```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import shapiro, mannwhitneyu, wilcoxon, ttest_ind, ttest_rel
from math import sqrt
```

Reading the data from the xlsx files

```
choice = pd.read_excel('choice.xlsx', sheet_name=None, header = None)
win = pd.read_excel('win.xlsx', sheet_name=None, header = None)
loss = pd.read_excel('loss.xlsx', sheet_name=None, header = None)
```

```
len(choice['group2'])
```

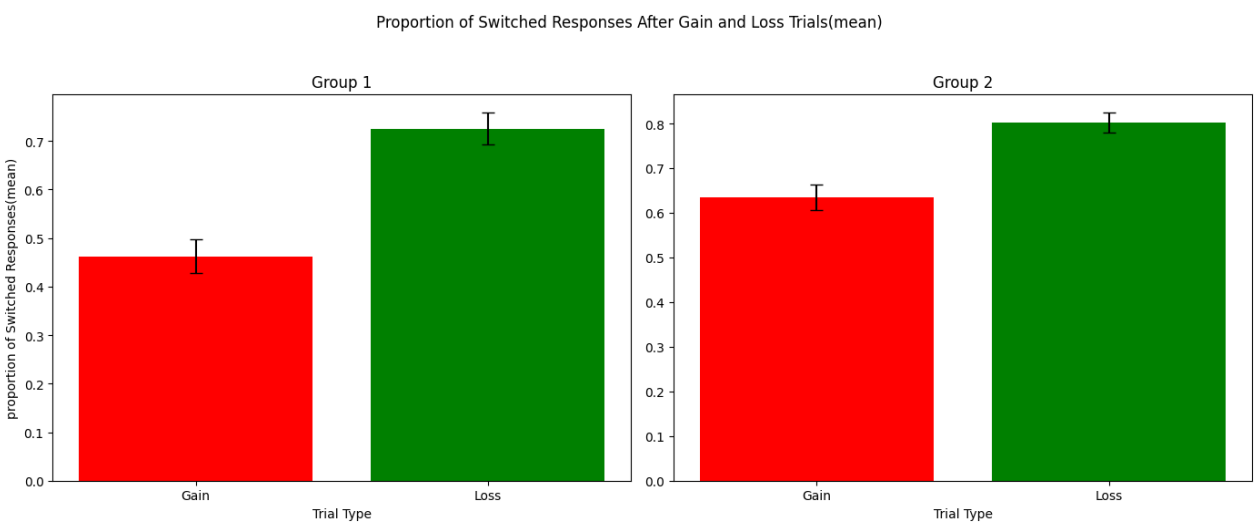
```
63
```

Question 2 (A)

```
Start coding or generate with AI.
```

```
# Function to calculate proportion of switches after a gain/loss trial
def calculate_switch_proportions(choice, win, loss):
    participants = choice.shape[0]
    trials = choice.shape[1]
    switch_after_gain = np.zeros(participants)
    switch_after_loss = np.zeros(participants)

    for i in range(participants):
        switches_gain = 0
        switches_loss = 0
        gain_trials = 0
        loss_trials = 0

        for t in range(trials - 1):
            if win.iloc[i, t] > abs(loss.iloc[i, t]):  # Gain trial (since loss
                gain_trials += 1
                if choice.iloc[i, t] != choice.iloc[i, t + 1]:
                    switches_gain += 1
            else:  # Loss trial
                loss_trials += 1
                if choice.iloc[i, t] != choice.iloc[i, t + 1]:
                    switches_loss += 1

        switch_after_gain[i] = switches_gain / (gain_trials)
        switch_after_loss[i] = switches_loss / (loss_trials)
```

```
        switch_after_loss[i] = switches_loss / (loss_trials)

    return switch_after_gain, switch_after_loss


switch_gain_group1, switch_loss_group1 = calculate_switch_proportions(choice['gr
switch_gain_group2, switch_loss_group2 = calculate_switch_proportions(choice['gr


mean_gain_group1 = np.mean(switch_gain_group1)
mean_loss_group1 = np.mean(switch_loss_group1)
mean_gain_group2 = np.mean(switch_gain_group2)
mean_loss_group2 = np.mean(switch_loss_group2)


sem_gain_group1 = stats.sem(switch_gain_group1)
sem_loss_group1 = stats.sem(switch_loss_group1)
sem_gain_group2 = stats.sem(switch_gain_group2)
sem_loss_group2 = stats.sem(switch_loss_group2)


# Plotting the bar plots with error bars
fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=False)
# Group 1 plot

axes[0].bar(['Gain', 'Loss'], [mean_gain_group1, mean_loss_group1], yerr=[sem_ga
axes[0].set_title('Group 1')
axes[0].set_xlabel('Trial Type')
axes[0].set_ylabel('proportion of Switched Responses(mean)')


# Group 2 plot
axes[1].bar(['Gain', 'Loss'], [mean_gain_group2, mean_loss_group2], yerr=[sem_ga
axes[1].set_title('Group 2')
axes[1].set_xlabel('Trial Type')


plt.suptitle('Proportion of Switched Responses After Gain and Loss Trials(mean)'
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



Checking additional stats and see if there is a need for correction

```
# Diagnostic check for variability and outliers
def diagnostic_check(data, group_name):
    print(f"Diagnostic check for {group_name}:")
    print(f"Mean: {np.mean(data):.3f}, Std Dev: {np.std(data):.3f}, Min: {np.mi
    print(f"Number of unique values: {len(np.unique(data))}")
    if len(np.unique(data)) < 5:
        print("Problem somewhere, variablity low")
    else:
        print("No problem detected")



# Run diagnostic checks
for group_data, group_name in zip([switch_gain_group1, switch_loss_group1, swit
                                   ["Group 1 Gain", "Group 1 Loss", "Group 2 Gai
    diagnostic_check(group_data, group_name)
```

```
Diagnostic check for Group 1 Gain:
Mean: 0.462, Std Dev: 0.275, Min: 0.024, Max: 1.000
Number of unique values: 57
No problem detected
Diagnostic check for Group 1 Loss:
Mean: 0.726, Std Dev: 0.252, Min: 0.067, Max: 1.000
Number of unique values: 42
No problem detected
Diagnostic check for Group 2 Gain:
Mean: 0.635, Std Dev: 0.227, Min: 0.024, Max: 1.000
Number of unique values: 57
No problem detected
Diagnostic check for Group 2 Loss:
Mean: 0.802, Std Dev: 0.172, Min: 0.200, Max: 1.000
Number of unique values: 42
No problem detected
```

Start coding or generate with AI

Start coding or generate with AI.

Running statistical tests and finding out the test metrics and p values

```
# Statistical tests
def perform_stat_tests(switch_gain_group1, switch_loss_group1, switch_gain_grou
    # Normality check using Shapiro-Wilk test
    p_gain_group1 = shapiro(switch_gain_group1).pvalue
    p_loss_group1 = shapiro(switch_loss_group1).pvalue
    p_gain_group2 = shapiro(switch_gain_group2).pvalue
    p_loss_group2 = shapiro(switch_loss_group2).pvalue

    print("Shapiro-Wilk test for normality:")
    print(f"Group 1 Gain trials: p-value = {p_gain_group1}")
    print(f"Group 1 Loss trials: p-value = {p_loss_group1}")
    print(f"Group 2 Gain trials: p-value = {p_gain_group2}")
    print(f"Group 2 Loss trials: p-value = {p_loss_group2}\n")

    # Determine whether to use parametric or non-parametric tests based on norm
    def use_parametric_test(p_values):

        return all(p > 0.05 for p in p_values)

    # Between groups comparison
    if use_parametric_test([p_gain_group1, p_gain_group2]):
        # Parametric test (t-test) for gain trials between groups
        t_gain, p_gain = ttest_ind(switch_gain_group1, switch_gain_group2)
        print(f"Gain trials between groups (t-test): t-statistic = {t_gain}, p-
    else:
        # Non-parametric test (Mann-Whitney U) for gain trials between groups
        u_gain, p_gain = mannwhitneyu(switch_gain_group1, switch_gain_group2, a
        print(f"Gain trials between groups (Mann-Whitney U): U-statistic = {u_g

    if use_parametric_test([p_loss_group1, p_loss_group2]):
        # Parametric test (t-test) for loss trials between groups
        t_loss, p_loss = ttest_ind(switch_loss_group1, switch_loss_group2)
        print(f"Loss trials between groups (t-test): t-statistic = {t_loss}, p-
    else:
        # Non-parametric test (Mann-Whitney U) for loss trials between groups
        u_loss, p_loss = mannwhitneyu(switch_loss_group1, switch_loss_group2, a
        print(f"Loss trials between groups (Mann-Whitney U): U-statistic = {u_l

    # Within group comparisons (Gain vs Loss)
    if use_parametric_test([p_gain_group1, p_loss_group1]):
        # Parametric test (paired t-test) for Group 1
        t_group1, p_group1 = ttest_rel(switch_gain_group1, switch_loss_group1)
        print(f"Group 1 (Gain vs Loss, t-test): t-statistic = {t_group1}, p-val
    else:
        # Non-parametric test (Wilcoxon) for Group 1
        w_group1, p_group1 = wilcoxon(switch_gain_group1, switch_loss_group1)
        print(f"Group 1 (Gain vs Loss, Wilcoxon): W-statistic = {w_group1}, p-v

    if use_parametric_test([p_gain_group2, p_loss_group2]):
```

```
    if use_parametric_test(p_gain_group2, p_loss_group2):
        # Parametric test (paired t-test) for Group 2
        t_group2, p_group2 = ttest_rel(switch_gain_group2, switch_loss_group2)
        print(f"Group 2 (Gain vs Loss, t-test): t-statistic = {t_group2}, p-val
    else:
        # Non-parametric test (Wilcoxon) for Group 2
        w_group2, p_group2 = wilcoxon(switch_gain_group2, switch_loss_group2)
        print(f"Group 2 (Gain vs Loss, Wilcoxon): W-statistic = {w_group2}, p-v

# Perform statistical tests
perform_stat_tests(switch_gain_group1, switch_loss_group1, switch_gain_group2,
```

```
Shapiro-Wilk test for normality:
Group 1 Gain trials: p-value = 0.04177296531284849
Group 1 Loss trials: p-value = 7.813588444336579e-05
Group 2 Gain trials: p-value = 0.002720013485003477
Group 2 Loss trials: p-value = 2.8209183037365264e-05

Gain trials between groups (Mann-Whitney U): U-statistic = 1210.0, p-value
Loss trials between groups (Mann-Whitney U): U-statistic = 1741.0, p-value

Group 1 (Gain vs Loss, Wilcoxon): W-statistic = 62.0, p-value = 1.439567616
Group 2 (Gain vs Loss, Wilcoxon): W-statistic = 130.0, p-value = 2.94039693
```

Start coding or generate with AI.

```
# Function to calculate deck choice proportions before and after loss trials
def calculate_deck_proportions(choice, loss,win):
    participants = choice.shape[0]
    trials = choice.shape[1]
    deck_choices_before_loss = {1: 0, 2: 0, 3: 0, 4: 0}
    deck_choices_after_loss = {1: 0, 2: 0, 3: 0, 4: 0}

    for i in range(participants):
        for t in range(1, trials):
            if win.iloc[i,t-1] + loss.iloc[i, t - 1] < 0 and choice.iloc[i,t] !
                deck_choices_after_loss[choice.iloc[i, t]] += 1
                deck_choices_before_loss[choice.iloc[i, t - 1]] += 1

                # print("participant: ",i+1)
                # print("trial :", t)
                # print("choice after loss : ", choice.iloc[i,t])
                # print("choice before loss : ", choice.iloc[i,t-1])


    total_before = sum(deck_choices_before_loss.values())
    total_after = sum(deck_choices_after_loss.values())

    proportions_before = {deck: count / total_before for deck, count in deck_ch
    proportions_after = {deck: count / total_after for deck, count in deck_choi

    return proportions_before, proportions_after
```

```
# Calculate deck choice proportions for both groups

proportions_before_group1, proportions_after_group1 = calculate_deck_proportior
proportions_before_group2, proportions_after_group2 = calculate_deck_proportior



# Print the results in a formatted way
def print_deck_analysis(proportions_before, proportions_after, group_name):
    print(f"{group_name} deck rankings (before loss):")
    for rank, (deck, proportion) in enumerate(sorted(proportions_before.items()
        print(f"Rank {rank}: Deck {deck} (proportion: {round(proportion,3)})")
    print()
    print(f"{group_name} deck rankings (after loss):")
    for rank, (deck, proportion) in enumerate(sorted(proportions_after.items(),
        print(f"Rank {rank}: Deck {deck} (proportion: {round(proportion,3)})")
    print()

# Print deck choice analysis for both groups
print("Analysis of deck choices BEFORE and AFTER loss trials:")
print_deck_analysis(proportions_before_group1, proportions_after_group1, "Group
print_deck_analysis(proportions_before_group2, proportions_after_group2, "Group
```

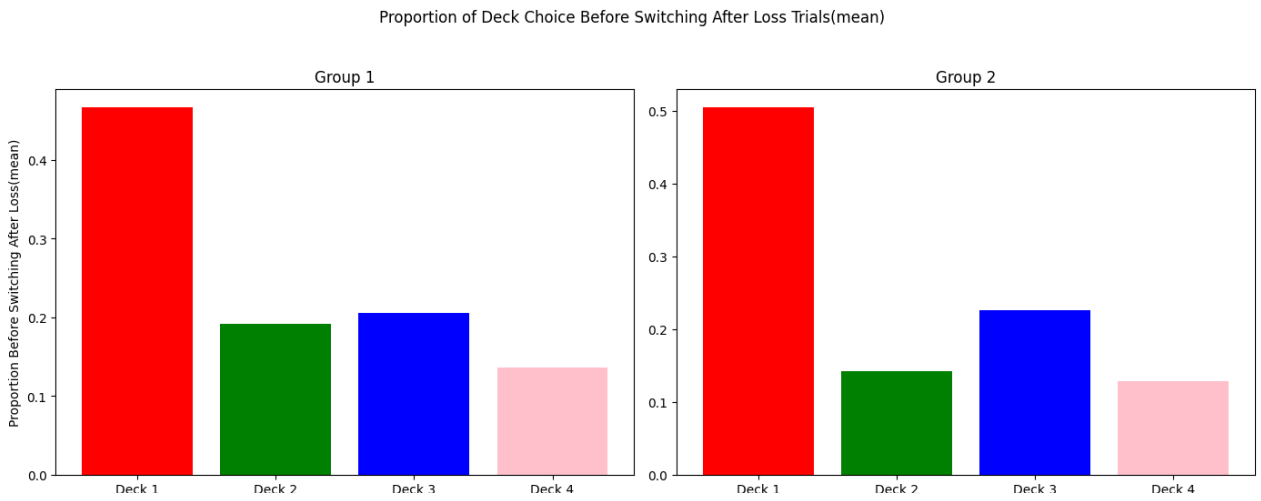
```
Analysis of deck choices BEFORE and AFTER loss trials:
Group 1 deck rankings (before loss):
Rank 1: Deck 1 (proportion: 0.524)
Rank 2: Deck 2 (proportion: 0.216)
Rank 3: Deck 4 (proportion: 0.153)
Rank 4: Deck 3 (proportion: 0.108)

Group 1 deck rankings (after loss):
Rank 1: Deck 2 (proportion: 0.375)
Rank 2: Deck 3 (proportion: 0.254)
Rank 3: Deck 4 (proportion: 0.248)
Rank 4: Deck 1 (proportion: 0.123)

Group 2 deck rankings (before loss):
Rank 1: Deck 1 (proportion: 0.504)
Rank 2: Deck 3 (proportion: 0.225)
Rank 3: Deck 2 (proportion: 0.142)
Rank 4: Deck 4 (proportion: 0.129)

Group 2 deck rankings (after loss):
Rank 1: Deck 2 (proportion: 0.403)
Rank 2: Deck 4 (proportion: 0.263)
Rank 3: Deck 3 (proportion: 0.205)
Rank 4: Deck 1 (proportion: 0.128)
```

Start coding or generate with AI.

Start coding or generate with AI.

```
# Function to determine the deck chosen before switching after a loss trial
def calculate_deck_before_switch(choice, loss, win):
```

```
def calculate_deck_before_switch(choice, loss, win):
    participants = choice.shape[0]
    trials = choice.shape[1]
    deck_before_switch = {1: 0, 2: 0, 3: 0, 4: 0}
    total_switches = 0

    for i in range(participants):
        for t in range(trials - 1):
            if win.iloc[i,t] + loss.iloc[i, t] < 0 and choice.iloc[i, t] != cho
                deck_before_switch[choice.iloc[i, t]] += 1
                total_switches += 1

    proportions_before_switch = {deck: count / total_switches for deck, count i
    return proportions_before_switch

# Calculate deck choice proportions before switching after loss trials for both
proportions_before_switch_group1 = calculate_deck_before_switch(choice['group1'
proportions_before_switch_group2 = calculate_deck_before_switch(choice['group2'


# Plotting the bar plots for deck choice before switching after loss trials
fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=False)

# Group 1 plot
axes[0].bar(['Deck 1', 'Deck 2', 'Deck 3', 'Deck 4'], list(proportions_before_s
axes[0].set_title('Group 1')
axes[0].set_xlabel('Deck')
axes[0].set_ylabel('Proportion Before Switching After Loss(mean)')

# Group 2 plot
axes[1].bar(['Deck 1', 'Deck 2', 'Deck 3', 'Deck 4'], list(proportions_before_s
axes[1].set_title('Group 2')
axes[1].set_xlabel('Deck')

plt.suptitle('Proportion of Deck Choice Before Switching After Loss Trials(mear
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

# Ranking decks in decreasing order based on their mean proportions for each gr
ranking_group1 = sorted(zip(['Deck 1', 'Deck 2', 'Deck 3', 'Deck 4'], proportic
ranking_group2 = sorted(zip(['Deck 1', 'Deck 2', 'Deck 3', 'Deck 4'], proportic
```



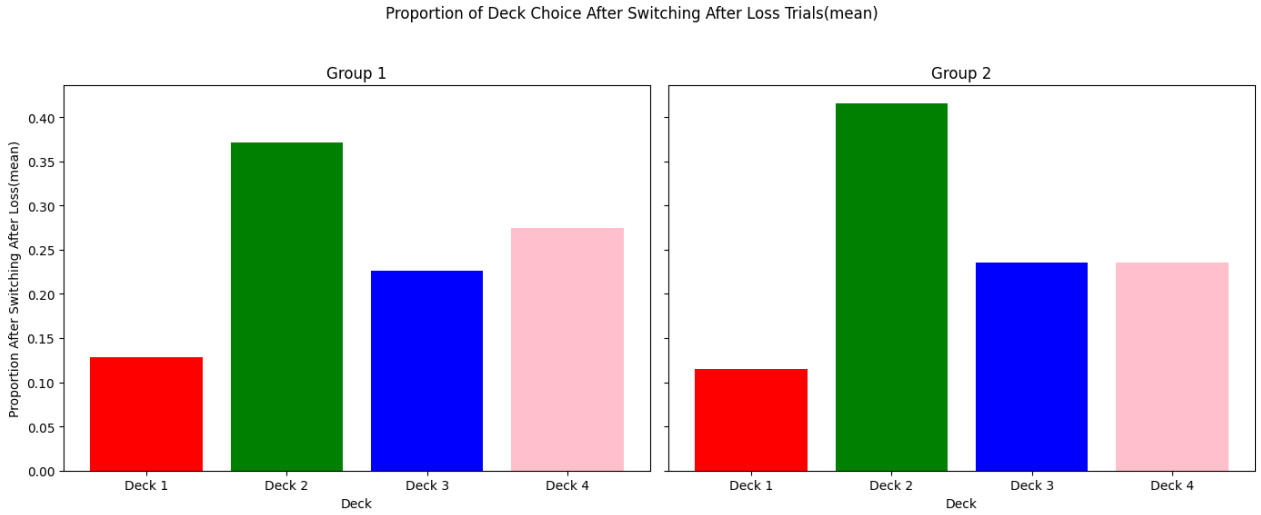
Deck Deck

```
print("Ranking of decks for Group 1 (based on mean proportions before switching
for rank, (deck, proportion) in enumerate(ranking_group1, start=1):
    print(f"Rank {rank}: {deck} (Proportion: {proportion:.3f})")

print("\nRanking of decks for Group 2 (based on mean proportions before switchi
for rank, (deck, proportion) in enumerate(ranking_group2, start=1):
    print(f"Rank {rank}: {deck} (Proportion: {proportion:.3f})")
```

```
Ranking of decks for Group 1 (based on mean proportions before switching af
Rank 1: Deck 1 (Proportion: 0.466)
Rank 2: Deck 3 (Proportion: 0.206)
Rank 3: Deck 2 (Proportion: 0.192)
Rank 4: Deck 4 (Proportion: 0.136)

Ranking of decks for Group 2 (based on mean proportions before switching af
Rank 1: Deck 1 (Proportion: 0.504)
Rank 2: Deck 3 (Proportion: 0.225)
Rank 3: Deck 2 (Proportion: 0.142)
Rank 4: Deck 4 (Proportion: 0.129)
```

```
# Function to determine the deck chosen after switching after a loss trial
def calculate_deck_after_switch(choice, loss, win):
    participants = choice.shape[0]
    trials = choice.shape[1]
    deck_after_switch = {1: 0, 2: 0, 3: 0, 4: 0}
    total_switches = 0

    for i in range(participants):
        for t in range(trials - 1):
            if win.iloc[i,t] + loss.iloc[i, t] < 0 and choice.iloc[i, t] != cho
                deck_after_switch[choice.iloc[i, t + 1]] += 1
                total_switches += 1

    proportions_after_switch = {deck: count / total_switches for deck, count ir
    return proportions_after_switch
```

```
# Calculate deck choice proportions after switching after loss trials for both
proportions_after_switch_group1 = calculate_deck_after_switch(choice['group1'],
proportions_after_switch_group2 = calculate_deck_after_switch(choice['group2'],



# Plotting the bar plots for deck choice after switching after loss trials
fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=True)

# Group 1 plot
axes[0].bar(['Deck 1', 'Deck 2', 'Deck 3', 'Deck 4'], list(proportions_after_swi
axes[0].set_title('Group 1')
axes[0].set_xlabel('Deck')
axes[0].set_ylabel('Proportion After Switching After Loss(mean)')

# Group 2 plot
axes[1].bar(['Deck 1', 'Deck 2', 'Deck 3', 'Deck 4'], list(proportions_after_swi
axes[1].set_title('Group 2')
axes[1].set_xlabel('Deck')

plt.suptitle('Proportion of Deck Choice After Switching After Loss Trials(mean)'
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



```
# Ranking decks in decreasing order based on their mean proportions for each gr
ranking_after_switch_group1 = sorted(zip(['Deck 1', 'Deck 2', 'Deck 3', 'Deck 4
ranking_after_switch_group2 = sorted(zip(['Deck 1', 'Deck 2', 'Deck 3', 'Deck 4

print("Ranking of decks for Group 1 (based on mean proportions after switching
for rank, (deck, proportion) in enumerate(ranking_after_switch_group1, start=1)
    print(f"Rank {rank}: {deck} (Proportion: {proportion:.3f})")

print("\nRanking of decks for Group 2 (based on mean proportions after switchir
for rank, (deck, proportion) in enumerate(ranking_after_switch_group2, start=1)
    print(f"Rank {rank}: {deck} (Proportion: {proportion:.3f})")
```

```
Ranking of decks for Group 1 (based on mean proportions after switching aft
Rank 1: Deck 2 (Proportion: 0.371)
Rank 2: Deck 4 (Proportion: 0.274)
Rank 3: Deck 3 (Proportion: 0.226)
Rank 4: Deck 1 (Proportion: 0.128)

Ranking of decks for Group 2 (based on mean proportions after switching aft
Rank 1: Deck 2 (Proportion: 0.415)
Rank 2: Deck 3 (Proportion: 0.235)
Rank 3: Deck 4 (Proportion: 0.235)
Rank 4: Deck 1 (Proportion: 0.115)
```

Start coding or generate with AI.