MA 354: Data Analysis I – Fall 2021 Homework 0:

Complete the following opportunities to use what we've talked about in class. These questions will be graded for correctness, communication and succinctness. Ensure you show your work and explain your logic in a legible and refined submission.

0. **Complete weekly diagnostics.**
1. Simple tasks.

(a) Print the first 25 even integers. **Solution:**

```
import numpy as np

a = np.arange(1, 26)  # Create a 1D array with values ranging from 1 to 25
print(a)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25]
```

(b) Print the square root of the first 50 odd natural numbers. **Solution:**

```
import numpy as np

b = np.arange(1, 100, 2)  # Create an array of all the odd integers from 1 to 100
print(np.sqrt(b))  # Find the square root of all the elements in the array
```

```
[1.         1.73205081 2.23606798 2.64575131 3.         3.31662479
 3.60555128 3.87298335 4.12310563 4.35889894 4.58257569 4.79583152
 5.         5.19615242 5.38516481 5.56776436 5.74456265 5.91607978
 6.08276253 6.244998   6.40312424 6.55743852 6.70820393 6.8556546
 7.         7.14142843 7.28010989 7.41619849 7.54983444 7.68114575
 7.81024968 7.93725393 8.06225775 8.18535277 8.30662386 8.42614977
 8.54400375 8.66025404 8.77496439 8.88819442 9.         9.11043358
 9.21954446 9.32737905 9.43398113 9.53939201 9.64365076 9.74679434
 9.8488578  9.94987437]
```

(c) Evaluate how many numbers between 1 and 100 are divisible by 3. **Solution**:

```
import numpy as np

c = np.arange(1, 100)  # Create an array of integers from 1 to 99
divisible_by_3 = c[c % 3 == 0]  # Select the elements of c that are divisible by 3

print(len(divisible_by_3))
```

```
33
```

(d) Evaluate y = exp(1) – (1 + 1/x)^x over a sequence from 10 to 200 (by 10). Where does it appear that y is approaching as x increases? For a visual representation check plot. **Solution**:

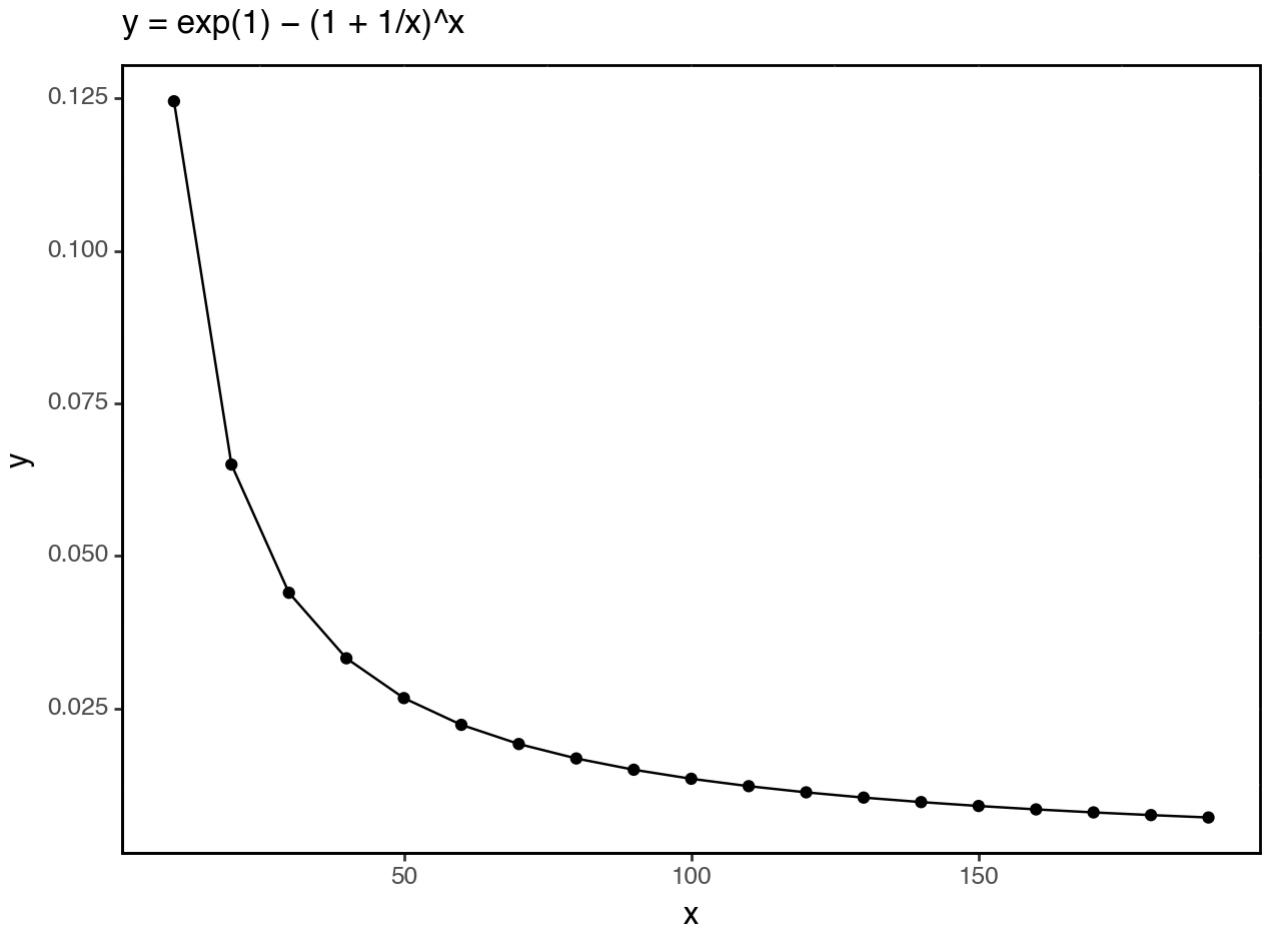
```
# Evaluate y = exp(1) – (1 + 1/x)^x over a sequence from 10 to 200 (by 10). Where does
import numpy as np
```

```
import plotnine as p9
import pandas as pd

sequence = np.arange(10, 200, 10)  # Create an array of integers from 10 to 200 (by 10
expression = np.exp(1) - (1 + 1 / sequence) ** sequence  # Evaluate
print(expression)
df = pd.DataFrame({"x": sequence, "y": expression})

# Plot
(
    p9.ggplot(df, p9.aes(x="x", y="y"))
    + p9.geom_point()
    + p9.geom_line()
    + p9.labs(title="y = exp(1) - (1 + 1/x)^x", x="x", y="y")
    + p9.theme_bw()
    + p9.theme(
        panel_grid_major=p9.element_line(color="white", size=0.1),
        panel_grid_minor=p9.element_line(color="white", size=0.1),
        panel_border=p9.element_rect(color="black", fill=None, size=1),
        axis_line=p9.element_line(color="black", size=1),
        legend_title=p9.element_text(size=10, face="bold"),
        legend_text=p9.element_text(size=8),
        legend_key=p9.element_rect(fill="white", size=0.5, linetype="solid"),
        plot_title=p9.element_text(size=12, face="bold"),
    )
)
```

```
[0.12453937 0.06498412 0.04396305 0.03321799 0.0266938  0.02231169
 0.01916546 0.01679689 0.01494937 0.013468   0.01225375 0.01124034
 0.01038175 0.00964501 0.00900592 0.00844625 0.00795208 0.00751253
 0.00711903]
```



<Figure Size: (640 x 480)>

(e) Write a function that takes a vector of coordinates in R2 and converts them to polar. For (x, y) we want

r = sqrt(x^2 +y^2) θ = arctan(y/x)

Make sure to return θ ∈ (0, 2π). Note that if (x, y) is in the second, or third quadrant we must add π to R's result for θ, and if (x, y) is in the fourth quadrant we must add 2π.

Solution:

```
import numpy as np


def polar(x, y):
    r = np.sqrt(x**2 + y**2)  # r = sqrt(x^2 + y^2)
    theta = (
        np.arctan(y / x) if x != 0 else np.pi / 2
    )  # theta = arctan(y/x) if x != 0 else π/2
    if x < 0 and y > 0:  # Make sure to return θ ∈ (0, 2π)
        theta += (
            np.pi
        )  # Note that if (x, y) is in the second, or third quadrant we must add π to
    elif x < 0 and y < 0:  # Make sure to return θ ∈ (0, 2π)
        theta += (
```

```
            2 * np.pi
        )  # Note that if (x, y) is in the fourth quadrant we must add 2π
    return r, theta  # Return r and theta


print(polar(0, -8))
```

```
(8.0, 1.5707963267948966)
```

(f) Add an optional argument to your function for part (e) that allows the user to specify that they want degrees instead of radians. Note that radians should be the default.

Solution:

```
def polar_updated(x, y, degrees=False):
    polar_result = polar(x, y)
    if degrees:
        return polar_result[0], polar_result[1] * 180 / np.pi
    return polar_result
```

Demonstrate your function for points (−3,10),(5,−8),(13,13),(−2,−10), and (0,−8) in both radians and degrees.

Solution:

```
print(polar_updated(-3, 10))
print(polar_updated(5, -8))
print(polar_updated(13, 13))
print(polar_updated(-2, -10))
print(polar_updated(0, -8))

print(polar_updated(-3, 10, degrees=True))
print(polar_updated(5, -8, degrees=True))
print(polar_updated(13, 13, degrees=True))
print(polar_updated(-2, -10, degrees=True))
print(polar_updated(0, -8, degrees=True))
```

```
(10.44030650891055, 1.8622531212727635)
(9.433981132056603, -1.0121970114513341)
(18.384776310850235, 0.7853981633974483)
(10.198039027185569, 7.656586074124602)
(8.0, 1.5707963267948966)
(10.44030650891055, 106.69924423399362)
(9.433981132056603, -57.9946167919165)
(18.384776310850235, 45.0)
(10.198039027185569, 438.69006752597977)
(8.0, 90.0)
```

2. Consider a tug-of-war competition for robots. In each matchup, two robots take turns tugging the rope until the marker indicates that one of the robots won.

- The match starts with the marker at 0.
- Robot A pulls the rope – use runif(n=1,min=0,max=0.50) to simulate the magnitude of the pull.

- Adding the simulated value to the marker position gives the new position of the marker.
- Robot B pulls the rope in the opposite direction – use runif(n=1,min=-0.50,max=0) to simulate the magnitude of the pull. Adding the simulated value to the marker position gives the new position of the marker. -The two robots continue taking turns until the marker moves past -0.50 or 0.50.

(a) Write a function that simulates a tug-of-war match. When called, it should simulate a tug-of-war contest, and return "Robot A" when robot A wins, and "Robot B" when robot B wins.

```
def tug_of_war():
    marker = 0
    while marker < 0.5 and marker > -0.5:
        marker += np.random.uniform(0, 0.5)
        if marker > 0.5:
            return "Robot A"
        marker += np.random.uniform(-0.5, 0)
        if marker < -0.5:
            return "Robot B"
    return "Robot A" if marker > 0 else "Robot B"
```

(b) Report the results of 10,000 simulated robot tug of war battles. Is the game fair?

```
results = []
for i in range(10000):
    results.append(tug_of_war())
print(results.count("Robot A"))
print(results.count("Robot B"))
```

```
6179
3821
```

(c) What can be done to make this game more fair? Be creative!

```
def tug_of_war_updated():
    marker = 0
    first_robot = np.random.choice(["A", "B"])  # Randomly select which robot goes fi
    while marker < 0.5 and marker > -0.5:
        if first_robot == "A":  # If robot A goes first
            marker += np.random.uniform(0, 0.5)
            if marker > 0.5:
                return "Robot A"
            marker += np.random.uniform(-0.5, 0)
            if marker < -0.5:
                return "Robot B"
        else:  # If robot B goes first
            marker += np.random.uniform(-0.5, 0)
            if marker < -0.5:
                return "Robot B"
            marker += np.random.uniform(0, 0.5)
            if marker > 0.5:
                return "Robot A"
    return "Robot A" if marker > 0 else "Robot B"
```

```
results = []
for i in range(10000):
    results.append(tug_of_war_updated())
print(results.count("Robot A"))
print(results.count("Robot B"))
```

```
5101
4899
```

3. In class, we wrote the following function for a project Euler problem exploring divisibility streaks.

```
def streak(n):
    k = 0  # starting k
    while True:
        remainder = (n + k) % (k + 1)  # is n+k divisible by k+1
        if remainder == 0:  # is the remainder 0?
            k += 1
        else:  # the remainder isn't 0 --> streak ended
            break
    return k
```

(a) Why does an issue occur when we plug in n = 1? Note that there is no error message – the worst!

If we plug in n = 1:

For k = 0: Remainder = (1 + 0) % (0 + 1) = 1 % 1 = 0

Since the remainder is 0, k will be incremented.

For k = 1: Remainder = (1 + 1) % (1 + 1) = 2 % 2 = 0

Again, the remainder is 0, and k will be incremented.

For k = 2: Remainder = (1 + 2) % (2 + 1) = 3 % 3 = 0

This pattern continues, and the loop will never break, causing the function to hang indefinitely. This is because the while True loop has no termination condition for the specific case when n = 1.

(b) Fix the issue in part (a).

```
def streak(n):
    k = 0  # starting k
    while True:
        remainder = (n + k) % (k + 1)  # is n+k divisible by k+1
        if remainder == 0:  # is the remainder 0?
            if k == 0 and n == 1:
                break
            k += 1
        else:  # the remainder isn't 0 --> streak ended
            break
    return k
```

```
print(streak(1))
```

```
0
```

(c) Define a function P(s,N) to be the number of integers n, 1 < n < N, for which streak(n) = s. Complete this using a loop.

```
import numpy as np


def P(s, N):
    count = 0
    for i in np.arange(2, N):
        if streak(i) == s:
            count += 1
    return count
```

(d) Redo part (c) using an apply-function-based approach.

```
def P(s, N):
    return sum([1 for i in range(2, N) if streak(i) == s])
```

(e) Write code to find sum from 1 to 31 of P(I, 4^i). Note that the apply function will not work because would have to create a vector of size 10^31, which would make us run out of memory (unless you have a wild computer).

The approach using the loop approach can work, but it could take days or weeks to finish (depending on your processor). Report the code to find the solution, but make sure not to evaulate it.

Instead, you can test this function by evaluating the sum from 1 to 5 of P(i, 4^i) noting that this function could hypothetically be used to evaluate the original sum with enough time or computational power.

```
import numpy as np


def compute_sum(end):
    total = 0
    for i in range(1, end + 1):
        total += P(i, 4**i)
    return total


# Test the function for the range 1 to 5:
print(compute_sum(5))
```

```
28
```

The standard approach to solving part (e) is rather inefficient. Let's write a more a efficient function. Note this would still take days to complete the calculation from part (e). Instead, you can test this

function by evaluating the sum form 1 to 5 of (i, 4^i). Write a function that does the following:

i. It takes I ∈ N as input ii. creates an I × I matrix – think of this as a table where rows are values of i (1:I), columns are values streaks (1:I), cell entries are counts of observations iii. for each value of interest i = 1, 2, ..., I A. calculates the streaks for the values on [4^(i-1) + 1, 4^i] B. updates the i^th row of the matrix iv. At the end, use the matrix to evaluate the sum.

```
import numpy as np


def compute_matrix(I):
    matrix = np.zeros((I, I))  # Create a matrix of zeros of size I x I

    # For each value of i from 1 to I
    for i in range(1, I + 1):
        # Calculate streaks for values in range [4^(i-1) + 1, 4^i]
        for value in range(
            4 ** (i - 1) + 1, 4**i + 1
        ):  # +1 at the end to make the range inclusive
            s = streak(value)
            if (
                s <= I
            ):  # To ensure the streak value doesn't exceed the matrix dimensions
                matrix[i - 1][
                    s - 1
                ] += 1  # i-1 and s-1 because list indices start from 0

    # Compute the sum from the matrix
    total = 0
    for i in range(I):
        for j in range(I):
            total += (i + 1) * matrix[i][
                j
            ]  # i+1 because list indices start from 0 and we want values from 1 to I

    return matrix, total


I = 5
matrix, sum_val = compute_matrix(I)
print("Matrix:")
for row in matrix:
    print(row)
print("Sum:", sum_val)
```

```
Matrix:
[2. 1. 0. 0. 0.]
[6. 4. 1. 1. 0.]
[24. 16.  4.  3.  0.]
[96. 64. 16. 13.  0.]
[384. 256.  64.  51.   0.]
Sum: 4699.0
```