Assignment 1: Learning and Memory PSY 306 (Winter 2024)

Name: Kunal Sharma

Roll Number: 2021331

Part B

1a)

Calculate the percent of total correct trials for each array set size for each participant and then the mean percent of total correct trials across all participants. Plot a simple bar diagram to represent the mean percent of total correct trials across all participants along with the standard error of the mean (as error bars) for each condition. [4 points]

Answer:

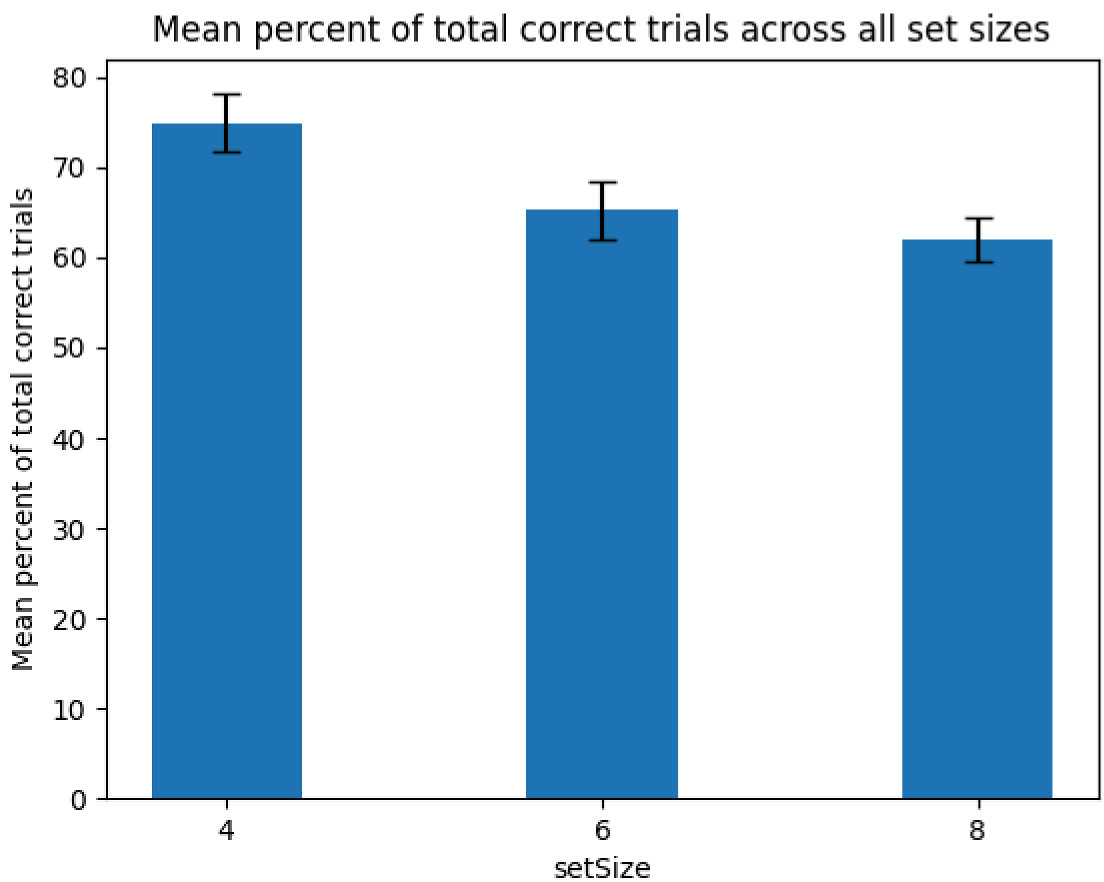
```
import matplotlib.pyplot as plt
import numpy as np
import scipy.io

def helper():
   i = 0
   while i < len(Subjects):   # Calculating correct trials of each participant
       Subject = Subjects[i]
       # Adding data in different 2D arrays
       Accuracy = Data_file[Subject]['accuracy'][0][0]
       setSize = Data_file[Subject]['setSize'][0][0]
       Correct_Trials = [0,0,0]
       Total_Trials = [0,0,0]
       j = 0
       while j < 192:       # Calculating correct trials of a participant for 4,6 and 8 set sizes.
           if setSize[j//4][j%4] == 4:
               Total_Trials[0] += 1
           elif setSize[j//4][j%4] == 6:
               Total_Trials[1] += 1
           elif setSize[j//4][j%4] == 8:
               Total_Trials[2] += 1

           if Accuracy[j//4][j%4] == 1 and setSize[j//4][j%4] == 4:
               Correct_Trials[0] += 1
           elif Accuracy[j//4][j%4] == 1 and setSize[j//4][j%4] == 6:
               Correct_Trials[1] += 1
           elif Accuracy[j//4][j%4] == 1 and setSize[j//4][j%4] == 8:
               Correct_Trials[2] += 1
```

```python
            j += 1
        k = 0
        while(k<=2):  # Appending the percentage into an array named correct
            Correct[k].append((Correct_Trials[k]/Total_Trials[k])*100)
            k+=1
        i+=1

    return Correct

if __name__=="__main__":

    # Loading data from the .mat file
    Data_file = scipy.io.loadmat('LM_A1_data.mat')

    # Taking participant keys
    Subjects = [f'p{i}' for i in range(1, 18)]

    Correct = [[],[],[]]
    helper()
    Mean = [0,0,0]
    Error = [0,0,0]
    Mean[0] = np.mean(Correct[0]) # Mean for setSize 4
    Error[0] = np.std(Correct[0]) / np.sqrt(len(Correct[0])) #Standard error for setSize 4
    Mean[1] = np.mean(Correct[1]) # Mean for setSize 6
    Error[1] = np.std(Correct[1]) / np.sqrt(len(Correct[1])) #Standard error for setSize 6
    Mean[2] = np.mean(Correct[2]) # Mean for setSize 8
    Error[2] = np.std(Correct[2]) / np.sqrt(len(Correct[2])) #Standard error for setSize 8
    print(f"Mean percentages: {Mean}")
    print(f"Standard Errors: {Error}")
    # Plotting the graph
    plt.bar(['4', '6', '8'], Mean, yerr=Error, capsize=5, width=0.4)
    plt.xlabel('setSize')
    plt.ylabel('Mean percent of total correct trials')
    plt.title('Mean percent of total correct trials across all set sizes')
    plt.show()
```

Output on the console:

> **Mean percentages:** [74.90808823529412, 65.25735294117646, 62.04044117647059]
> **Standard Errors:** [3.2325528060660815, 3.1932676563126416, 2.4135011076612893]

Bar Graph: To represent the mean percent of total correct trials across all participants along with the standard error of the mean (as error bars) for each condition is given below:



1b)

To compare the mean percent correct trials (across participants) across three conditions, conduct an appropriate statistical test and report the results with the appropriate test statistics and p values. Based on a comparison of the accuracies in all conditions, what can be concluded about the relationship between response accuracy and visual working memory capacity from the experimental data? [3+2+1 points]
[Hint: Check for assumptions of appropriate statistical test stepwise to conduct a test followed by appropriate post-hoc test as discussed in the class to solve the above. Indicate the main steps in your code with clear comments.]

Answer:

```
import pandas as pd
import pingouin as pg
import numpy as np
import scipy.io
```

```python
def helper():
    i = 0
    while i < len(Subjects):   # Calculating correct trials of each participant
        Subject = Subjects[i]
        # Adding data in different 2D arrays
        Accuracy = Data_file[Subject]['accuracy'][0][0]
        setSize = Data_file[Subject]['setSize'][0][0]
        Correct_Trials = [0,0,0]
        Total_Trials = [0,0,0]
        j = 0
        while j < 192:         # Calculating correct trials of a participant for 4,6 and 8 set sizes.
            if setSize[j//4][j%4] == 4:
                Total_Trials[0] += 1
            elif setSize[j//4][j%4] == 6:
                Total_Trials[1] += 1
            elif setSize[j//4][j%4] == 8:
                Total_Trials[2] += 1

            if Accuracy[j//4][j%4] == 1 and setSize[j//4][j%4] == 4:
                Correct_Trials[0] += 1
            elif Accuracy[j//4][j%4] == 1 and setSize[j//4][j%4] == 6:
                Correct_Trials[1] += 1
            elif Accuracy[j//4][j%4] == 1 and setSize[j//4][j%4] == 8:
                Correct_Trials[2] += 1
            j += 1
        k = 0
        while(k<=2):  # Appending the percentage into an array named correct
            Correct[k].append((Correct_Trials[k]/Total_Trials[k])*100)
            k+=1
        i+=1
    return Correct

if __name__=="__main__":

    # Loading data from the .mat file
    Data_file = scipy.io.loadmat('LM_A1_data.mat')
    # Taking participant keys
    Subjects = [f'p{i}' for i in range(1, 18)]
    Correct = [[],[],[]]
    helper()
    percentages = Correct[0] + Correct[1] + Correct[2]  # Merging the percentages lists.
    setSizes = [4,6,8]
    setSizes = np.repeat(setSizes, 17)
    setSizes = setSizes.tolist()
```

```
    array = np.tile(Subjects, 3)  # Copy the array three times
    participants = array.tolist() # Convert the resulting array back to a list

    data_frame = pd.DataFrame({'Participant': participants, 'Set Size': setSizes, 'Percentage':
percentages})
    # Performing Mauchly's test to check assumption of sphericity
    result1 = pg.sphericity(data=data_frame, dv='Percentage', subject='Participant', within='Set
Size')[-1]
    print(f"Mauchly's Test Results: \np-value = {result1}")

    # Performing Shapiro-Wilk's test to check assumption of Normality
    result2 = pg.normality(data=data_frame, dv='Percentage', group='Set Size')
    print(f"Shapiro-Wilk's Test Results: \n{result2}")

    #Performing Levene's test to check assumption of Equal Variances
    result3 = pg.homoscedasticity(data_frame, dv='Percentage', group='Set Size')
    print(f"Levene's Test Results: {result3}")

    #Repeated measures anova
    result4 = pg.rm_anova(dv='Percentage', within='Set Size', subject='Participant',
data=data_frame, detailed=True)
    print(f"Repeated Measures Anova Test Results: {result4}")

    # Post_hocs - Tukey's test
    result5 = tu.MultiComparison(data_frame['Percentage'],data_frame['Set Size']).tukeyhsd()
    print(f"Tukey's test results: {result5}")
```

Output on the console:

Mauchly's Test Results:
p-value = 0.243469218420624447

Shapiro-Wilk's Test Results:

Set Size	W	pval	normal
4	0.947274	0.414862	True
6	0.950353	0.462133	True
8	0.914128	0.117474	True

Levene's Test Results:

	W	pval	equal_var
levene	1.11581	0.335999	True

Repeated Measures Anova Test Results:

	Source	SS	DF	MS	F	p-unc	ng2	eps
0	Set Size	1524.682138	2	762.341069	15.228305	0.000023	0.16618	0.853468
1	Error	1601.945466	32	50.060796	NaN	NaN	NaN	NaN

Post Hocs Results:
Tukey's test results: Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
4	6	-9.6507	0.0765	-20.1232	0.8218	False
4	8	-12.8676	0.0126	-23.3402	-2.3951	True
6	8	-3.2169	0.7393	-13.6894	7.2556	False

Results:

The statistical analysis of the data gives many key findings. Firstly, Mauchly's Test indicated that there is no violation of sphericity among the three conditions, with a p-value of 0.3705, suggesting equal variances of differences between all possible pairs of conditions. Additionally, the Shapiro-Wilk's Test demonstrated that all set sizes exhibit normal distribution ($p > 0.05$). Levene's Test showed no significant difference in variances across different set sizes ($p > 0.05$). The RM Anova Test revealed a significant difference in average accuracies among conditions ($F(2,32)=15.22$, $p < 0.05$), prompting further investigation through post hoc tests. These post hoc(Tukey) tests identified significant differences between set sizes 4 & 8, as their corrected p-values were below 0.05, indicating significantly distinct average accuracy values. Conversely, the difference in average accuracies between set sizes 4 & 6 and 6 & 8 was not significantly different, as the corrected p-value exceeded 0.05.

At last, we can conclude that the set size is inversely proportional to the average response accuracy of the participants i.e. if the set size increases the average accuracy of the participants decreases, or if the set size decreases the average accuracy of the participants increases.

2a)

Calculate the 'd prime' for each array size for all trials for each participant and average 'd prime' across participants. Create a bar diagram for each array size showing mean 'd prime' (across participants) and standard error of the mean as error bars. [5 points]

Answer:

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.io
import scipy.stats as stats
```

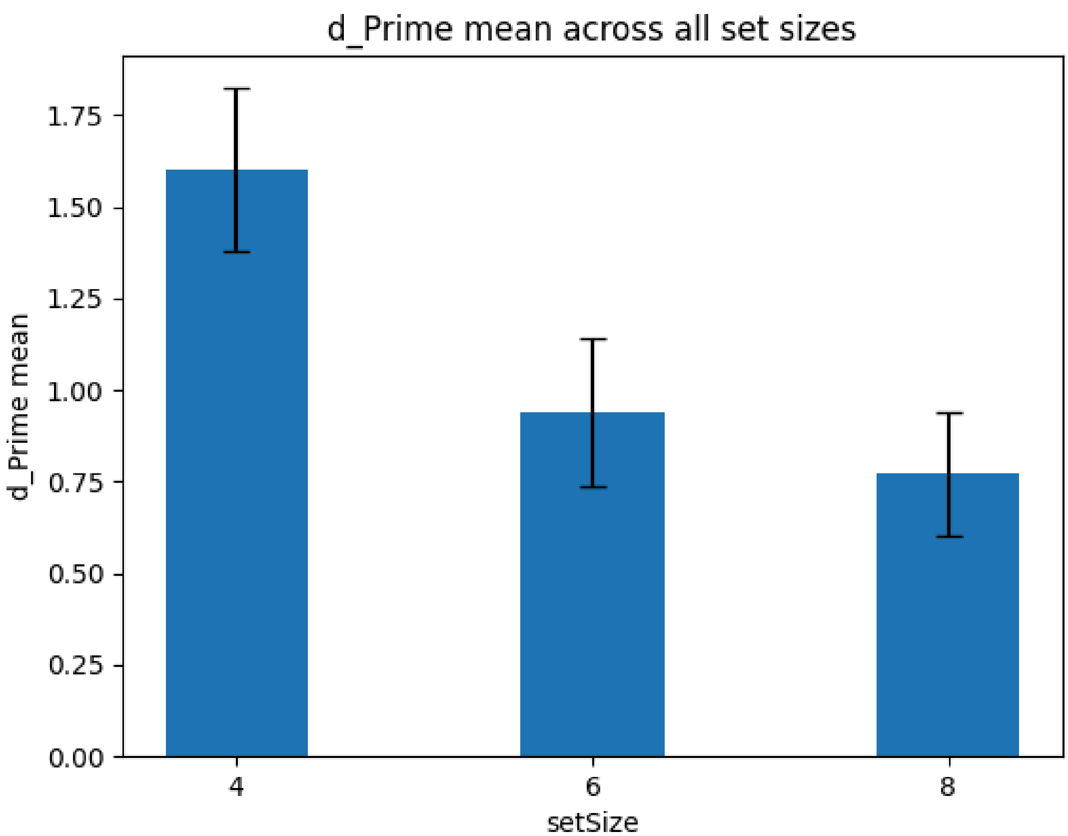
```
def helper():
   i = 0
   while i < len(Subjects):   # Calculating d_Prime of each participant
       Subject = Subjects[i]
       # Adding data in different 2D arrays
       Change = Data_file[Subject]['change'][0][0]
       Accuracy = Data_file[Subject]['accuracy'][0][0]
       setSize = Data_file[Subject]['setSize'][0][0]
       Number_of_hits = [0,0,0]
       False_alarms = [0,0,0]
       Change_1 = [0,0,0]
       Change_0 = [0,0,0]
       j = 0
       while j < 192:          # Calculating d_Prime of a participant for 4,6 and 8 set size.
           temp1 = j//4
           temp2 = j%4
           if Change[temp1][temp2] == 1 and setSize[temp1][temp2] == 4:
               Change_1[0] +=1
           elif Change[temp1][temp2] == 0 and setSize[temp1][temp2] == 4:
               Change_0[0] +=1
           elif Change[temp1][temp2] == 1 and setSize[temp1][temp2] == 6:
               Change_1[1] +=1
           elif Change[temp1][temp2] == 0 and setSize[temp1][temp2] == 6:
               Change_0[1] +=1
           elif Change[temp1][temp2] == 1 and setSize[temp1][temp2] == 8:
               Change_1[2] +=1
           elif Change[temp1][temp2] == 0 and setSize[temp1][temp2] == 8:
               Change_0[2] +=1
           if Change[temp1][temp2] == 0 and Accuracy[temp1][temp2] == 0 and setSize[temp1][temp2]
== 4:
               False_alarms[0] += 1
           elif Change[temp1][temp2] == 1 and Accuracy[temp1][temp2] == 1 and setSize[temp1][temp2]
== 4:
               Number_of_hits[0] += 1
           elif Change[temp1][temp2] == 0 and Accuracy[temp1][temp2] == 0 and setSize[temp1][temp2]
== 6:
               False_alarms[1] += 1
           elif Change[temp1][temp2] == 1 and Accuracy[temp1][temp2] == 1 and setSize[temp1][temp2]
== 6:
               Number_of_hits[1] += 1
           elif Change[temp1][temp2] == 0 and Accuracy[temp1][temp2] == 0 and setSize[temp1][temp2]
== 8:
               False_alarms[2] += 1
```

```
            elif Change[temp1][temp2] == 1 and Accuracy[temp1][temp2] == 1 and setSize[temp1][temp2]
== 8:
                Number_of_hits[2] += 1

            j += 1

        k = 0
        while(k<=2): # Appending the d prime value into an array named d_Prime
            d_Prime[k].append(stats.norm.ppf(Number_of_hits[k]/Change_1[k]) -
stats.norm.ppf(False_alarms[k]/Change_0[k]))
            k+=1
        i+=1
    return d_Prime

if __name__=="__main__":

    # Loading data from the .mat file
    Data_file = scipy.io.loadmat('LM_A1_data.mat')

    # Taking participant keys
    Subjects = [f'p{i}' for i in range(1, 18)]
    d_Prime = [[],[],[]]

    print(helper())
    Mean = [0,0,0]
    Error = [0,0,0]
    Mean[0] = np.mean(d_Prime[0]) # Mean for setSize 4
    Error[0] = np.std(d_Prime[0]) / np.sqrt(len(d_Prime[0])) #Standard error for setSize 4
    Mean[1] = np.mean(d_Prime[1]) # Mean for setSize 6
    Error[1] = np.std(d_Prime[1]) / np.sqrt(len(d_Prime[1])) #Standard error for setSize 6
    Mean[2] = np.mean(d_Prime[2]) # Mean for setSize 8
    Error[2] = np.std(d_Prime[2]) / np.sqrt(len(d_Prime[2])) #Standard error for setSize 8

    print(f"Average d_prime: {Mean}")
    print(f"Standard Error: {Error}")
    # Plotting the graph
    plt.bar(['4', '6', '8'], Mean, yerr=Error, capsize=5, width=0.4)
    plt.xlabel('setSize')
    plt.ylabel('d_Prime mean')
    plt.title('d_Prime mean across all set sizes')
    plt.show()
```

Output on the console:

Average d_primes: [1.6007323915730254, 0.9375675732377838, 0.7711763832559827]
Standard Errors: [0.22217459936936373, 0.20097877150277169, 0.16703690842575525]

Bar Graph: Created bar graph for each array size showing mean 'd prime' (across participants) and standard error of the mean as error bars which is given below:



2b)

To compare the mean 'd prime' (across participants) across three conditions (array size), conduct an appropriate statistical test and report the results with test statistics and p values. Interpret the results of the test statistics. [2+2+1 points]
[Hint: Check for assumptions of appropriate statistical test stepwise to conduct a test followed by appropriate post-hoc test as discussed in the class to solve the above. Indicate the main steps in your code with clear comments.]

Answer:

```
import numpy as np
import scipy.io
import scipy.stats as stats
import pandas as pd
import pingouin as pg
def helper():
    i = 0
```

```
    while i < len(Subjects):   # Calculating d_Prime of each participant
        Subject = Subjects[i]
        # Adding data in different 2D arrays
        Change = Data_file[Subject]['change'][0][0]
        Accuracy = Data_file[Subject]['accuracy'][0][0]
        setSize = Data_file[Subject]['setSize'][0][0]
        Number_of_hits = [0,0,0]
        False_alarms = [0,0,0]
        Change_1 = [0,0,0]
        Change_0 = [0,0,0]
        j = 0
        while j < 192:         # Calculating d_Prime of a participant for 4,6 and 8 set sizes.
            temp1 = j//4
            temp2 = j%4
            if Change[temp1][temp2] == 1 and setSize[temp1][temp2] == 4:
                Change_1[0] +=1
            elif Change[temp1][temp2] == 0 and setSize[temp1][temp2] == 4:
                Change_0[0] +=1
            elif Change[temp1][temp2] == 1 and setSize[temp1][temp2] == 6:
                Change_1[1] +=1
            elif Change[temp1][temp2] == 0 and setSize[temp1][temp2] == 6:
                Change_0[1] +=1
            elif Change[temp1][temp2] == 1 and setSize[temp1][temp2] == 8:
                Change_1[2] +=1
            elif Change[temp1][temp2] == 0 and setSize[temp1][temp2] == 8:
                Change_0[2] +=1
            if Change[temp1][temp2] == 0 and Accuracy[temp1][temp2] == 0 and setSize[temp1][temp2]
== 4:
                False_alarms[0] += 1
            elif Change[temp1][temp2] == 1 and Accuracy[temp1][temp2] == 1 and setSize[temp1][temp2]
== 4:
                Number_of_hits[0] += 1
            elif Change[temp1][temp2] == 0 and Accuracy[temp1][temp2] == 0 and setSize[temp1][temp2]
== 6:
                False_alarms[1] += 1
            elif Change[temp1][temp2] == 1 and Accuracy[temp1][temp2] == 1 and setSize[temp1][temp2]
== 6:
                Number_of_hits[1] += 1
            elif Change[temp1][temp2] == 0 and Accuracy[temp1][temp2] == 0 and setSize[temp1][temp2]
== 8:
                False_alarms[2] += 1
            elif Change[temp1][temp2] == 1 and Accuracy[temp1][temp2] == 1 and setSize[temp1][temp2]
== 8:
                Number_of_hits[2] += 1
```

```
            j += 1
        k = 0
        while(k<=2): # Appending the d prime value into an array named d_Prime
            d_Prime[k].append(stats.norm.ppf(Number_of_hits[k]/Change_1[k]) -
stats.norm.ppf(False_alarms[k]/Change_0[k]))
            k+=1
        i+=1
    return d_Prime

if __name__=="__main__":
    # Loading data from the .mat file
    Data_file = scipy.io.loadmat('LM_A1_data.mat')
    # Taking participant keys
    Subjects = [f'p{i}' for i in range(1, 18)]
    d_Prime = [[],[],[]]
    helper()
    d_Primes = d_Prime[0] + d_Prime[1] + d_Prime[2]  # Merging the d_Primes lists.
    setSizes = [4,6,8]
    setSizes = np.repeat(setSizes, 17)
    setSizes = setSizes.tolist()
    array = np.tile(Subjects, 3)  # Copy the list three times
    participants = array.tolist() # Convert the resulting array back to a list
    data_frame = pd.DataFrame({'Participant': participants, 'Set Size': setSizes, 'd_Prime':
d_Primes})
    # Performing Mauchly's test to check assumption of sphericity
    result1 = pg.sphericity(data=data_frame, dv='d_Prime', subject='Participant', within='Set
Size')[-1]
    print(f"Mauchly's Test Results: \np-value = {result1}")

    # Performing Shapiro-Wilk's test to check assumption of Normality
    result2 = pg.normality(data=data_frame, dv='d_Prime', group='Set Size')
    print(f"Shapiro-Wilk's Test Results: \n{result2}")

    #Performing Levene's test to check assumption of Equal Variances
    result3 = pg.homoscedasticity(data_frame, dv='d_Prime', group='Set Size')
    print(f"Levene's Test Results: {result3}")

    # Friedman's test
    result4 = pg.friedman(data=data_frame, dv='d_Prime', within='Set Size', subject='Participant')
    print(f"Friedman's Test Results: {result4}")

    #Post_hocs - bonferroni test
    result5 = pg.pairwise_tests(dv='d_Prime', within='Set Size', subject='Participant',
padjust='bonferroni', data=data_frame)
```

```
print(f"Post Hocs Results: {result5}")
```

Output on the console:

Mauchly's Test Results:
p-value = 0.370540450149887

Shapiro-Wilk's Test Results:

Set Size	W	pval	normal
4	0.978443	0.941618	True
6	0.959636	0.624648	True
8	0.881288	0.033364	False

Levene's Test Results:

	W	pval	equal_var
levene	0.957751	0.390966	True

Friedman's Test Results:

	Source	W	ddof1	Q	p-unc
Friedman	Set Size	0.387543	2	13.176471	0.001376

Post Hocs Results:

	Contrast	A	B	Paired	Parametric	T	dof	alternative	p-unc	p-corr	p-adjust	BF10	hedges
0	Set Size	4	6	True	True	5.316	16.0	two-sided	0.000070	0.000209	bonferroni	391.393	0.719
1	Set Size	4	8	True	True	4.879	16.0	two-sided	0.000167	0.000501	bonferroni	180.268	0.969
2	Set Size	6	8	True	True	1.005	16.0	two-sided	0.329444	0.988331	bonferroni	0.386	0.206

Results:

The statistical analysis of the data gives many key findings. Firstly, Mauchly's Test indicated that there is no violation of sphericity among the three conditions, with a p-value of 0.3705, suggesting equal variances of differences between all possible pairs of conditions. Additionally, the Shapiro-Wilk's Test demonstrated that set sizes 4 and 6 exhibit normal distribution ($p > 0.05$), while set size 8 deviates from normality ($p < 0.05$). Levene's Test showed no significant difference in variances across different set sizes ($p > 0.05$). The RM Anova Test cannot be used as for set size 8 the data turns out to be deviating from normality so instead of RM Anova, Friedman's Test is being used which revealed a significant difference in average d-primes as $p < 0.05$, prompting further investigation through post hoc tests. These post hoc(Bonferroni) tests identified significant differences between set sizes 4 & 6 and 4 & 8, as their corrected p-values were below 0.05, indicating significantly distinct average d-prime values. Conversely, the difference in average d-primes between set sizes 6 & 8 was not significantly different, as the corrected p-value exceeded 0.05.