correlation coefficient between alcohol and colour\_intensity

print(f"The correlation coefficient between alcohol and color intensity is {round(st.pearsonr(alcohol,colour\_intensity)[0],2)}")

…The correlation coefficient between alcohol and color intensity is 0.55

# BONUS: Generate the correlation matrix and find the strongest positive and negative correlations

wine\_corr = wine\_data.corr()

wine\_corr.unstack().sort\_values()

**Day 3, activity 8 - California Housing (Regression)**

# Import dependencies

from matplotlib import pyplot as plt

from scipy.stats import linregress

import numpy as np

from sklearn import datasets

import pandas as pd

**Compare different factors in the California housing dataset**

# Read in the California housing dataset

california\_dataset = datasets.fetch\_california\_housing()

housing\_data = pd.DataFrame(data=california\_dataset.data,columns=california\_dataset.feature\_names)

housing\_data['MEDV'] = california\_dataset.target

# Reduce the dataset to remove AveRooms outliers

housing\_data\_reduced = pd.DataFrame(housing\_data.loc[housing\_data['AveRooms']<10,:])

# Reduce the dataset to the San Diego Area (based on approx latitude & longitude area)

san\_diego\_housing = pd.DataFrame(housing\_data\_reduced.loc[((housing\_data\_reduced['Latitude']>32.664282) &

                                                          (housing\_data\_reduced['Latitude']<32.980514) &

                                                          (housing\_data\_reduced['Longitude']>-117.300418) &

                                                          (housing\_data\_reduced['Longitude']<-117.01950)),:])

**A diagram of blue dots

Description automatically generated**# Plot out rooms versus median house price

x\_values = san\_diego\_housing['AveRooms']

y\_values = san\_diego\_housing['MEDV']

plt.scatter(x\_values,y\_values)

plt.xlabel('Rooms in House')

plt.ylabel('Median House Prices ($100,000)')

plt.show()

# Add the linear regression equation and line to plot

x\_values = san\_diego\_housing['AveRooms']

y\_values = san\_diego\_housing['MEDV']

(slope, intercept, rvalue, pvalue, stderr) = linregress(x\_values, y\_values)

regress\_values = x\_values \* slope + intercept

line\_eq = "y = " + str(round(slope,2)) + "x + " + str(round(intercept,2))

plt.scatter(x\_values,y\_values)

plt.plot(x\_values,regress\_values,"r-")

plt.annotate(line\_eq,(5.8,0.8),fontsize=15,color="red")

**A diagram of a red line and blue dots

Description automatically generated**plt.xlabel('Rooms in House')

plt.ylabel('Median House Prices ($100,000)')

plt.show()

# Print out the r-squared value along with the plot.

x\_values = san\_diego\_housing['AveRooms']

y\_values = san\_diego\_housing['MEDV']

(slope, intercept, rvalue, pvalue, stderr) = linregress(x\_values, y\_values)

regress\_values = x\_values \* slope + intercept

line\_eq = "y = " + str(round(slope,2)) + "x + " + str(round(intercept,2))

plt.scatter(x\_values,y\_values)

plt.plot(x\_values,regress\_values,"r-")

plt.annotate(line\_eq,(5.8,0.8),fontsize=15,color="red")

plt.xlabel('Rooms in House')

**A graph of blue dots and red line

Description automatically generated**plt.ylabel('Median House Prices ($100,000)')

print(f"The r-squared is: {rvalue\*\*2}")

plt.show()

**Compare linear relationships with the diabetes dataset**

# Read in the diabetes dataset

diabetes\_data = datasets.load\_diabetes()

data = pd.DataFrame(diabetes\_data.data,columns=diabetes\_data.feature\_names)

data['1Y\_Disease\_Progress'] = diabetes\_data.target

# Plot the different factors in a scatter plot

x\_values = data['bp']

y\_values = data['1Y\_Disease\_Progress']

(slope, intercept, rvalue, pvalue, stderr) = linregress(x\_values, y\_values)

regress\_values = x\_values \* slope + intercept

line\_eq = "y = " + str(round(slope,2)) + "x + " + str(round(intercept,2))

plt.scatter(x\_values,y\_values)

plt.plot(x\_values,regress\_values,"r-")

plt.annotate(line\_eq,(0,50),fontsize=15,color="red")

**A graph of a number of blue dots with a red line

Description automatically generated**plt.xlabel('Normalised Blood Pressure')

plt.ylabel('1Y\_Disease\_Progress')

print(f"The r-squared is: {rvalue\*\*2}")

plt.show()

**Day 3, Activity 9 – Singapore Motor Vehicle Population**

# Dependencies

from matplotlib import pyplot as plt

from scipy import stats

import numpy as np

import pandas as pd

# Load vehicle data set into pandas

vehicle\_data = pd.read\_csv("../Resources/singapore-motor-vehicle-population.csv")

vehicle\_data.head()

**A screenshot of a black screen

Description automatically generated**

# Generate a scatter plot of year versus number of petrol-electric cars

year = vehicle\_data.loc[(vehicle\_data["type"]=="Cars") & (vehicle\_data["engine"]=="Petrol-Electric"),"year"]

petrol\_electric\_cars = vehicle\_data.loc[(vehicle\_data["type"]=="Cars") & (vehicle\_data["engine"]=="Petrol-Electric"),"number"]

**A graph with blue dots

Description automatically generated**plt.scatter(year,petrol\_electric\_cars)

plt.xticks(year, rotation=90)

plt.xlabel('Year')

plt.ylabel('Petrol Electric Cars')

plt.show()

# Perform a linear regression on year versus petrol-electric cars

pe\_slope, pe\_int, pe\_r, pe\_p, pe\_std\_err = stats.linregress(year, petrol\_electric\_cars)

# Create equation of line to calculate predicted number of petrol-electric cars

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

# Plot the linear model on top of scatter plot

year = vehicle\_data.loc[(vehicle\_data["type"]=="Cars") & (vehicle\_data["engine"]=="Petrol-Electric"),"year"]

petrol\_electric\_cars = vehicle\_data.loc[(vehicle\_data["type"]=="Cars") & (vehicle\_data["engine"]=="Petrol-Electric"),"number"]

plt.scatter(year,petrol\_electric\_cars)

**A graph with blue dots and numbers

Description automatically generated**plt.plot(year,pe\_fit,"--")

plt.xticks(year, rotation=90)

plt.xlabel('Year')

plt.ylabel('Petrol Electric Cars')

plt.show()

# Repeat plotting scatter and linear model for year versus petrol cars

petrol\_cars = vehicle\_data.loc[(vehicle\_data["type"]=="Cars") & (vehicle\_data["engine"]=="Petrol"), "number"]

p\_slope, p\_int, p\_r, p\_p, p\_std\_err = stats.linregress(year, petrol\_cars)

**A graph with blue dots and lines

Description automatically generated**p\_fit = p\_slope \* year + p\_int

plt.scatter(year,petrol\_cars)

plt.plot(year,p\_fit,"--")

plt.xticks(year, rotation=90)

plt.xlabel('Year')

plt.ylabel('Petrol Cars')

plt.show()

# Repeat plotting scatter and linear model for year versus electric cars

diesel\_cars = vehicle\_data.loc[(vehicle\_data["type"]=="Cars") & (vehicle\_data["engine"]=="Diesel"), "number"]

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

    year, diesel\_cars)

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

plt.scatter(year,diesel\_cars)

plt.plot(year,d\_fit,"--")

plt.xticks(year, rotation=90)

plt.xlabel('Year')

plt.ylabel('Diesel Cars')

plt.show()

# Generate a facet plot of all 3 figures

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

fig.suptitle("Number of Vehicles Over Time", fontsize=16, fontweight="bold")

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

ax1.plot(year, petrol\_electric\_cars, linewidth=1, marker="o")

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

ax1.set\_ylabel("Petrol-Electric Cars")

ax2.plot(year, petrol\_cars, linewidth=1, marker="o", color="y")

ax2.plot(year, p\_fit, "y--", linewidth=1)

ax2.set\_ylabel("Petrol Cars")

ax3.plot(year, diesel\_cars, linewidth=1, marker="o", color="g")

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

ax3.set\_ylabel("Diesel Cars")

**A graph of vehicles over time

Description automatically generated**ax3.set\_xlabel("Year")

plt.show()

# Calculate the number of cars for 2024

year = 2024

print(f"The number of petrol-electic cars in 2024 will be {round(pe\_slope \* year + pe\_int,0)}.")

print(f"The number of petrol cars in 2024 will be {round(p\_slope \* year + p\_int,0)}.")

print(f"The number of diesel cars in 2024 will be {round(d\_slope \* year + d\_int,0)}.")

The number of petrol-electic cars in 2024 will be 27516.0.

The number of petrol cars in 2024 will be 645370.0.

The number of diesel cars in 2024 will be 20541.0.