Finalproject_summarized_6.5-final

June 8, 2024

https://docs.google.com/document/d/10CuEpVO_PUfcHJh5bTfpkghqMNlCtT4msHg_zRjYrdQ/edit

file:///C:/Users/Muru/Downloads/FinalProject-GroupXYZ.pdf

file:///C:/Users/Muru/Downloads/FinalProject-GroupXYZ_3.pdf

1 UC San Diego: Neural Data Science

1.1 Analysis on Memory and Anxiety Medication

1.2 Permissions

Place an `X` in the appropriate bracket below to specify if you would like your group's project to be made available to the public. (Note that student names will be included (but PIDs will be scraped from any groups who include their PIDs).

- ['X'] YES - make available
- ☐ NO - keep private

2 Names

- Brian Lee
- Manjari Muruganandam
- Bianca Yongyuth

3 Overview

We have chosen the data set "Memory Test on Drugged Islanders" collected by Steve Ahn to help us understand and solve our research question revolving around the interconnections between anti-anxiety medication, emotional priming, and memory recall. Our group plans to import the available CSV file to analyze the data further with Python due to its pandas library allowing for proper data analysis, manipulation, and statistical computation.These plots will provide a clear and intuitive way to compare the data on memory recall scores across the different levels of drug dosages and emotional priming conditions.

Research Question

Does the efficacy of Alprazolam and Triazolam in modulating memory recall vary depending on dosage and the emotional priming of happy or sad memories?

Does the medications Alprazolam and Triazolam alongside the emotional priming of happy or sad memories significantly affect memory recall????

3.1 Background & Prior Work

The motivation behind our research question stems from our curiosity about the possible relationships between anti-anxiety medication, emotional priming, and memory recall. By investigating the efficacy of Alprazolam and Triazolam across different dosages and emotional priming contexts, we aim to analyze how these medications regulate memory processes. Given the prevalence of benzodiazepine usage and the growing interest in memory enhancement and emotional regulation, analyzing how these drugs interact can provide valuable insights into both clinical and cognitive fields. It is particularly intriguing to understand the extent to which memory recall is affected, and whether the benefits outweigh the costs of using these medications in select populations.

The broader context involves understanding the relationship between anxiety medications and memory recall, as well as how factors such as age and genetics influence the medications' adverse side effects, guiding clinicians in prescribing therapeutics. According to the paper "Savoring the Past: Positive Memories Evoke Value Representations in the Striatum," positive memories enhance memory recall and resiliency, correlating with increased activity in the striatum and medial prefrontal cortex. If anxiety medication negatively impacts mood, it could reduce activity in these brain regions, thereby diminishing resilience and memory recall. Additionally, the study "With Sadness Comes Accuracy; With Happiness, False Memory: Mood and the False Memory Effect" explains that the Deese-Roediger-McDermott paradigm reveals that positive affective cues encourage relational processing during encoding, whereas negative affective cues encourage item-specific processing. Thus, anxiety medications impacting mood could shift processing strategies during encoding, selectively affecting memory recall. Furthermore, the paper "Analysis of Adverse Behavioral Effects of Benzodiazepines With a Discussion on Drawing Scientific Conclusions from the FDA's Spontaneous Reporting System" highlights the link between benzodiazepines, such as Alprazolam, and mental and behavioral abnormalities due to CNS suppression and the binding of benzodiazepine to GABA receptors. By integrating findings from these studies, we aim to deepen our understanding of how anxiety medications influence cognitive processes, ultimately contributing to more informed clinical practices.

References (include links): - 1) https://www.google.com/url?q=https://www.sciencedirect.com/science/article/pii
- 2) https://www.google.com/url?q=https://www.jstor.org/stable/40064315&sa=D&source=docs&ust=17177279
- 3) https://www.google.com/url?q=http://www.jstor.org/stable/43854146&sa=D&source=docs&ust=171772798

4 Hypothesis

[ ]:

5 Dataset(s)

- Dataset Name: Memory Test on Drugged Islanders
- Link to the dataset: https://www.kaggle.com/datasets/steveahn/memory-test-on-drugged-islanders-data/data
- Number of observations: 198 Participants

This dataset contains information on memory test scores of participants before and after administration of anti-anxiety medications, Alprazolam and Triazolam, along with varying dosages and emotional priming contexts. The data includes demographic details and memory score differences, enabling analysis of how these factors influence memory recall.

6 Data Wrangling

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


import plotly.graph_objects as go
import plotly.express as px
import plotly.io as pio
pio.templates.default = "plotly_dark"
from plotly.subplots import make_subplots

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

df = pd.read_csv('Islander_data.csv')
df

#diff = after - before
```

[1]:

	first_name	last_name	age	Happy_Sad_group	Dosage	Drug	\
0	Bastian	Carrasco	25	H	1	A	
1	Evan	Carrasco	52	S	1	A	
2	Florencia	Carrasco	29	H	1	A	
3	Holly	Carrasco	50	S	1	A	
4	Justin	Carrasco	52	H	1	A	
..	…	…	…	…	…	…	
193	Jacob	Novak	52	H	3	T	
194	Teo	Steiner	41	S	3	T	
195	Alexander	Takahashi	54	S	3	T	
196	Alexandere	Takahashi	40	H	3	T	
197	Chloe	Takahashi	32	S	3	T	

	Mem_Score_Before	Mem_Score_After	Diff
0	63.5	61.2	-2.3
1	41.6	40.7	-0.9
2	59.7	55.1	-4.6
3	51.7	51.2	-0.5
4	47.0	47.1	0.1

```
..                ...               ...   ...
193              71.3              74.3   3.0
194              72.5              70.4  -2.1
195              30.8              33.1   2.3
196              53.6              53.8   0.2
197              43.1              42.1  -1.0

[198 rows x 9 columns]
```

7 Data Cleaning

7.0.1 remove Nan and duplicate values

[2]:
```
# Data Cleaning
df.drop_duplicates(inplace = True)  # Remove duplicates
df.dropna(inplace = True)  # Remove rows with null values
df
```

[2]:
```
     first_name  last_name  age Happy_Sad_group  Dosage Drug  \
0       Bastian   Carrasco   25                H       1    A
1          Evan   Carrasco   52                S       1    A
2     Florencia   Carrasco   29                H       1    A
3         Holly   Carrasco   50                S       1    A
4        Justin   Carrasco   52                H       1    A
..          ...        ...  ...              ...     ...  ...
193       Jacob      Novak   52                H       3    T
194         Teo    Steiner   41                S       3    T
195   Alexander  Takahashi   54                S       3    T
196  Alexandere  Takahashi   40                H       3    T
197       Chloe  Takahashi   32                S       3    T

     Mem_Score_Before  Mem_Score_After  Diff
0                63.5             61.2  -2.3
1                41.6             40.7  -0.9
2                59.7             55.1  -4.6
3                51.7             51.2  -0.5
4                47.0             47.1   0.1
..                ...              ...   ...
193              71.3             74.3   3.0
194              72.5             70.4  -2.1
195              30.8             33.1   2.3
196              53.6             53.8   0.2
197              43.1             42.1  -1.0

[198 rows x 9 columns]
```

7.0.2 check for inconsistencies in data

[3]:
```
df.describe(include='all')  # Summary statistics
```

[3]:

	first_name	last_name	age	Happy_Sad_group	Dosage	Drug
count	198	198	198.000000	198	198.000000	198
unique	139	18	NaN	2	NaN	3
top	Jun	Durand	NaN	H	NaN	A
freq	5	44	NaN	99	NaN	67
mean	NaN	NaN	39.530303	NaN	1.989899	NaN
std	NaN	NaN	12.023099	NaN	0.818504	NaN
min	NaN	NaN	24.000000	NaN	1.000000	NaN
25%	NaN	NaN	30.000000	NaN	1.000000	NaN
50%	NaN	NaN	37.000000	NaN	2.000000	NaN
75%	NaN	NaN	48.000000	NaN	3.000000	NaN
max	NaN	NaN	83.000000	NaN	3.000000	NaN

	Mem_Score_Before	Mem_Score_After	Diff
count	198.000000	198.000000	198.000000
unique	NaN	NaN	NaN
top	NaN	NaN	NaN
freq	NaN	NaN	NaN
mean	57.967677	60.922222	2.954545
std	15.766007	18.133851	10.754603
min	27.200000	27.100000	-40.400000
25%	46.525000	47.175000	-3.175000
50%	54.800000	56.750000	1.700000
75%	68.400000	73.250000	5.925000
max	110.000000	120.000000	49.000000

[4]:
```
df.info() # shows the datatype of each column (ensures we have one dtype per column)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 198 entries, 0 to 197
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   first_name        198 non-null    object
 1   last_name         198 non-null    object
 2   age               198 non-null    int64
 3   Happy_Sad_group   198 non-null    object
 4   Dosage            198 non-null    int64
 5   Drug              198 non-null    object
 6   Mem_Score_Before  198 non-null    float64
 7   Mem_Score_After   198 non-null    float64
 8   Diff              198 non-null    float64
dtypes: float64(3), int64(2), object(4)
```

```
memory usage: 15.5+ KB
```

[5]:
```python
# Checking unique values for categorical columns
categorical_columns = df.select_dtypes(include=['object']).columns
for column in categorical_columns:
    unique_values = df[column].unique()
    print(f"\nUnique values for column '{column}':")
    print(unique_values)

# Checking range of values for numerical columns
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
for column in numerical_columns:
    min_value = df[column].min()
    max_value = df[column].max()
    print(f"\nRange of values for column '{column}':")
    print(f"Min: {min_value}, Max: {max_value}")
```

```
Unique values for column 'first_name':
['Bastian' 'Evan' 'Florencia' 'Holly' 'Justin' 'Liam' 'Ava' 'Jamie'
 'Josefa' 'Mark' 'Maximiliano' 'Ayano' 'Grace' 'Ai' 'Kaito' 'Jun' 'Takuya'
 'Justine' 'Nik' 'Carlos' 'Anna' 'Daichi' 'Dean' 'Riley' 'Sofia' 'Darren'
 'Fernado' 'Misaki' 'Orla' 'Robert' 'Valentina' 'Ryan' 'Jose' 'Shota'
 'Anthony' 'Nina' 'Lara' 'Daiki' 'Felipe' 'Camila' 'Hama' 'Miki' 'Riko'
 'Benjamin' 'Hina' 'Kevin' 'Takahiro' 'Megan' 'Akane' 'Ren' 'Laura'
 'Ariane' 'Naoto' 'Jade' 'Tomax' 'Ami' 'Mai' 'Yuta' 'Marianne' 'Mathis'
 'Martina' 'William' 'Tatsuya' 'Raphael' 'Fabian' 'Paula' 'Sho'
 'Frederique' 'Killian' 'Jeremy' 'Lan' 'Riku' 'Rin' 'Karin' 'Christian'
 'Ignacio' 'Joaquin' 'Momoko' 'Sara' 'Alejandra' 'Rok' 'Carla' 'Alexia'
 'Nanami' 'Victor' 'Sophia' 'Kana' 'Aya' 'Eva' 'Shun' 'Adam' 'Ayaka'
 'Ryouta' 'Antoine' 'Ciara' 'Mitsuku' 'Takumi' 'Kenta' 'Pia' 'Erin'
 'Michael' 'Sakura' 'Chloe' 'Tobias' 'Shauna' 'Ross' 'Daniel' 'Asuka'
 'Emma' 'Nathan' 'Akira' 'David' 'Manuel' 'Sean' 'Sebastian' 'Sophie'
 'Diego' 'Dylan' 'Millaray' 'Cristobal' 'Nicole' 'Elias' 'James' 'Conor'
 'Jacob' 'Maximilian' 'Aaron' 'Luka' 'Amy' 'Haru' 'Lukas' 'Ellen' 'Naoki'
 'Rina' 'Noemie' 'Gregor' 'Teo' 'Alexander' 'Alexandere']

Unique values for column 'last_name':
['Carrasco' 'Durand' 'Gonzalez' 'Kennedy' 'Lopez' 'McCarthy' 'Morin'
 'Price' 'Summers' 'Takahashi' 'Bernard' 'Hajek' 'Rodriguez' 'Steiner'
 'Connolly' 'Castro' 'Fiala' 'Novak']

Unique values for column 'Happy_Sad_group':
['H' 'S']

Unique values for column 'Drug':
['A' 'S' 'T']
```

```
Range of values for column 'age':
Min: 24, Max: 83

Range of values for column 'Dosage':
Min: 1, Max: 3

Range of values for column 'Mem_Score_Before':
Min: 27.2, Max: 110.0

Range of values for column 'Mem_Score_After':
Min: 27.1, Max: 120.0

Range of values for column 'Diff':
Min: -40.4, Max: 49.0
```

Overall, dtypes, values, and the df are clean and we are ready for data visualization. There are few inconsistencies.

8 Data Visualization

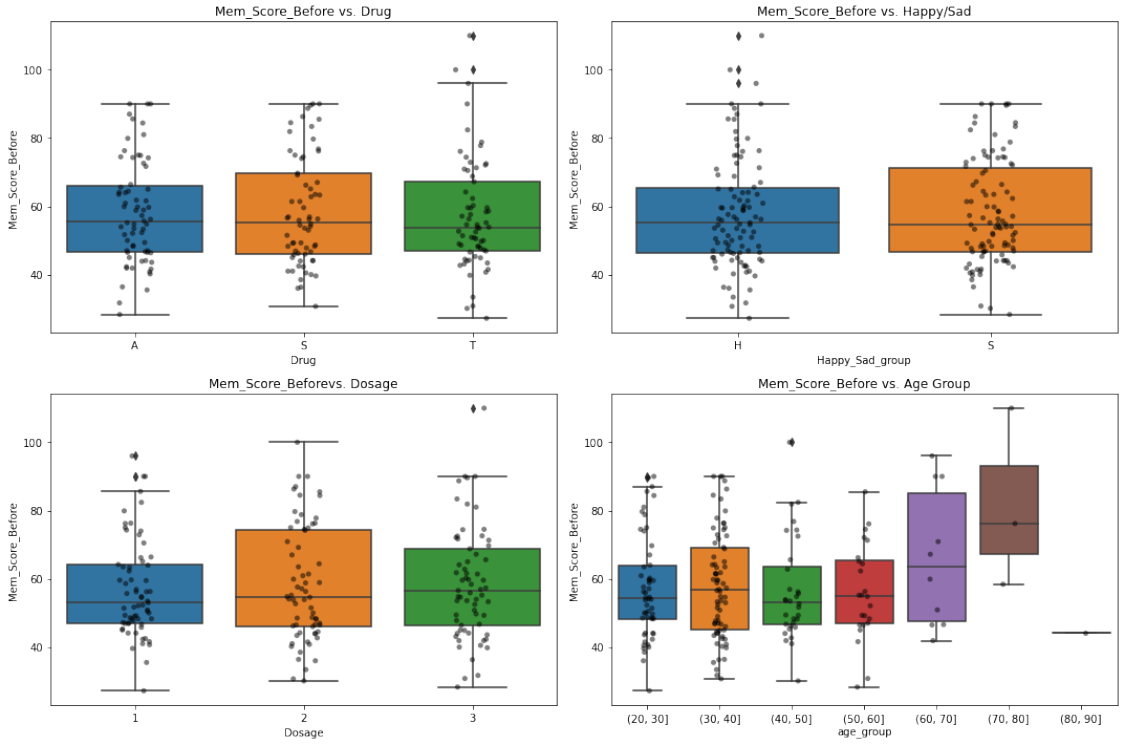
8.1 checking for confounds

Check to see if the averages between groups are significantly different * if middle line of boxplot is outside the box of the other boxplot, there is a significant difference * That is not the case for any of these

check for outliers * black points outside of boxplot whiskers are considered outliers

[6]:
```
# Box plots
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# diff vs. drug
sns.boxplot(data=df, x='Drug', y='Mem_Score_Before', ax=axes[0, 0])
sns.stripplot(data=df, x='Drug', y='Mem_Score_Before', color='black', alpha=0.
 ↪5, ax=axes[0, 0])
axes[0, 0].set_title('Mem_Score_Before vs. Drug')

# diff vs. happy/sad
sns.boxplot(data=df, x='Happy_Sad_group', y='Mem_Score_Before', ax=axes[0, 1])
sns.stripplot(data=df, x='Happy_Sad_group', y='Mem_Score_Before',␣
 ↪color='black', alpha=0.5, ax=axes[0, 1])
axes[0, 1].set_title('Mem_Score_Before vs. Happy/Sad')

# diff vs. dosage
sns.boxplot(data=df, x='Dosage', y='Mem_Score_Before', ax=axes[1, 0])
sns.stripplot(data=df, x='Dosage', y='Mem_Score_Before', color='black', alpha=0.
 ↪5, ax=axes[1, 0])
axes[1, 0].set_title('Mem_Score_Beforevs. Dosage')
```

```
# diff vs. age (grouped by ranges)
age_bins = [20, 30, 40, 50, 60, 70, 80, 90]
df['age_group'] = pd.cut(df['age'], bins=age_bins)
sns.boxplot(data=df, x='age_group', y='Mem_Score_Before', ax=axes[1, 1])
sns.stripplot(data=df, x='age_group', y='Mem_Score_Before', color='black',␣
 ↪alpha=0.5, ax=axes[1, 1])
axes[1, 1].set_title('Mem_Score_Before vs. Age Group')

plt.tight_layout()
plt.show()
```

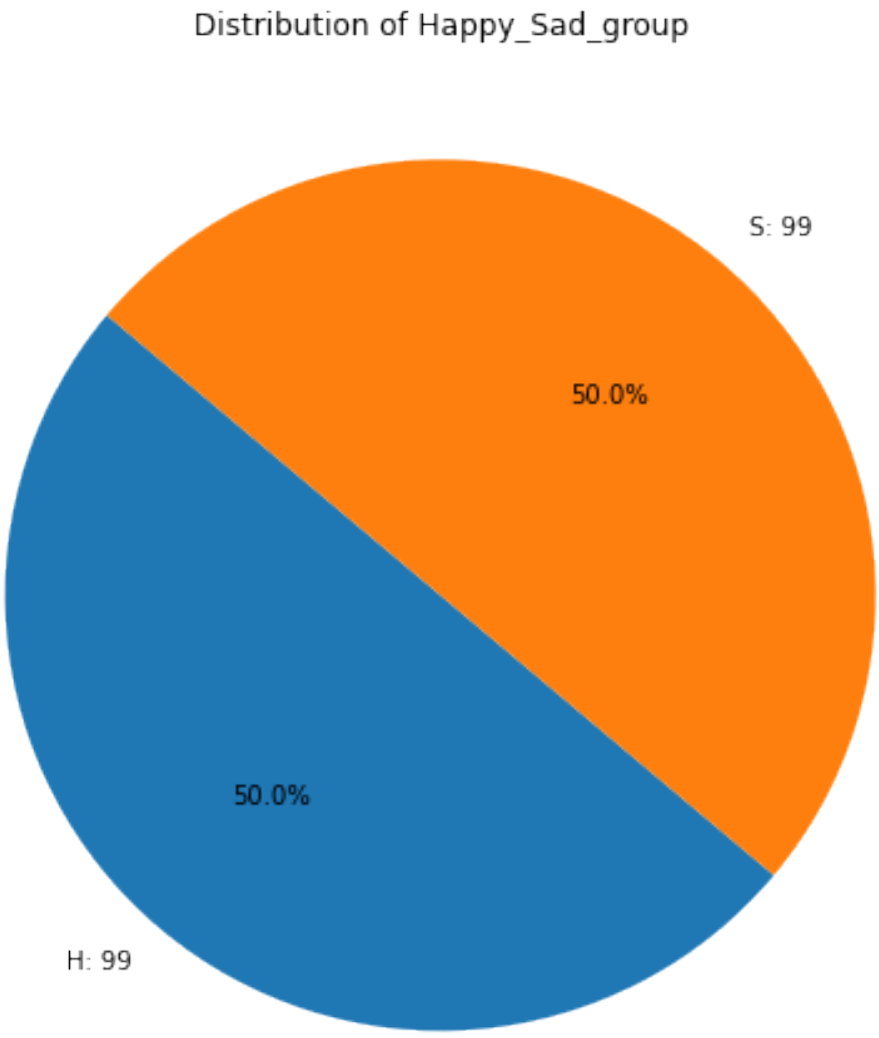


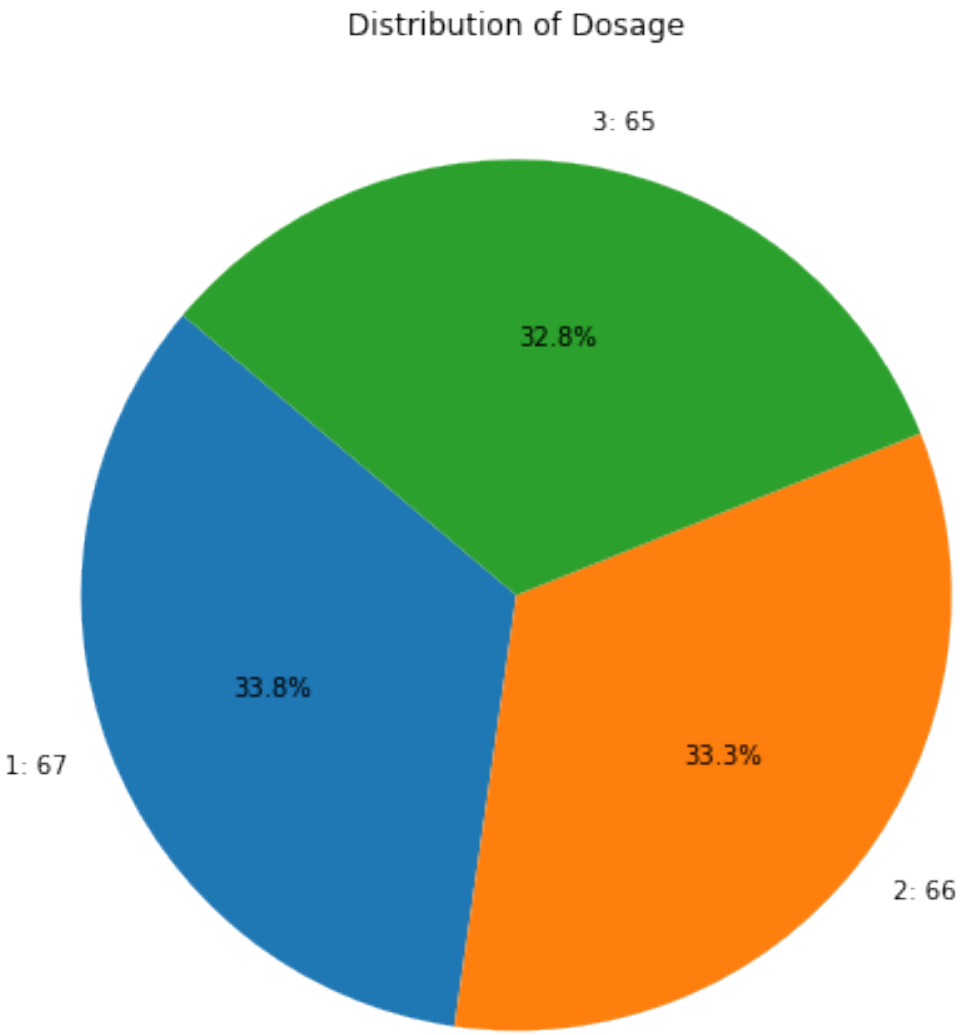
8.1.1 Number of datapoints per feature

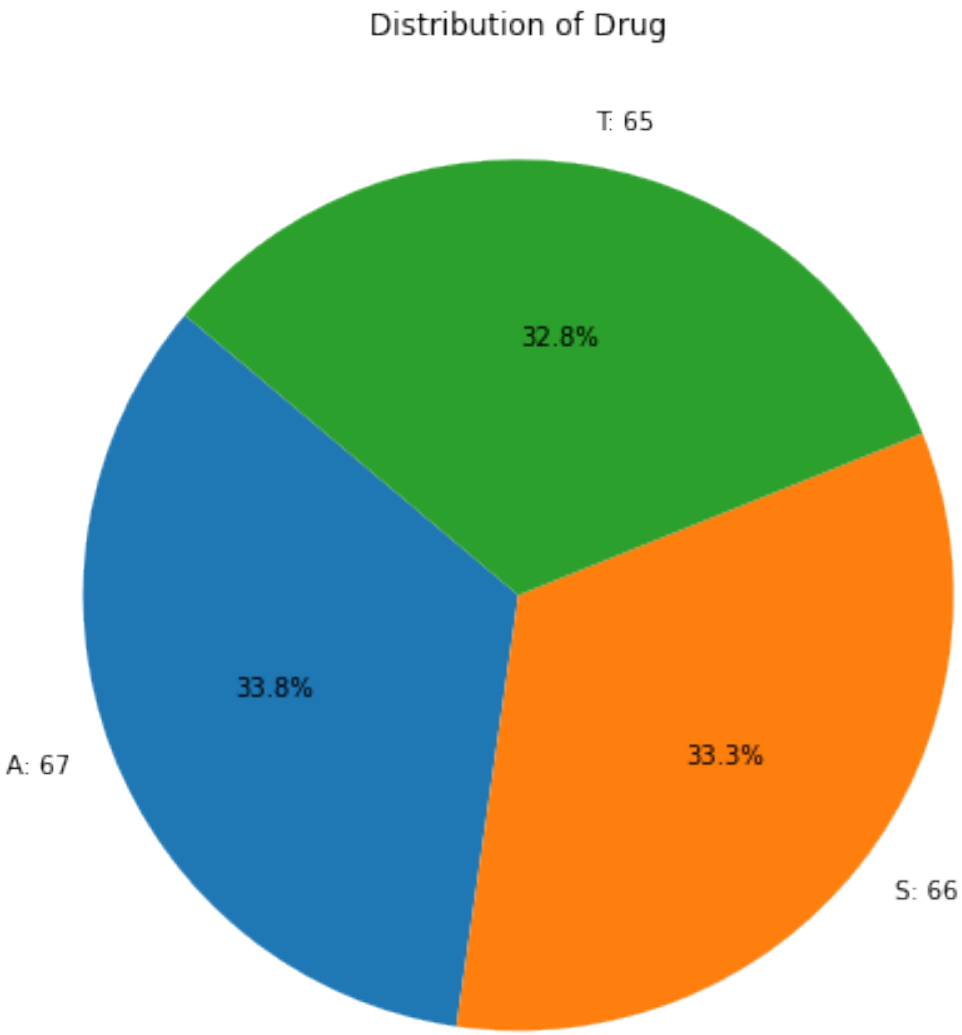
- Columns 'Happy_Sad_group', 'Drug', and 'Dosage' have approximately equal distributions of each category.
- However, the pie chart for the 'age' column shows that fewer 50-60 and 60-70 year olds enrolled in the study compared to 20-30, 30-40, and 40-50 year olds. There are extremely few 70-80 and 80-90 year olds
  - if age is a major factor in memory, will need to either correct for this or remove datapoints

[7]:
```
# Function to plot pie chart
def plot_pie_chart(series, title):
```

```
    counts = series.value_counts()
    labels = [f"{label}: {count}" for label, count in zip(counts.index, counts)]
    plt.figure(figsize=(8, 8))
    plt.pie(counts, labels=labels, autopct='%1.1f%%', startangle=140)
    plt.title(title)
    plt.show()

# Plot pie chart for 'Happy_Sad_group'
plot_pie_chart(df['Happy_Sad_group'], 'Distribution of Happy_Sad_group')

# Plot pie chart for 'Dosage'
plot_pie_chart(df['Dosage'], 'Distribution of Dosage')

# Plot pie chart for 'Drug'
plot_pie_chart(df['Drug'], 'Distribution of Drug')
```

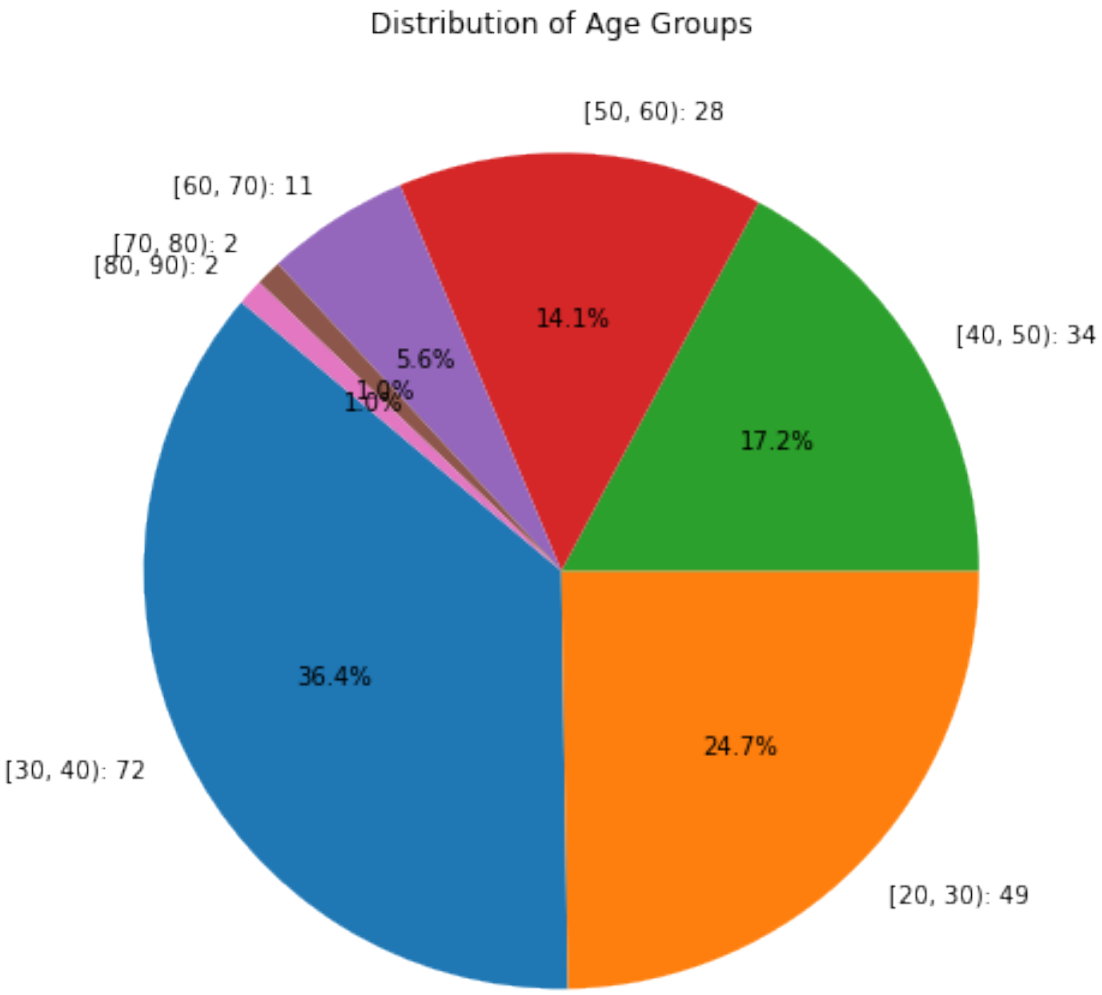

Distribution of Happy_Sad_group
S: 99
50.0%
50.0%
H: 99


Distribution of Dosage
3: 65
32.8%
33.8%
1: 67
33.3%
2: 66



Note that we have fewer older people in this study.

[8]:
```
# Group 'age' into bins and plot pie chart
age_bins = pd.cut(df['age'], bins=[20, 30, 40, 50, 60, 70, 80, 90], right=False)
plot_pie_chart(age_bins, 'Distribution of Age Groups')
```



8.1.2 Removing age group 70 and up due to extremely small group size

(2 datapoints cannot be representative of the whole population of 70-80 and 80-90 year olds)

[9]:
```
df = df[df['age'] <= 70]
df
```

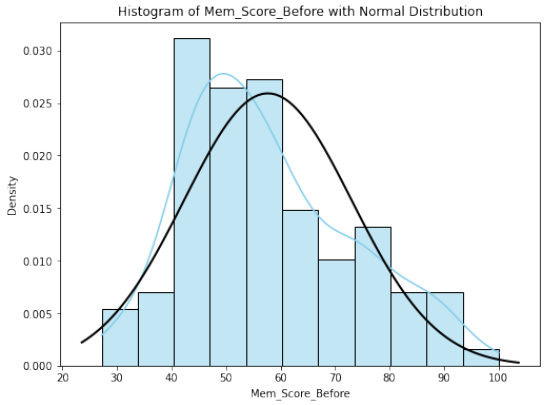
[9]:

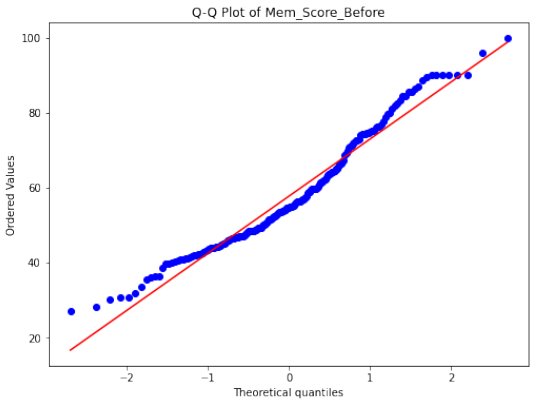
	first_name	last_name	age	Happy_Sad_group	Dosage	Drug
0	Bastian	Carrasco	25	H	1	A
1	Evan	Carrasco	52	S	1	A
2	Florencia	Carrasco	29	H	1	A
3	Holly	Carrasco	50	S	1	A
4	Justin	Carrasco	52	H	1	A

```
..          …        …  …              …     …  …
193       Jacob      Novak   52              H       3    T
194         Teo    Steiner   41              S       3    T
195   Alexander  Takahashi   54              S       3    T
196  Alexandere  Takahashi   40              H       3    T
197       Chloe  Takahashi   32              S       3    T

     Mem_Score_Before  Mem_Score_After  Diff age_group
0                63.5             61.2  -2.3  (20, 30]
1                41.6             40.7  -0.9  (50, 60]
2                59.7             55.1  -4.6  (20, 30]
3                51.7             51.2  -0.5  (40, 50]
4                47.0             47.1   0.1  (50, 60]
..                …                …   …        …
193              71.3             74.3   3.0  (50, 60]
194              72.5             70.4  -2.1  (40, 50]
195              30.8             33.1   2.3  (50, 60]
196              53.6             53.8   0.2  (30, 40]
197              43.1             42.1  -1.0  (30, 40]

[194 rows x 10 columns]
```

Checking to see if the mem_score_before is normally distributed * there is a slight left skew * The Q-Q plot shows some deviations from the red line, especially in the tails, indicating slight skewness * However, this data is close enough to normal

[10]:
```
from scipy.stats import norm, probplot
fig, axs = plt.subplots(1, 2, figsize=(18, 6))

# Histogram with fitted normal distribution curve
sns.histplot(df['Mem_Score_Before'], kde=True, ax=axs[0], color='skyblue',␣
 ↪stat='density')
xmin, xmax = axs[0].get_xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, df['Mem_Score_Before'].mean(), df['Mem_Score_Before'].std())
axs[0].plot(x, p, 'k', linewidth=2)
axs[0].set_title('Histogram of Mem_Score_Before with Normal Distribution')

# Q-Q plot
probplot(df['Mem_Score_Before'], dist="norm", plot=axs[1])
axs[1].set_title('Q-Q Plot of Mem_Score_Before')

plt.show()
```





9 EDA Analysis & Results

9.1 Data distribution

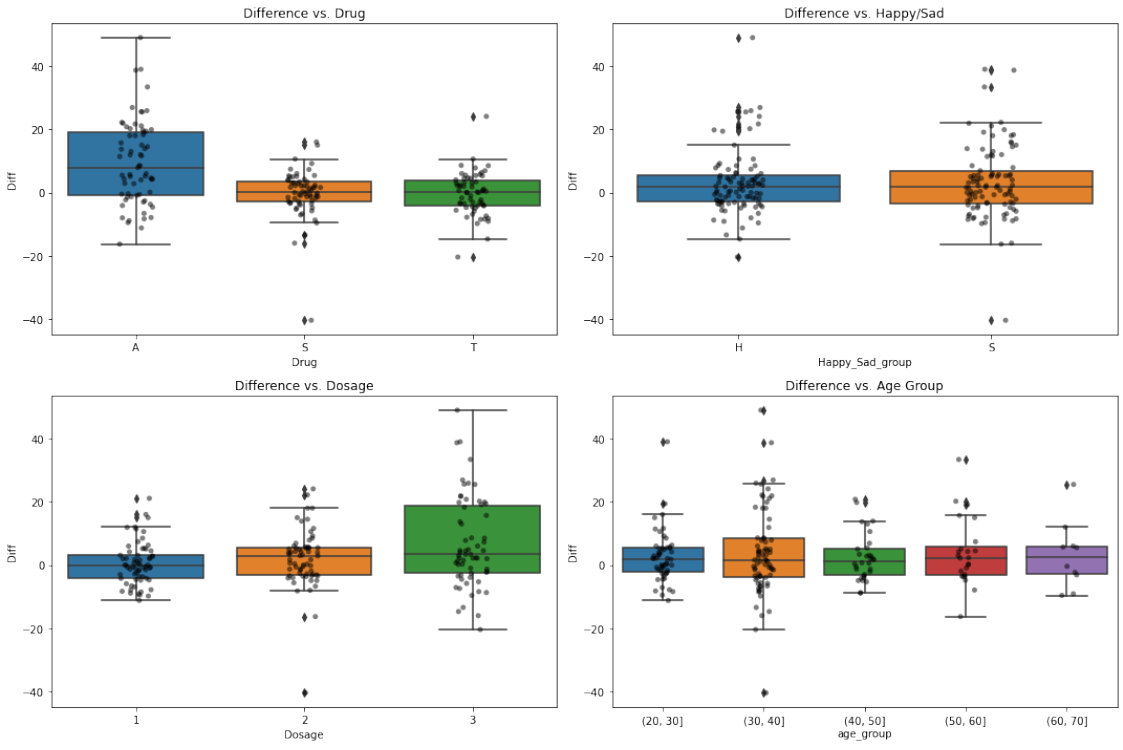
9.1.1 Diff v. drug/Happy Sad/ Dosage/ age

- if middle line of boxplot is outside the box of the other boxplot, there is a significant difference
- Note that the median line for A is higher than the Interquartile range for S and T
  - This indicates a significant difference in memory between A with S and T
  - We will need to separate the drug types in further analysis to avoid averaging out the effect of drug A

[11]:
```
# Box plots
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# diff vs. drug
sns.boxplot(data=df, x='Drug', y='Diff', ax=axes[0, 0])
sns.stripplot(data=df, x='Drug', y='Diff', color='black', alpha=0.5, ax=axes[0,␣
↪0])
axes[0, 0].set_title('Difference vs. Drug')

# diff vs. happy/sad
sns.boxplot(data=df, x='Happy_Sad_group', y='Diff', ax=axes[0, 1])
sns.stripplot(data=df, x='Happy_Sad_group', y='Diff', color='black', alpha=0.5,␣
↪ax=axes[0, 1])
axes[0, 1].set_title('Difference vs. Happy/Sad')

# diff vs. dosage
sns.boxplot(data=df, x='Dosage', y='Diff', ax=axes[1, 0])
sns.stripplot(data=df, x='Dosage', y='Diff', color='black', alpha=0.5,␣
↪ax=axes[1, 0])
axes[1, 0].set_title('Difference vs. Dosage')
```

```
# diff vs. age (grouped by ranges)
#age_bins = [20, 30, 40, 50, 60]
age_bins = [20, 30, 40, 50, 60, 70]
df['age_group'] = pd.cut(df['age'], bins=age_bins)
sns.boxplot(data=df, x='age_group', y='Diff', ax=axes[1, 1])
sns.stripplot(data=df, x='age_group', y='Diff', color='black', alpha=0.5,␣
 ↪ax=axes[1, 1])
axes[1, 1].set_title('Difference vs. Age Group')

plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_272/578076523.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['age_group'] = pd.cut(df['age'], bins=age_bins)
```



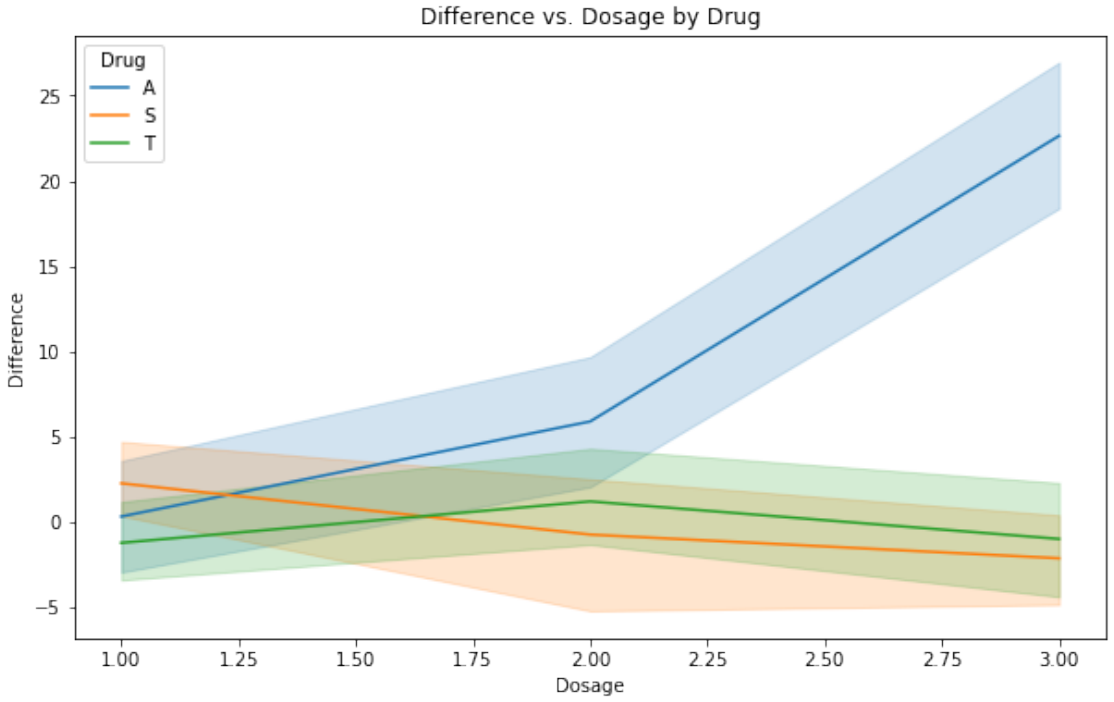
9.1.2 Dosage v difference for diff drugs

A postulate for determining wheter a relationship is causal is the Dose response relationship. It states that the greater the exposure, the greater the likelihood of the particular outcome (except

with tresholds) * Hence, the greater the dose of anxiety medication, the greater the difference in memory * Drug 'A' shows the dose-response effect, while Drug 'T' does not

[12]:
```
# Convert drug to categorical type- maybe change this
df['Drug'] = df['Drug'].astype('category')

# Line plot: dosage vs. difference with separate lines for each drug
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x='Dosage', y='Diff', hue='Drug')
plt.title('Difference vs. Dosage by Drug')
plt.xlabel('Dosage')
plt.ylabel('Difference')
plt.legend(title='Drug')
plt.show()
```

```
/tmp/ipykernel_272/3790329003.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Drug'] = df['Drug'].astype('category')
```



9.1.3 t-tests (for graph above):

** What is a t-test and when is it used?** * A t-test is a statistical test used to determine if there is a significant difference between the means of two groups * in this case- either a given dosage and comparing drugs, or comparing dosages with a given drug * we will be using indepdendent samples t tests: * Used when you have two different groups and you want to compare their means.

What is this code doing * testing to see if there are significant differences between different number of doses for the same drug * also testing to see if the 3 drugs significantly differ per dose * Drug 'A' significantly differs from dose to dose, and significantly differs from Drugs 'T' and 'S'

confusing aspects of the results

- there is a significant difference between 'S' and 'T'- at dose 1 only
- Significant difference between doses 1 and 3 of Drug 'S;

[13]:
```
from scipy.stats import ttest_ind
import itertools

# Function to perform t-test between all three drugs at each dosage level
def perform_ttest_between_drugs(dosage):
    drugs = df['Drug'].unique()
    significant_results = []
    non_significant_results = []

    for drug1, drug2 in itertools.combinations(drugs, 2):
        group1 = df[(df['Dosage'] == dosage) & (df['Drug'] == drug1)]['Diff']
        group2 = df[(df['Dosage'] == dosage) & (df['Drug'] == drug2)]['Diff']
        t_stat, p_value = ttest_ind(group1, group2, equal_var=False)

        result = (f"Dosage: {dosage}, Drug Comparison: {drug1} vs {drug2}\n"
                  f"T-statistic: {t_stat}, P-value: {p_value}")
        if p_value < 0.05:
            significant_results.append(result + f"\nSignificant difference␣
 ↪between {drug1} and {drug2} at dosage {dosage} (p < 0.05)")
        else:
            non_significant_results.append(result + f"\nNo significant␣
 ↪difference between {drug1} and {drug2} at dosage {dosage} (p >= 0.05)")

    return significant_results, non_significant_results

# Function to perform t-test between dosages for the same drug
def perform_ttest_between_dosages(drug):
    dosages = df['Dosage'].unique()
    significant_results = []
    non_significant_results = []

    for dosage1, dosage2 in itertools.combinations(dosages, 2):
        group1 = df[(df['Dosage'] == dosage1) & (df['Drug'] == drug)]['Diff']
```

```
        group2 = df[(df['Dosage'] == dosage2) & (df['Drug'] == drug)]['Diff']
        t_stat, p_value = ttest_ind(group1, group2, equal_var=False)

        result = (f"Drug: {drug}, Dosage Comparison: {dosage1} vs {dosage2}\n"
                  f"T-statistic: {t_stat}, P-value: {p_value}")
        if p_value < 0.05:
            significant_results.append(result + f"\nSignificant difference␣
↪between dosage {dosage1} and dosage {dosage2} for drug {drug} (p < 0.05)")
        else:
            non_significant_results.append(result + f"\nNo significant␣
↪difference between dosage {dosage1} and dosage {dosage2} for drug {drug} (p␣
↪>= 0.05)")

    return significant_results, non_significant_results

# Accumulate and print the results
all_significant_results = []
all_non_significant_results = []

# Performing t-tests between drugs at each dosage level
for dosage in df['Dosage'].unique():
    significant, non_significant = perform_ttest_between_drugs(dosage)
    all_significant_results.extend(significant)
    all_non_significant_results.extend(non_significant)

# Performing t-tests between dosages for each drug
for drug in df['Drug'].unique():
    significant, non_significant = perform_ttest_between_dosages(drug)
    all_significant_results.extend(significant)
    all_non_significant_results.extend(non_significant)

# Print significant results first
for res in all_significant_results:
    print(res)
    print("-" * 50)

# Add space between significant and non-significant results
if all_significant_results and all_non_significant_results:
    print("\n" + "=" * 50 + "\n")

# Print non-significant results
for res in all_non_significant_results:
    print(res)
    print("-" * 50)
```

```
Dosage: 1, Drug Comparison: S vs T
T-statistic: 2.105639380389042, P-value: 0.04140547853607697
```

```
Significant difference between S and T at dosage 1 (p < 0.05)
--------------------------------------------------
Dosage: 2, Drug Comparison: A vs S
T-statistic: 2.2992181647407577, P-value: 0.026562407052004872
Significant difference between A and S at dosage 2 (p < 0.05)
--------------------------------------------------
Dosage: 3, Drug Comparison: A vs S
T-statistic: 9.655574459722134, P-value: 3.04259266859377e-11
Significant difference between A and S at dosage 3 (p < 0.05)
--------------------------------------------------
Dosage: 3, Drug Comparison: A vs T
T-statistic: 8.350579304583334, P-value: 3.863433162556694e-10
Significant difference between A and T at dosage 3 (p < 0.05)
--------------------------------------------------
Drug: A, Dosage Comparison: 1 vs 2
T-statistic: -2.133439731134499, P-value: 0.03876946210222351
Significant difference between dosage 1 and dosage 2 for drug A (p < 0.05)
--------------------------------------------------
Drug: A, Dosage Comparison: 1 vs 3
T-statistic: -7.947067405097625, P-value: 9.172652436740602e-10
Significant difference between dosage 1 and dosage 3 for drug A (p < 0.05)
--------------------------------------------------
Drug: A, Dosage Comparison: 2 vs 3
T-statistic: -5.657003001911807, P-value: 1.2913738906752695e-06
Significant difference between dosage 2 and dosage 3 for drug A (p < 0.05)
--------------------------------------------------
Drug: S, Dosage Comparison: 1 vs 3
T-statistic: 2.526057287351899, P-value: 0.01551706906065347
Significant difference between dosage 1 and dosage 3 for drug S (p < 0.05)
--------------------------------------------------

==================================================

Dosage: 1, Drug Comparison: A vs S
T-statistic: -0.9349638837615545, P-value: 0.3557131596682275
No significant difference between A and S at dosage 1 (p >= 0.05)
--------------------------------------------------
Dosage: 1, Drug Comparison: A vs T
T-statistic: 0.7392789105974101, P-value: 0.46421766479149384
No significant difference between A and T at dosage 1 (p >= 0.05)
--------------------------------------------------
Dosage: 2, Drug Comparison: A vs T
T-statistic: 1.8978430157435537, P-value: 0.0651736822366722
No significant difference between A and T at dosage 2 (p >= 0.05)
--------------------------------------------------
Dosage: 2, Drug Comparison: S vs T
T-statistic: -0.7497444223550023, P-value: 0.4580870911717686
No significant difference between S and T at dosage 2 (p >= 0.05)
```

```
--------------------------------------------------
Dosage: 3, Drug Comparison: S vs T
T-statistic: -0.5196552468940717, P-value: 0.6066592905065584
No significant difference between S and T at dosage 3 (p >= 0.05)
--------------------------------------------------
Drug: S, Dosage Comparison: 1 vs 2
T-statistic: 1.244533298459185, P-value: 0.22218696328765902
No significant difference between dosage 1 and dosage 2 for drug S (p >= 0.05)
--------------------------------------------------
Drug: S, Dosage Comparison: 2 vs 3
T-statistic: 0.5611344582845272, P-value: 0.578306775703592
No significant difference between dosage 2 and dosage 3 for drug S (p >= 0.05)
--------------------------------------------------
Drug: T, Dosage Comparison: 1 vs 2
T-statistic: -1.2735400699274106, P-value: 0.21048721624537686
No significant difference between dosage 1 and dosage 2 for drug T (p >= 0.05)
--------------------------------------------------
Drug: T, Dosage Comparison: 1 vs 3
T-statistic: -0.10871029266008979, P-value: 0.9141067495693362
No significant difference between dosage 1 and dosage 3 for drug T (p >= 0.05)
--------------------------------------------------
Drug: T, Dosage Comparison: 2 vs 3
T-statistic: 0.9510606336601836, P-value: 0.3478494937097464
No significant difference between dosage 2 and dosage 3 for drug T (p >= 0.05)
--------------------------------------------------
```

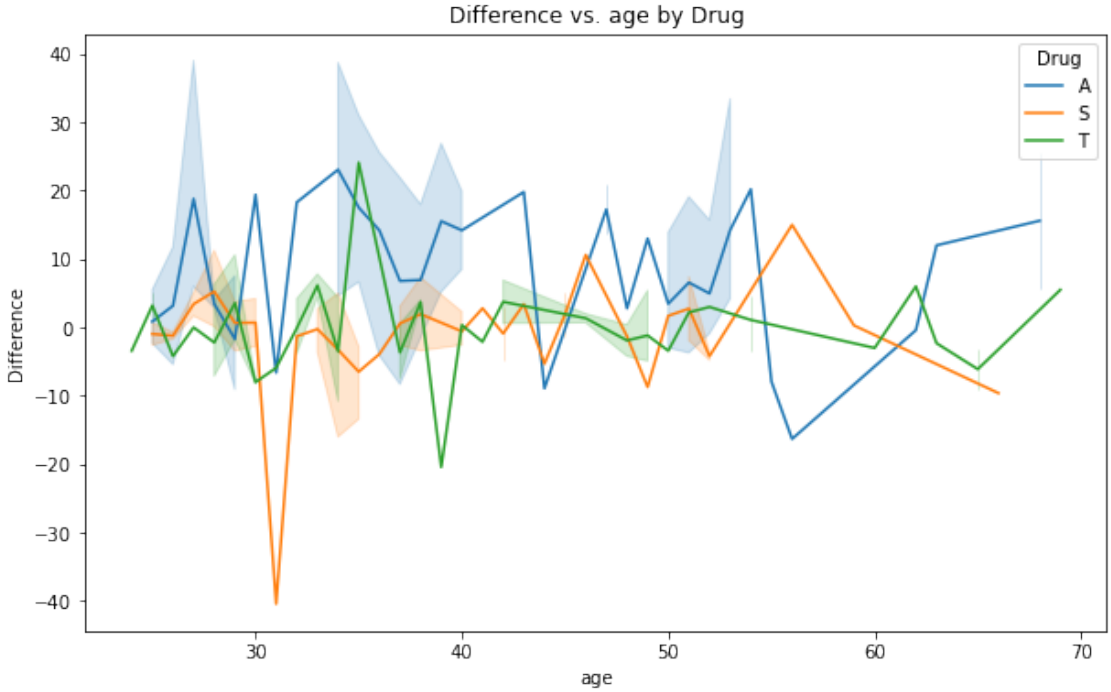
9.1.4 age v difference for diff drugs

- no obvious trend
- obviously doesn't need t-test
- Age doesn't significantly impact memory score- hence the lack of older participants doesn't have to be corrected

[14]:
```
# Convert drug to categorical type- maybe change this
df['Drug'] = df['Drug'].astype('category')

# Line plot: dosage vs. difference with separate lines for each drug
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x='age', y='Diff', hue='Drug')
plt.title('Difference vs. age by Drug')
plt.xlabel('age')
plt.ylabel('Difference')
plt.legend(title='Drug')
plt.show()
```

```
/tmp/ipykernel_272/417658996.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

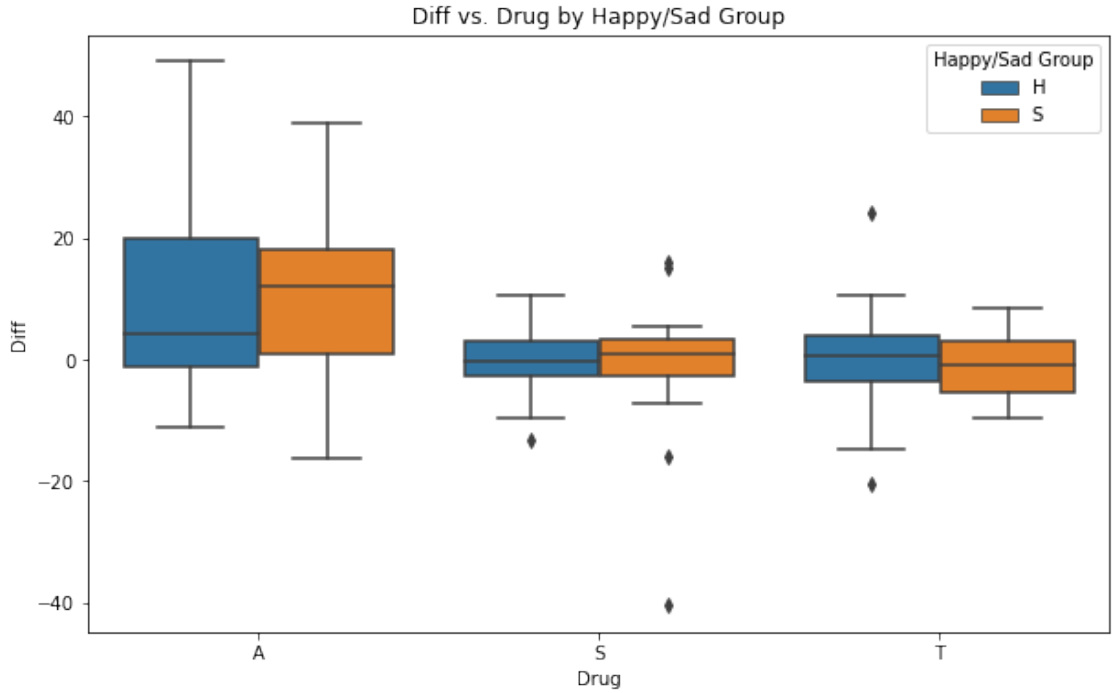
```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Drug'] = df['Drug'].astype('category')
```



9.1.5 Diff vs. Drug by Happy/Sad Group'

Vertical Lines are error bars * The reason the difference is 0 for S is because it is not supposed to have an effect on memory (control) * Drug 'T has a similar effect as drug 'S' * Drug 'A' has a ositive mean differences in memory scores

[15]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df is your DataFrame and already loaded
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Drug', y='Diff', hue='Happy_Sad_group')
plt.title('Diff vs. Drug by Happy/Sad Group')
plt.xlabel('Drug')
plt.ylabel('Diff')
plt.legend(title='Happy/Sad Group')
plt.show()
```



[16]:
```
#filtered_df = df[(df['Drug'] == 'S') & (df['Happy_Sad_group'] == 'S')]
#filtered_df
```

9.1.6 checking to see if there is any significant difference

- no significant difference

[17]:
```
from scipy.stats import ttest_ind
# Function to perform t-test and print results
def perform_ttest(drug):
    happy_group = df[(df['Drug'] == drug) & (df['Happy_Sad_group'] ==␣
 ↪'H')]['Diff']
    sad_group = df[(df['Drug'] == drug) & (df['Happy_Sad_group'] ==␣
 ↪'S')]['Diff']
    t_stat, p_value = ttest_ind(happy_group, sad_group, equal_var=False)
    print(f"Drug: {drug}")
    print(f"T-statistic: {t_stat}, P-value: {p_value}")
    if p_value < 0.05:
        print(f"Significant difference between Happy and Sad groups for drug␣
 ↪{drug} (p < 0.05)")
    else:
        print(f"No significant difference between Happy and Sad groups for drug␣
 ↪{drug} (p >= 0.05)")
    print("-" * 50)
```

```
# Performing t-tests for each drug
for drug in df['Drug'].unique():
    perform_ttest(drug)
```

```
Drug: A
T-statistic: -0.6704281950534117, P-value: 0.5049620787469711
No significant difference between Happy and Sad groups for drug A (p >= 0.05)
--------------------------------------------------
Drug: S
T-statistic: 0.29462725763459835, P-value: 0.7695314947014337
No significant difference between Happy and Sad groups for drug S (p >= 0.05)
--------------------------------------------------
Drug: T
T-statistic: 1.009055998661902, P-value: 0.3175703364613326
No significant difference between Happy and Sad groups for drug T (p >= 0.05)
--------------------------------------------------
```

[18]:
```
drug_a = df[df['Drug'] == 'A']
```

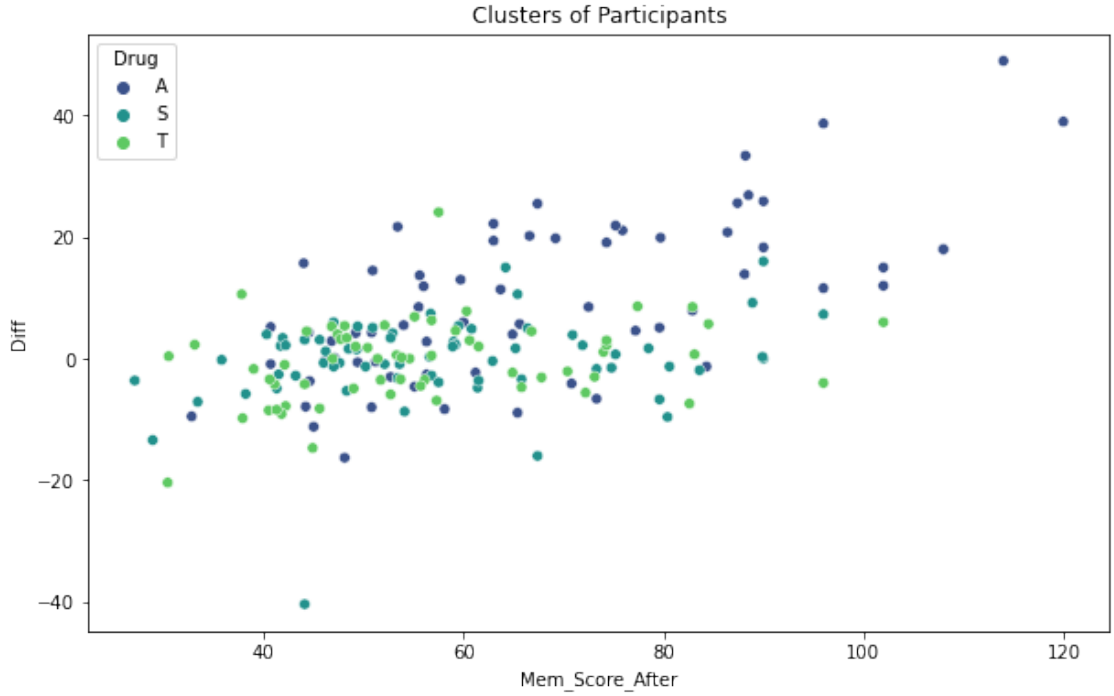
10 Machine Learning program to predict Mem_after_Score given features

[19]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

[20]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Preprocess the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[['age', 'Dosage', 'Mem_Score_Before',␣
 ↪'Mem_Score_After', 'Diff']])

# KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Visualization of clusters
plt.figure(figsize=(10, 6))
```

```
sns.scatterplot(x='Mem_Score_After', y='Diff', hue='Drug', data=df,␣
 ↪palette='viridis')
plt.title('Clusters of Participants')
plt.show()
```

```
/tmp/ipykernel_272/3327649468.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Cluster'] = kmeans.fit_predict(X_scaled)
```



[21]:
```
df[df['Cluster'] == 2]['Drug'].value_counts()
```

[21]:
```
A    27
S    25
T    24
Name: Drug, dtype: int64
```

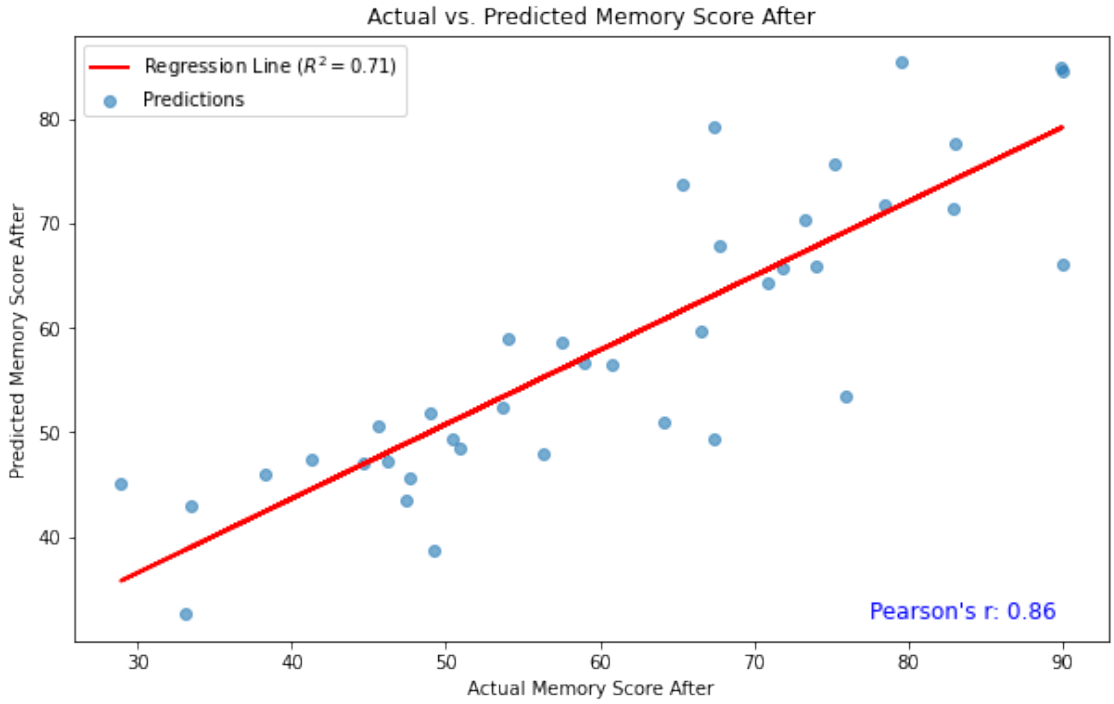
10.1 model 1#note- removed age and Happy-sad-group

model 2: removed happy-sad-group and age

[22]:
```
# Prepare the data for regression
X = df[[ 'Dosage', 'Drug', 'Mem_Score_Before', 'Happy_Sad_group', 'age']] # Add␣
 ↪age and Happy_Sad_group
y = df['Mem_Score_After']

# Convert categorical variables to numeric
X = pd.get_dummies(X, columns=['Drug', 'Happy_Sad_group'], drop_first=True)␣
 ↪#Happy_Sad_group

# Add polynomial features
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

# Standardize the features
scaler = StandardScaler()
X_poly_scaled = scaler.fit_transform(X_poly)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_poly_scaled, y,␣
 ↪test_size=0.2, random_state=42)

# Linear Regression model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Predictions
y_pred = lin_reg.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")

# Plot predictions vs actual
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, label='Predictions', alpha=0.6)
plt.xlabel('Actual Memory Score After')
plt.ylabel('Predicted Memory Score After')
plt.title('Actual vs. Predicted Memory Score After')

# Add regression line
slope, intercept = np.polyfit(y_test, y_pred, 1)
plt.plot(y_test, slope * y_test + intercept, color='red', linewidth=2,␣
 ↪label=f'Regression Line ($R^2 = {r2:.2f}$)')
```

```
# Add Pearson correlation coefficient
from scipy.stats import pearsonr
pearson_corr, _ = pearsonr(y_test, y_pred)
plt.text(0.95, 0.05, f"Pearson's r: {pearson_corr:.2f}", fontsize=12,␣
 ↪color='blue', ha='right', va='center', transform=plt.gca().transAxes)
#plt.text(min(y_test), max(y_pred), f"Pearson's r: {pearson_corr:.2f}",␣
 ↪fontsize=12, color='blue')

plt.legend()
plt.show()
```

```
Mean Squared Error: 77.78616120325266
R^2 Score: 0.7108320014052898
```

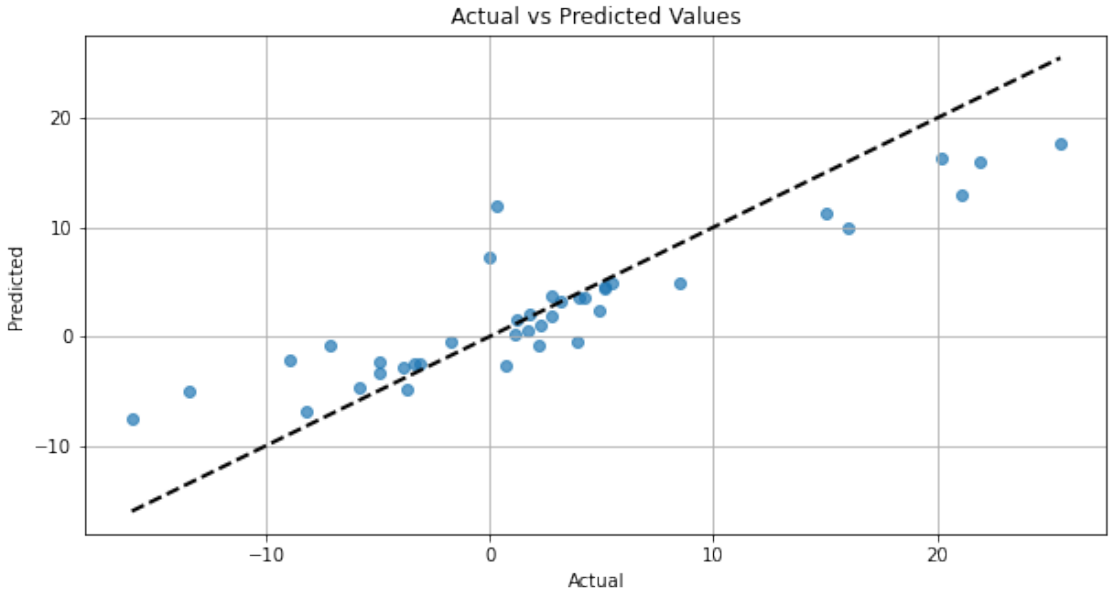


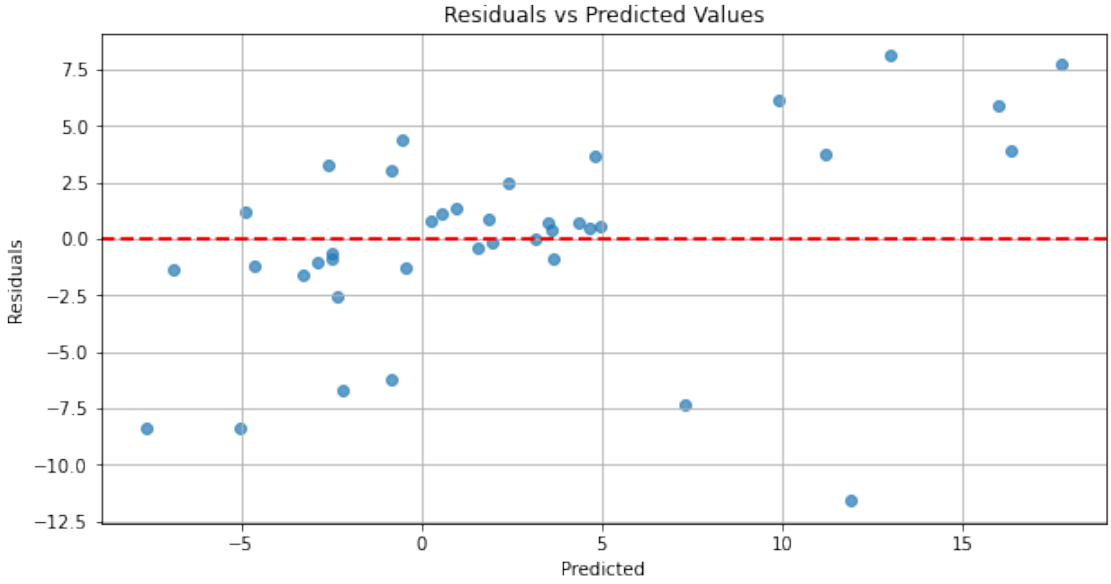
[24]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt
```

```
#Separate the target variable and features
X = df.drop(columns=['Diff', 'first_name', 'last_name']) #irrelevant features␣
 ↪for model
y = df['Diff'] #target var

#Preprocess categorical features and numerical features
categorical_features = ['Happy_Sad_group', 'Drug']
numerical_features = ['age', 'Dosage', 'Mem_Score_Before', 'Mem_Score_After']

#Define the preprocessing steps for numerical and categorical data
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
]) #imputer- replace missing value with means, scsaler- scale all numerical␣
 ↪vals to 0-1 (equal impact in model)

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
]) #onehot- categorical columns into numerical vars

#Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

#Define the model pipeline (Pipeline- combine preprocessing steps and model-␣
 ↪less error prone)
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=100, random_state=42))
])

#Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

#Train the model
model.fit(X_train, y_train)

#Predict on the test set
y_pred = model.predict(X_test)

#Evaluate the model
```

```
r2 = r2_score(y_test, y_pred)
print(f"R^2 Score: {r2}")

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)

print(f'MSE: {mse}, MAE: {mae}, RMSE: {rmse}')

#Scatter plot of predicted vs actual values
plt.figure(figsize=(10, 5))
plt.scatter(y_test, y_pred, alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',␣
 ↪lw=2)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted Values')
plt.grid(True)
plt.show()

#Residual plot
residuals = y_test - y_pred
plt.figure(figsize=(10, 5))
plt.scatter(y_pred, residuals, alpha=0.7)
plt.axhline(y=0, color='r', linestyle='--', lw=2)
plt.xlabel('Predicted')
plt.ylabel('Residuals')
plt.title('Residuals vs Predicted Values')
plt.grid(True)
plt.show()
```

```
MSE: 18.586809435897433, MAE: 3.1082051282051277, RMSE: 4.311242214941934
```






```
R^2 Score: 0.7781618310317374
```

11 Conclusion & Discussion

- Discussion of your results and how they address your experimental question(s).
- Discussed limitations of your analyses.
- You can also discuss future directions you'd like to pursue.

[ ]: