**Q2)**

**A researcher conducted a random dot motion discrimination task with 2 different motion**

**coherence levels as different conditions. 30 participants performed 100 trials in each**

**condition and the evidence accumulation was recorded. Column 1 of cell array = Condition 1**

**and each cell of Column 1 has one participant’s data. Each cell has a 100 x 1000 matrix. Each**

**row of the matrix = one trial for 1000 ms. The evidence accumulation starts from 300 and**

**reaches the decision threshold at 600.The same convention applies to data from Column 2.**

**The data is attached herewith: Assignment2-NDM.mat’.**

**[ links about importing MATLAB data arrays into Python and R**

**https://in.mathworks.com/help/matlab/matlab\_external/matlab-arrays-as-python-variables.html**

**https://stackoverflow.com/questions/11671883/importing-an-array-from-matlab-into-r ]**

**Now solve the following. Insert a figure (wherever required) and paste the MATLAB/Python/R**

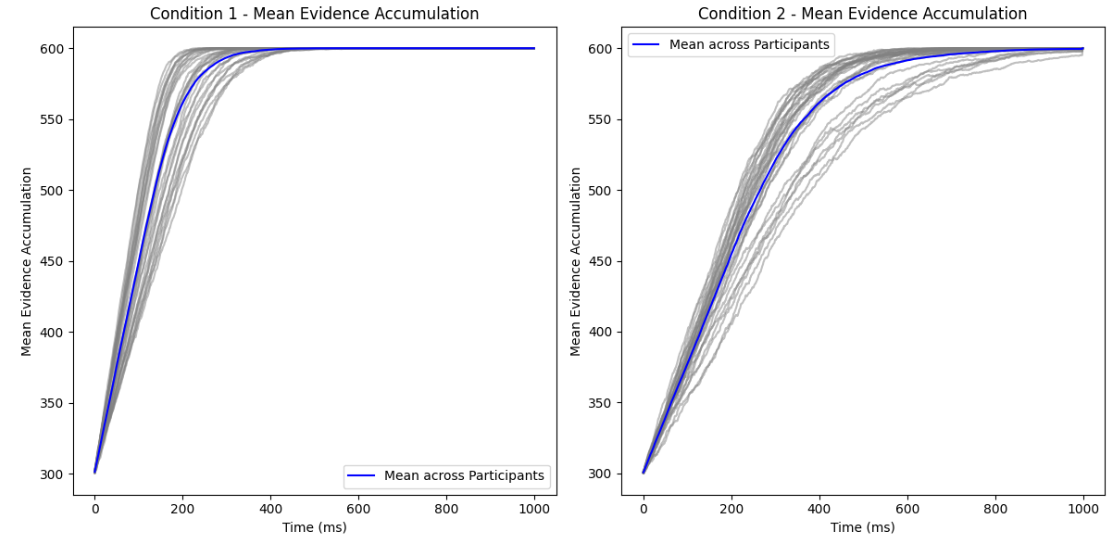
**code for the same. Any figure must provide all information necessary to interpret it including**

**axes labels, captions/legends (simple figure titles as captions are not enough).**

**A i. Create two subplots for the two conditions separately. Plot the mean evidence**

**accumulation (across 100 trials) of each participant in grey colour and the mean**

**evidence accumulation (across 30 participants) in blue colour in each subplot.**

****

**Code is:**

#!/usr/bin/env python

# coding: utf-8

# Create two subplots for the two conditions separately. Plot the mean evidence

# accumulation (across 100 trials) of each participant in grey colour and the mean

# evidence accumulation (across 30 participants) in blue colour in each subplot.

# In[20]:

import os

import scipy.io as sio

import numpy as np

import matplotlib.pyplot as pl

# In[21]:

# Load data from the .mat file

dataset = sio.loadmat("NDM\_Assignment2.mat")

# In[22]:

# Extract the data

dataset = dataset['NDM\_Assignment2']

fig, axs = plt.subplots(1, 2, figsize=(12, 6))

labels = ["Condition 1", "Condition 2"]

idx = 0

while idx < 2:

    label = labels[idx]

    data = dataset[:, idx]

    mean\_participants = np.empty((len(data), 1000))

    mean\_trials = np.empty((len(data), 1000))

    # Calculate the mean evidence accumulation

    i = 0

    while i < len(data):

        participant\_data = data[i]

        mean\_trials[i] = np.mean(participant\_data,axis =0 )

        i += 1

    # Calculate the mean evidence accumulation

    mean\_participants = np.mean(mean\_trials,axis =0)

    # Plot the mean evidence accumulation

    i = 0

    while i < len(mean\_trials):

        axs[idx].plot(mean\_trials[i], color='grey', alpha=0.5)

        i += 1

    # Plot the mean evidence accumulation

    axs[idx].plot(mean\_participants, color='blue', label="Mean across Participants")

    axs[idx].set\_title(f"{label} - Mean Evidence Accumulation")

    axs[idx].set\_xlabel("Time (ms)")

    axs[idx].set\_ylabel("Mean Evidence Accumulation")

    axs[idx].legend()

    idx += 1

# Display the plots

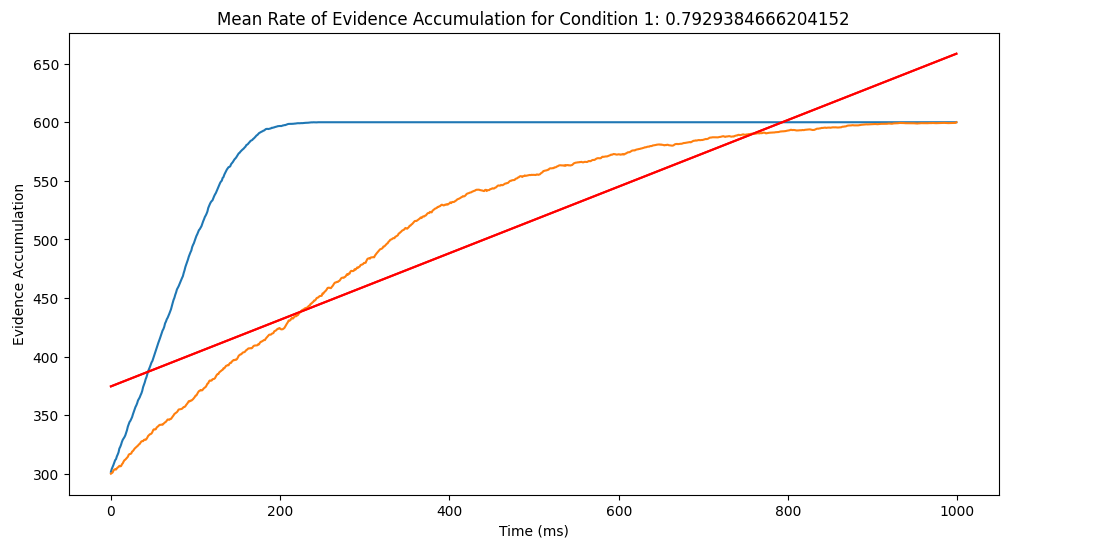
plt.tight\_layout()

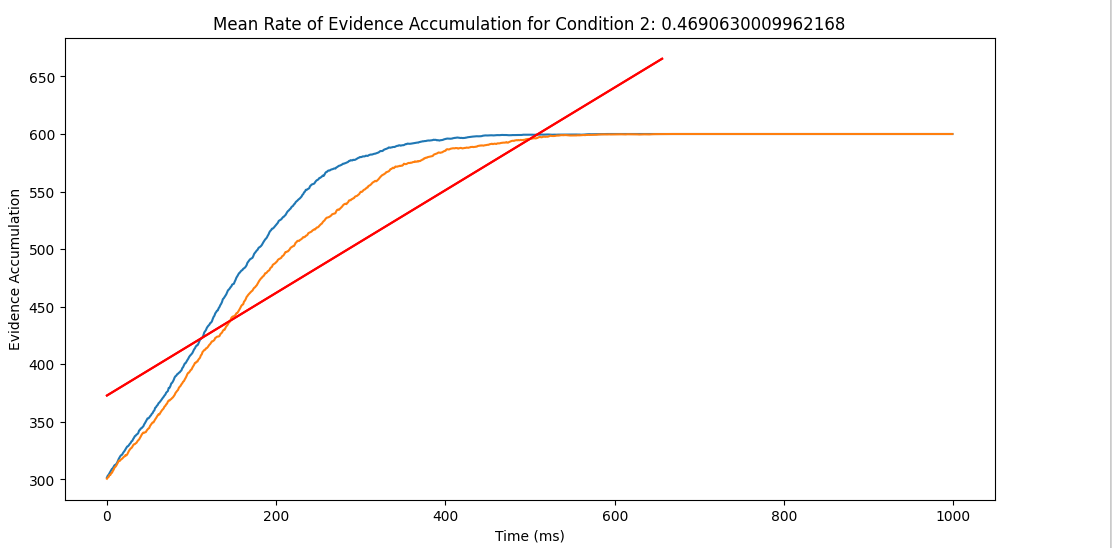
plt.show()

**Calculate the rate of mean evidence accumulation (across participants) for each of the**

**two conditions separately and report on the title of each subplot.**

**Graphs:**

****

****

# Calculate the rate of mean evidence accumulation (across participants) for each of the

# two conditions separately and report on the title of each subplot.

# [Hint: Fit a straight line to the data between the starting point of mean evidence

# accumulation and the earliest point of reaching the threshold of maximum evidence to

# calculate the average rate.]

# In[7]:

import scipy.io

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import linregress

# In[9]:

# Load the MATLAB file

data = scipy.io.loadmat("NDM\_Assignment2.mat")

data['NDM\_Assignment2']

# In[22]:

# Assuming that the data is stored in two variables 'Condition1' and 'Condition2'

condition1 = data['NDM\_Assignment2'][0]

condition2 = data['NDM\_Assignment2'][1]

def CalculateRate(condition, ConditionName):

    # Initialize a list where we store the rates of each participant

    Rates = []

    # Now looping over each participant's data using a while loop

    participant\_index = 0

    while participant\_index < len(condition):

        ParticipantData = condition[participant\_index]

        # Calculate the mean evidence accumulation across each trial

        MeanEvidence = np.mean(ParticipantData, axis=0)

        Start\_index = np.where(MeanEvidence >= 300)[0][0]

        End\_index = np.where(MeanEvidence >= 600)[0][0]

        # Fit a straight line to the data between start and end indices

        slope, intercept, \_, \_, \_ = linregress(range(Start\_index, End\_index+1), MeanEvidence[Start\_index:End\_index+1])

        # Append the slope (rate of evidence accumulation) to the list

        Rates.append(slope)

        # Increment the participant index

        participant\_index += 1

    # Calculate the mean rate across participants

    mean\_rate = np.mean(Rates)

    # Plot mean evidence accumulation for each participant with fitted line

    plt.figure(figsize=(12, 6))

    plt.title(f'Mean Rate of Evidence Accumulation for {ConditionName}: {mean\_rate}')

    plt.xlabel('Time (ms)')

    plt.ylabel('Evidence Accumulation')

    # Reset the participant index and loop again to create the plots

    participant\_index = 0

    while participant\_index < len(condition):

        ParticipantData = condition[participant\_index]

        mean\_evidence = np.mean(ParticipantData, axis=0)

        plt.plot(mean\_evidence)

        plt.plot(range(Start\_index, End\_index+1), intercept + slope\*range(Start\_index, End\_index+1), 'r', label='fitted line')

        # Increment the participant index

        participant\_index += 1

    plt.show()

# Calculate and plot rate for each condition

CalculateRate(condition1, 'Condition 1')

CalculateRate(condition2, 'Condition 2')

# In[ ]:

# In[ ]:

**Whole code:**

#!/usr/bin/env python

# coding: utf-8

# Create two subplots for the two conditions separately. Plot the mean evidence

# accumulation (across 100 trials) of each participant in grey colour and the mean

# evidence accumulation (across 30 participants) in blue colour in each subplot.

# In[20]:

import os

import scipy.io as sio

import numpy as np

import matplotlib.pyplot as pl

# In[21]:

# Load data from the .mat file

dataset = sio.loadmat("NDM\_Assignment2.mat")

# In[22]:

# Extract the data

dataset = dataset['NDM\_Assignment2']

fig, axs = plt.subplots(1, 2, figsize=(12, 6))

labels = ["Condition 1", "Condition 2"]

idx = 0

while idx < 2:

    label = labels[idx]

    data = dataset[:, idx]

    mean\_participants = np.empty((len(data), 1000))

    mean\_trials = np.empty((len(data), 1000))

    # Calculate the mean evidence accumulation

    i = 0

    while i < len(data):

        participant\_data = data[i]

        mean\_trials[i] = np.mean(participant\_data,axis =0 )

        i += 1

    # Calculate the mean evidence accumulation

    mean\_participants = np.mean(mean\_trials,axis =0)

    # Plot the mean evidence accumulation

    i = 0

    while i < len(mean\_trials):

        axs[idx].plot(mean\_trials[i], color='grey', alpha=0.5)

        i += 1

    # Plot the mean evidence accumulation

    axs[idx].plot(mean\_participants, color='blue', label="Mean across Participants")

    axs[idx].set\_title(f"{label} - Mean Evidence Accumulation")

    axs[idx].set\_xlabel("Time (ms)")

    axs[idx].set\_ylabel("Mean Evidence Accumulation")

    axs[idx].legend()

    idx += 1

# Display the plots

plt.tight\_layout()

plt.show()

# Calculate the rate of mean evidence accumulation (across participants) for each of the

# two conditions separately and report on the title of each subplot.

# [Hint: Fit a straight line to the data between the starting point of mean evidence

# accumulation and the earliest point of reaching the threshold of maximum evidence to

# calculate the average rate.]

# In[7]:

import scipy.io

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import linregress

# In[9]:

# Load the MATLAB file

data = scipy.io.loadmat("NDM\_Assignment2.mat")

data['NDM\_Assignment2']

# In[22]:

# Assuming that the data is stored in two variables 'Condition1' and 'Condition2'

condition1 = data['NDM\_Assignment2'][0]

condition2 = data['NDM\_Assignment2'][1]

def CalculateRate(condition, ConditionName):

    # Initialize a list where we store the rates of each participant

    Rates = []

    # Now looping over each participant's data using a while loop

    participant\_index = 0

    while participant\_index < len(condition):

        ParticipantData = condition[participant\_index]

        # Calculate the mean evidence accumulation across each trial

        MeanEvidence = np.mean(ParticipantData, axis=0)

        Start\_index = np.where(MeanEvidence >= 300)[0][0]

        End\_index = np.where(MeanEvidence >= 600)[0][0]

        # Fit a straight line to the data between start and end indices

        slope, intercept, \_, \_, \_ = linregress(range(Start\_index, End\_index+1), MeanEvidence[Start\_index:End\_index+1])

        # Append the slope (rate of evidence accumulation) to the list

        Rates.append(slope)

        # Increment the participant index

        participant\_index += 1

    # Calculate the mean rate across participants

    mean\_rate = np.mean(Rates)

    # Plot mean evidence accumulation for each participant with fitted line

    plt.figure(figsize=(12, 6))

    plt.title(f'Mean Rate of Evidence Accumulation for {ConditionName}: {mean\_rate}')

    plt.xlabel('Time (ms)')

    plt.ylabel('Evidence Accumulation')

    # Reset the participant index and loop again to create the plots

    participant\_index = 0

    while participant\_index < len(condition):

        ParticipantData = condition[participant\_index]

        mean\_evidence = np.mean(ParticipantData, axis=0)

        plt.plot(mean\_evidence)

        plt.plot(range(Start\_index, End\_index+1), intercept + slope\*range(Start\_index, End\_index+1), 'r', label='fitted line')

        # Increment the participant index

        participant\_index += 1

    plt.show()

# Calculate and plot rate for each condition

CalculateRate(condition1, 'Condition 1')

CalculateRate(condition2, 'Condition 2')

# In[ ]:

# In[ ]:

**ii) To compare the mean rate of evidence accumulation (across participants) between**

**the two conditions, conduct an appropriate statistical test and report the results with**

**test statistics and p values. What can be concluded about the motion coherence in the**

**two conditions from the results? [2+1 points]**

**(Hint: If the data in each of the two groups follow a normal distribution, use a parametric**

**statistical test for testing the difference of two independent group means. Otherwise,**

**use a suitable non-parametric counterpart of the parametric test.**

**https://in.mathworks.com/help/stats/hypothesis-tests-1.html). Normality assumption can be**

**checked using Lilliefors test.)**

T-test:

t-statistic: 12.87221400582511

P-value (t-test): 1.2035690396569046e-18

Difference in mean rate of evidence accumulation is significant of (t-test).

Now Using Mann-Whitney U test:

U-statistic: 900.0

P-value (Mann-Whitney U test): 3.019859359162157e-11

Difference in mean rate of evidence accumulation is significant of (Mann-Whitney U test).

[ ]:

**Code is:**

#!/usr/bin/env python

# coding: utf-8

# In[5]:

from scipy.stats import ttest\_ind, mannwhitneyu

import numpy as np

import scipy.stats as stats

import os

import scipy.io as sio

import numpy as np

import matplotlib.pyplot as pl

# In[7]:

data = sio.loadmat("NDM\_Assignment2.mat")

data['NDM\_Assignment2']

# In[13]:

# Extract data for both conditions

Cond1Data = data['NDM\_Assignment2'][:, 0]

Cond2Data = data['NDM\_Assignment2'][:, 1]

# Convert the data to float

MeanCond1 = np.array([np.mean(participant) for participant in Cond1Data], dtype=float)

MeanCond2 = np.array([np.mean(participant) for participant in Cond2Data], dtype=float)

# Perform a two-sample independent t-test

TValue, PValue = ttest\_ind(MeanCond1, MeanCond2)

# Perform a Mann-Whitney U test

Stat, PValueU = mannwhitneyu(MeanCond1, MeanCond2)

# Report the results

print("T-test:")

print(f"t-statistic: {TValue}")

print(f"P-value (t-test): {PValue}")

if PValue < 0.05:

    print("Difference in mean rate of evidence accumulation is significant  of (t-test).")

else:

    print("No   significant difference in mean rate of evidence accumulation of (t-test).")

print("\n Now Using Mann-Whitney U test:")

print(f"U-statistic: {Stat}")

print(f"P-value (Mann-Whitney U test): {PValueU}")

if PValueU < 0.05:

    print("Difference in mean rate of evidence accumulation is significant  of (Mann-Whitney U test).")

else:

    print("No   significant difference in mean rate of evidence accumulation o (Mann-Whitney U test).")

# In[ ]:

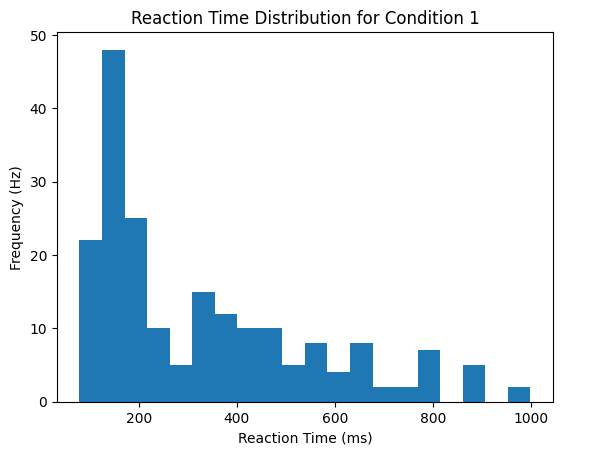
**B i. i) From the data, calculate reaction time (RT) for all 100 trials of each participant. Divide**

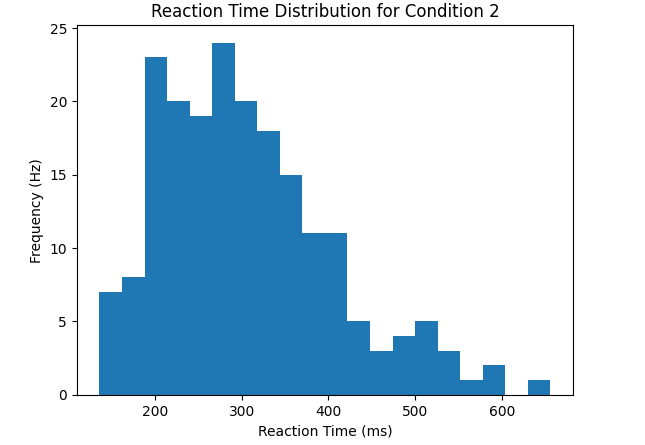
**the time axis in 20 bins and plot the mean RT for all 30 participants for both conditions**

**separately. Plot one histogram of the reaction time distribution for each condition.**

**Interpret the findings with respect to motion coherence.**

**Graphs:**

****

****

**Code:**

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import scipy.io

import numpy as np

import matplotlib.pyplot as plt

# In[3]:

mat\_data = scipy.io.loadmat('NDM\_Assignment2.mat')

# In[12]:

# Extract data for Condition 1 and Condition 2

cond1\_data = mat\_data['NDM\_Assignment2'][0]

cond2\_data = mat\_data['NDM\_Assignment2'][1]

# Function to calculate reaction time

def compute\_ReactionTime(data):

    ReactionTime = []  # Change rt=[] to ReactionTime=[]

    index = 0  # Change participant\_index to index

    while index < len(data):

        participant = data[index]

        TrailIndex = 0  # Change trial\_index to TrailIndex

        while TrailIndex < len(participant):

            trial = participant[TrailIndex]

            ReactionTime.append(np.where(trial >= 600)[0][0])

            TrailIndex += 1

        index += 1

    return np.array(ReactionTime)

# Calculate reaction time for both conditions

rt\_cond1 = compute\_ReactionTime(cond1\_data)

rt\_cond2 = compute\_ReactionTime(cond2\_data)

# Plot reaction time for both conditions

condition=1

plt.hist(rt\_cond1, bins=20)

plt.xlabel('Reaction Time (ms)')

plt.ylabel('Frequency (Hz)')

plt.title(f'Reaction Time Distribution for Condition {condition}')

plt.show()

condition=2

plt.hist(rt\_cond2, bins=20)

plt.xlabel('Reaction Time (ms)')

plt.ylabel('Frequency (Hz)')

plt.title(f'Reaction Time Distribution for Condition {condition}')

plt.show()

# In[ ]:

**ii) From the data, calculate the median bias of evidence accumulation (across**

**participants) separately for condition 1 and condition 2. Conduct the appropriate**

**statistical test to compare both with a reasoning. Report relevant test statistics and p**

**values. [Use the hint in the previous question to conduct the test]**

Ans:

Median Bias for Experiment 1: [600.0, 589.2167847205883]

Median Bias for Experiment 2: [600.0, 600.0]

Mann-Whitney U statistic: 1.0

P-value: 0.6170750774519738

T-value: -1.0

No, there is no significant difference between Experiment 1 and Experiment 2.

Since the p-value is greater than 0.05, there is no significant difference between Experiment 1 and Experiment 2.

Code:

# ii) From the data, calculate the median bias of evidence accumulation (across

# participants) separately for condition 1 and condition 2. Conduct the appropriate

# statistical test to compare both with a reasoning. Report relevant test statistics and p

# values. [Use the hint in the previous question to conduct the test]

# In[21]:

import scipy.io

import numpy as np

from scipy.stats import mannwhitneyu, ttest\_ind

# In[22]:

mat = scipy.io.loadmat('NDM\_Assignment2.mat')

mat

# In[25]:

# Extracting data for Experiment 1 and Experiment 2

experiment1\_data = mat['NDM\_Assignment2'][0]

experiment2\_data = mat['NDM\_Assignment2'][1]

# Initialize lists to store median bias for each participant

median\_bias\_experiment1 = []

median\_bias\_experiment2 = []

# Iterate over participants to calculate the median bias for each

i = 0

while i < len(experiment1\_data):

    median\_bias\_experiment1.append(np.median(experiment1\_data[i]))

    i += 1

i = 0

while i < len(experiment2\_data):

    median\_bias\_experiment2.append(np.median(experiment2\_data[i]))

    i += 1

# Perform Mann-Whitney U test to compare medians

u\_statistic, p\_val = mannwhitneyu(median\_bias\_experiment1, median\_bias\_experiment2)

# Perform t-test to calculate t-value

t\_val, \_ = ttest\_ind(median\_bias\_experiment1, median\_bias\_experiment2)

# Report the results

print("Median Bias for Experiment 1:", median\_bias\_experiment1)

print("Median Bias for Experiment 2:", median\_bias\_experiment2)

print("Mann-Whitney U statistic:", u\_statistic)

print("P-value:", p\_val)

print("T-value:", t\_val)

# Interpret the results

if p\_val < 0.05:

    print("Yes there is a significant difference between Experiment 1 and Experiment 2.")

else:

    print("No there is no significant difference between Experiment 1 and Experiment 2.")

# In[ ]: