Contents

# Iowa Gambling Task Assignment

## Why was the Iowa Gambling Task created?

The Iowa Gambling Task (IGT) was created to study decision making. The original purpose of this paper was to investigate the decision making skills of a participant with damage to their ventromedial prefrontal cortex. The original paper, written in 1994, showed that participants with damaged prefrontal cortices showed less awareness to the future consequences of their choices. There have been further variations of this study carried out in recent years, to focus on human responses not necessarily associated to brain function.

## How does the Iowa Gambling Task work?

The game involves a participant who has been given a loan of $2000 and is told to maximise their winnings. There are 4 cards, numbered 1 through 4, and each card has a corresponding reward value and a potential loss value. Participants have to choose a number of cards one-by-one, but they do not know what each card will yield in advance, each time they choose a card they are shown whether they have won and/or lost money. The winning/losing values assigned to each card are not random, there are certain stacks of cards that will yield higher, more stable return, whereas some stacks may be considered high risk, offering high winnings but high losses too. Some participants will notice this and begin to choose the long-term, higher yielding cards more regularly, while others continue to choose the high-risk cards. This game used to be played with physical cards, but there are now virtual cards. A demo is available for the Iowa Gambling Task here.

## About the Datasets

### Dataset Origins

The sources of these 9 datasets can be found in this paper. The dataset contains results obtained by 10 different studies. These results were collected between the years of 2004 and 2015. The results shown in these datasets are representing 617 healthy participants.

### Dataset Contents

There are four values being measured in the datasets, the choice each subject made each trial, the reward value for making that choice, the penalty cost for making that choice and the study that they were a part of. All of the studies either have 95, 100 or 150 trials, so there are 4 datasets for each study size and for each value being measures, giving 12 datasets.

The datasets are formatted as follows:

- **choice_{#trials}**: The dataset containing the choices of subjects who played {#trials} trials
- **wi_{#trials}**: The dataset containing the reward values of subjects who played {#trials} trials
- **lo_{#trials}**: The dataset containing the losses of subjects who played {#trials} trials
- **index_{#trials}**: The dataset containing the studies that the subjects who played {#trials} trials participated in

### Dataset Structure

Each dataset follows the same structural layout. All of the datasets have named columns and rows. The column names of each dataset are the same depending on the value being measured. For example, all three datasets that measure the winning values (wi_95, wi_100, wi_150) all have the naming convention 'Wins_{trial}' for columns. Similarly, the naming conventions for the choices, losses and study are 'Choice_{trial}', 'Losses_{trial}' and 'Study' respectively.

There are a total of 617 subjects accounted for in these datasets. 15 subjects played 95 trials, 504 subjects played 100 trials and 98 played 150 trials.

# Data Inspection

This is an initial inspection of the datasets. This will ensure that the data will be ready to used for clustering.

## 1. Importing relevant packages and Datasets

```
import pandas as pd
import numpy as np
from functions import *
```

```
choice_95 = pd.DataFrame(pd.read_csv('../data/choice_95.csv'))
win_95 = pd.DataFrame(pd.read_csv('../data/wi_95.csv'))
loss_95 = pd.DataFrame(pd.read_csv('../data/lo_95.csv'))

choice_100 = pd.DataFrame(pd.read_csv('../data/choice_100.csv'))
win_100 = pd.DataFrame(pd.read_csv('../data/wi_100.csv'))
loss_100 = pd.DataFrame(pd.read_csv('../data/lo_100.csv'))

choice_150 = pd.DataFrame(pd.read_csv('../data/choice_150.csv'))
win_150 = pd.DataFrame(pd.read_csv('../data/wi_150.csv'))
loss_150 = pd.DataFrame(pd.read_csv('../data/lo_150.csv'))
```

## 2. Ensuring Data is Clean

It is important to check that the data is clean so it can be processed correctly for analysis.

This includes checking that the data is accurate, that there are no structural errors in the data, that there are no null values and investigating outliers or duplicate values.

The data seems to be reliable, as they are all results from studies performed in the papers listed here.

All of the datasets seem to be inherent of identical structures, with 'Choice_{number}', 'Wins_{number}' and 'Losses_{number}' as the column names for the choices, wins and losses datasets respectively. The row names in all datasets follow the same structure also, being 'Subj_{number}'

The code snippets below examine the datasets for outliers and duplicate rows and ensure they are all the same type.

```
datasets = [choice_95, win_95, loss_95,
            win_100, loss_100,
           choice_150, win_150, loss_150,
           choice_100]

for dataset in datasets:
    check_for_null_values(dataset)
    check_for_duplicate_rows(dataset)
```

```
There were 0 duplicates found.
There were 0 duplicates found.
There were 0 duplicates found.
There were 3 duplicates found.
There were 2 duplicates found.
There were 1 duplicates found.
There were 1 duplicates found.
There were 0 duplicates found.
There were 2 duplicates found.
```

The check_for_null_values() function uses an assertion, so as there are no errors in the output, it shows that all datasets contain non-null values only.

The check_for_duplicate_rows() function checks the quantity of duplicate rows. This function does not use an assertion, as it is entirely possible that two subjects participating in the task may have received the same sequence of rewards or penalties. The validity of this function is subjective, but I do not believe that there are any mistaken duplicates in these datasets.

```
check_for_outlier(choice_95, [1,2,3,4])
check_for_outlier(choice_100, [1,2,3,4])
check_for_outlier(choice_150, [1,2,3,4])

check_for_outlier(win_95, range(0,200))
check_for_outlier(win_100, range(0,200))
check_for_outlier(win_150, range(0,200))

check_for_outlier(loss_95, range(-3000, 1))
check_for_outlier(loss_100, range(-3000, 1))
check_for_outlier(loss_150, range(-3000, 1))
```

```
check_all_data_type(choice_95, np.int64)
check_all_data_type(choice_100, np.int64)
check_all_data_type(choice_150, np.int64)

check_all_data_type(win_95, np.int64)
check_all_data_type(win_100, np.int64)
check_all_data_type(win_150, np.int64)

check_all_data_type(loss_95, np.int64)
check_all_data_type(loss_100, np.int64)
check_all_data_type(loss_150, np.int64)
```

Both of the functions above, 'check_for_outlier()' and 'check_all_data_types()', both use assertion statements. No output shows us that all datasets are clear from outliers or mismatched data types.

The datasets seem to be clean, with no inconsistent data types or structural differences, and no obvious outliers or inaccuracies.

# Exploring Rewards and Loss Systems

There are 3 payout schemes that are implemented in each study, and this notebook intends to see if this is reflected in and/or has an influnce on the subjects' choices.

## Importing Relevant Packages and Datasets

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from functions import *
```

```
index_95 = pd.DataFrame(pd.read_csv('../data/index_95.csv'))
choice_95 = pd.DataFrame(pd.read_csv('../data/choice_95.csv'))
win_95 = pd.DataFrame(pd.read_csv('../data/wi_95.csv'))
loss_95 = pd.DataFrame(pd.read_csv('../data/lo_95.csv'))

index_100 = pd.DataFrame(pd.read_csv('../data/index_100.csv'))
choice_100 = pd.DataFrame(pd.read_csv('../data/choice_100.csv'))
win_100 = pd.DataFrame(pd.read_csv('../data/wi_100.csv'))
loss_100 = pd.DataFrame(pd.read_csv('../data/lo_100.csv'))

index_150 = pd.DataFrame(pd.read_csv('../data/index_150.csv'))
choice_150 = pd.DataFrame(pd.read_csv('../data/choice_150.csv'))
win_150 = pd.DataFrame(pd.read_csv('../data/wi_150.csv'))
loss_150 = pd.DataFrame(pd.read_csv('../data/lo_150.csv'))
```

## Creating Separate Datasets per Study

```
## Payoff 1
study_fridberg = index_95
study_maia = index_100.iloc[181:221]
study_worthy = index_100.iloc[469:]

# Payoff 2
study_hortsmann = index_100.iloc[:162]
study_kjome = index_100.iloc[162:181]
study_steingrover_inprep = index_100.iloc[221:291]
study_premkumar = index_100.iloc[291:316]
study_steingrover_2011 = index_150.iloc[:57]
study_wetzels = index_150.iloc[57:]

# Payoff 3
study_wood = index_100.iloc[316:469]
```

```
## Payoff 1
choice_fridberg = choice_95
choice_maia = choice_100.iloc[181:221]
choice_worthy = choice_100.iloc[469:]

# Payoff 2
choice_hortsmann = choice_100.iloc[:162]
choice_kjome = choice_100.iloc[162:181]
choice_steingrover_inprep = choice_100.iloc[221:291]
choice_premkumar = choice_100.iloc[291:316]
choice_steingrover_2011 = choice_150.iloc[:57]
choice_wetzels = choice_150.iloc[57:]

# Payoff 3
choice_wood = choice_100.iloc[316:469]
```

```
# Payoff 1
win_fridberg = win_95
win_maia = win_100.iloc[181:221]
win_worthy = win_100.iloc[469:]

# Payoff 2
win_hortsmann = win_100.iloc[:162]
win_kjome = win_100.iloc[162:181]
win_steingrover_inprep = win_100.iloc[221:291]
win_premkumar = win_100.iloc[291:316]
win_steingrover_2011 = win_150.iloc[:57]
win_wetzels = win_150.iloc[57:]

# Payoff 3
win_wood = win_100.iloc[316:469]
```

```
# Payoff 1
loss_fridberg = loss_95
loss_maia = loss_100.iloc[181:221]
loss_worthy = loss_100.iloc[469:]

# Payoff 2
loss_hortsmann = loss_100.iloc[:162]
loss_kjome = loss_100.iloc[162:181]
loss_steingrover_inprep = loss_100.iloc[221:291]
loss_premkumar = loss_100.iloc[291:316]
loss_steingrover_2011 = loss_150.iloc[:57]
loss_wetzels = loss_150.iloc[57:]

# Payoff 3
loss_wood = loss_100.iloc[316:469]
```

## Deeper Analysis of the Risk/Reward Associated with each Choice and Study

```
## Payoff 1

times_1_chosen = 0
times_1_loss = 0
losses_1 = []
times_1_win = 0
wins_1 = []

times_2_chosen = 0
times_2_loss = 0
losses_2 = []
times_2_win = 0
wins_2 = []

times_3_chosen = 0
times_3_loss = 0
losses_3 = []
times_3_win = 0
wins_3 = []

times_4_chosen = 0
times_4_loss = 0
losses_4 = []
times_4_win = 0
wins_4 = []


for choice_win_loss_dataset in [[choice_fridberg, win_fridberg, loss_fridberg],
                          [choice_maia, win_maia, loss_maia],
                          [choice_worthy, win_worthy, win_worthy]]:
    choice_df = choice_win_loss_dataset[0]
    win_df = choice_win_loss_dataset[1]
    loss_df = choice_win_loss_dataset[2]
    num_subjects = len(choice_df.iloc[:,0])
    num_trials = len(choice_df.iloc[0])
    for subject in range(0,num_subjects):
        for round in range(0,num_trials):
            choice_made = choice_df.iloc[subject][round]
            if choice_made == 1:
                times_1_chosen += 1
                loss = loss_df.iloc[subject][round]
                win = win_df.iloc[subject][round]
                if loss < 0:
                    times_1_loss +=1
                    losses_1.append(loss)
                if win > 0:
                    times_1_win += 1
                    wins_1.append(win)
            elif choice_made == 2:
                times_2_chosen += 1
                loss = loss_df.iloc[subject][round]
                win = win_df.iloc[subject][round]
                if loss < 0:
                    times_2_loss +=1
                    losses_2.append(loss)
                if win > 0:
                    times_2_win += 1
                    wins_2.append(win)
            elif choice_made == 3:
                times_3_chosen += 1
                loss = loss_df.iloc[subject][round]
                win = win_df.iloc[subject][round]
                if loss < 0:
                    times_3_loss +=1
                    losses_3.append(loss)
                if win > 0:
                    times_3_win += 1
                    wins_3.append(win)
            elif choice_made == 4:
                times_4_chosen += 1
                loss = loss_df.iloc[subject][round]
                win = win_df.iloc[subject][round]
                if loss < 0:
                    times_4_loss +=1
                    losses_4.append(loss)
                if win > 0:
                    times_4_win += 1
                    wins_4.append(win)

print("Times 1 was chosen=", times_1_chosen)
print("Times 1 was a loss=", times_1_loss)
print("Times 1 was a win=", times_1_win)
print("Average amount lost when choosing 1= ", str(sum(losses_1)/times_1_chosen))
print("Average amount won when choosing 1= ", str(sum(wins_1)/times_1_chosen))

print("Times 2 was chosen=", times_2_chosen)
print("Times 2 was a loss=", times_2_loss)
print("Times 2 was a win=", times_2_win)
print("Average amount lost when choosing 2= ", str(sum(losses_2)/times_2_chosen))
print("Average amount won when choosing 2= ", str(sum(wins_2)/times_2_chosen))

print("Times 3 was chosen=", times_3_chosen)
```

```
print("Times 3 was a loss=", times_3_loss)
print("Times 3 was a win=", times_3_win)
print("Average amount lost when choosing 3= ", str(sum(losses_3)/times_3_chosen))
print("Average amount won when choosing 3= ", str(sum(wins_3)/times_3_chosen))

print("Times 4 was chosen=", times_4_chosen)
print("Times 4 was a loss=", times_4_loss)
print("Times 4 was a win=", times_4_win)
print("Average amount lost when choosing 4= ", str(sum(losses_4)/times_4_chosen))
print("Average amount won when choosing 4= ", str(sum(wins_4)/times_4_chosen))
```

```
Times 1 was chosen= 1400
Times 1 was a loss= 383
Times 1 was a win= 1400
Average amount lost when choosing 1=  -69.32142857142857
Average amount won when choosing 1=  100.0
Times 2 was chosen= 2516
Times 2 was a loss= 153
Times 2 was a win= 2516
Average amount lost when choosing 2=  -76.01351351351352
Average amount won when choosing 2=  100.0
Times 3 was chosen= 2232
Times 3 was a loss= 694
Times 3 was a win= 2232
Average amount lost when choosing 3=  -15.367383512544803
Average amount won when choosing 3=  50.0
Times 4 was chosen= 2777
Times 4 was a loss= 169
Times 4 was a win= 2777
Average amount lost when choosing 4=  -15.214259992797983
Average amount won when choosing 4=  50.0
```

```
## Payoff 2

times_1_chosen = 0
times_1_loss = 0
losses_1 = []
times_1_win = 0
wins_1 = []

times_2_chosen = 0
times_2_loss = 0
losses_2 = []
times_2_win = 0
wins_2 = []

times_3_chosen = 0
times_3_loss = 0
losses_3 = []
times_3_win = 0
wins_3 = []

times_4_chosen = 0
times_4_loss = 0
losses_4 = []
times_4_win = 0
wins_4 = []

for choice_win_loss_dataset in [[choice_hortsmann, win_hortsmann, loss_hortsmann],
                          [choice_kjome, win_kjome, loss_kjome],
                          [choice_steingrover_inprep, win_steingrover_inprep,
win_steingrover_inprep],
                          [choice_premkumar, win_premkumar, loss_premkumar],
                          [choice_steingrover_2011, win_steingrover_2011, win_steingrover_2011],
                          [choice_wetzels, win_wetzels, loss_wetzels],]:
    choice_df = choice_win_loss_dataset[0]
    win_df = choice_win_loss_dataset[1]
    loss_df = choice_win_loss_dataset[2]
    num_subjects = len(choice_df.iloc[:,0])
    num_trials = len(choice_df.iloc[0])
    for subject in range(0,num_subjects):
        for round in range(0,num_trials):
            choice_made = choice_df.iloc[subject][round]
            if choice_made == 1:
                times_1_chosen += 1
                loss = loss_df.iloc[subject][round]
                win = win_df.iloc[subject][round]
                if loss < 0:
                    times_1_loss +=1
                    losses_1.append(loss)
                if win > 0:
                    times_1_win += 1
                    wins_1.append(win)
            elif choice_made == 2:
                times_2_chosen += 1
                loss = loss_df.iloc[subject][round]
                win = win_df.iloc[subject][round]
                if loss < 0:
                    times_2_loss +=1
                    losses_2.append(loss)
                if win > 0:
                    times_2_win += 1
                    wins_2.append(win)
            elif choice_made == 3:
                times_3_chosen += 1
                loss = loss_df.iloc[subject][round]
                win = win_df.iloc[subject][round]
                if loss < 0:
                    times_3_loss +=1
                    losses_3.append(loss)
                if win > 0:
                    times_3_win += 1
                    wins_3.append(win)
            elif choice_made == 4:
                times_4_chosen += 1
                loss = loss_df.iloc[subject][round]
                win = win_df.iloc[subject][round]
                if loss < 0:
                    times_4_loss +=1
                    losses_4.append(loss)
                if win > 0:
                    times_4_win += 1
                    wins_4.append(win)

print("Times 1 was chosen=", times_1_chosen)
print("Times 1 was a loss=", times_1_loss)
print("Times 1 was a win=", times_1_win)
print("Average amount lost when choosing 1= ", str(sum(losses_1)/times_1_chosen))
print("Average amount won when choosing 1= ", str(sum(wins_1)/times_1_chosen))

print("Times 2 was chosen=", times_2_chosen)
print("Times 2 was a loss=", times_2_loss)
print("Times 2 was a win=", times_2_win)
print("Average amount lost when choosing 2= ", str(sum(losses_2)/times_2_chosen))
```

```
print("Average amount won when choosing 2= ", str(sum(wins_2)/times_2_chosen))

print("Times 3 was chosen=", times_3_chosen)
print("Times 3 was a loss=", times_3_loss)
print("Times 3 was a win=", times_3_win)
print("Average amount lost when choosing 3= ", str(sum(losses_3)/times_3_chosen))
print("Average amount won when choosing 3= ", str(sum(wins_3)/times_3_chosen))

print("Times 4 was chosen=", times_4_chosen)
print("Times 4 was a loss=", times_4_loss)
print("Times 4 was a win=", times_4_win)
print("Average amount lost when choosing 4= ", str(sum(losses_4)/times_4_chosen))
print("Average amount won when choosing 4= ", str(sum(wins_4)/times_4_chosen))
```

```
Times 1 was chosen= 5793
Times 1 was a loss= 2039
Times 1 was a win= 5793
Average amount lost when choosing 1=  -88.14949076471603
Average amount won when choosing 1=  100.62316588986708
Times 2 was chosen= 13683
Times 2 was a loss= 851
Times 2 was a win= 13683
Average amount lost when choosing 2=  -79.9714974786231
Average amount won when choosing 2=  100.71548636994811
Times 3 was chosen= 10485
Times 3 was a loss= 3182
Times 3 was a win= 10485
Average amount lost when choosing 3=  -14.82832618025751
Average amount won when choosing 3=  50.59418216499762
Times 4 was chosen= 12339
Times 4 was a loss= 825
Times 4 was a win= 12339
Average amount lost when choosing 4=  -17.240051868060622
Average amount won when choosing 4=  51.030472485614716
```

```
## Payoff 3

times_1_chosen = 0
times_1_loss = 0
losses_1 = []
times_1_win = 0
wins_1 = []

times_2_chosen = 0
times_2_loss = 0
losses_2 = []
times_2_win = 0
wins_2 = []

times_3_chosen = 0
times_3_loss = 0
losses_3 = []
times_3_win = 0
wins_3 = []

times_4_chosen = 0
times_4_loss = 0
losses_4 = []
times_4_win = 0
wins_4 = []

choice_df = choice_wood
win_df = win_wood
loss_df = loss_wood
num_subjects = len(choice_df.iloc[:,0])
num_trials = len(choice_df.iloc[0])
for subject in range(0,num_subjects):
    for round in range(0,num_trials):
        choice_made = choice_df.iloc[subject][round]
        if choice_made == 1:
            times_1_chosen += 1
            loss = loss_df.iloc[subject][round]
            win = win_df.iloc[subject][round]
            if loss < 0:
                times_1_loss +=1
                losses_1.append(loss)
            if win > 0:
                times_1_win += 1
                wins_1.append(win)
        elif choice_made == 2:
            times_2_chosen += 1
            loss = loss_df.iloc[subject][round]
            win = win_df.iloc[subject][round]
            if loss < 0:
                times_2_loss +=1
                losses_2.append(loss)
            if win > 0:
                times_2_win += 1
                wins_2.append(win)
        elif choice_made == 3:
            times_3_chosen += 1
            loss = loss_df.iloc[subject][round]
            win = win_df.iloc[subject][round]
            if loss < 0:
                times_3_loss +=1
                losses_3.append(loss)
            if win > 0:
                times_3_win += 1
                wins_3.append(win)
        elif choice_made == 4:
            times_4_chosen += 1
            loss = loss_df.iloc[subject][round]
            win = win_df.iloc[subject][round]
            if loss < 0:
                times_4_loss +=1
                losses_4.append(loss)
            if win > 0:
                times_4_win += 1
                wins_4.append(win)

print("Times 1 was chosen=", times_1_chosen)
print("Times 1 was a loss=", times_1_loss)
print("Times 1 was a win=", times_1_win)
print("Average amount lost when choosing 1= ", str(sum(losses_1)/times_1_chosen))
print("Average amount won when choosing 1= ", str(sum(wins_1)/times_1_chosen))

print("Times 2 was chosen=", times_2_chosen)
print("Times 2 was a loss=", times_2_loss)
print("Times 2 was a win=", times_2_win)
print("Average amount lost when choosing 2= ", str(sum(losses_2)/times_2_chosen))
print("Average amount won when choosing 2= ", str(sum(wins_2)/times_2_chosen))

print("Times 3 was chosen=", times_3_chosen)
print("Times 3 was a loss=", times_3_loss)
print("Times 3 was a win=", times_3_win)
print("Average amount lost when choosing 3= ", str(sum(losses_3)/times_3_chosen))
print("Average amount won when choosing 3= ", str(sum(wins_3)/times_3_chosen))
```

```
print("Times 4 was chosen=", times_4_chosen)
print("Times 4 was a loss=", times_4_loss)
print("Times 4 was a win=", times_4_win)
print("Average amount lost when choosing 4= ", str(sum(losses_4)/times_4_chosen))
print("Average amount won when choosing 4= ", str(sum(wins_4)/times_4_chosen))
```

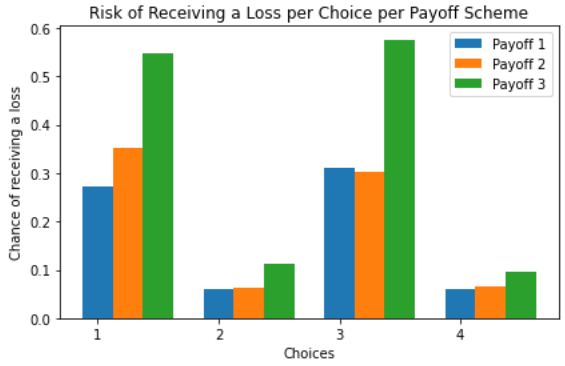
```
Times 1 was chosen= 2564
Times 1 was a loss= 1402
Times 1 was a win= 2564
Average amount lost when choosing 1=  -138.59204368174727
Average amount won when choosing 1=  104.9804992199688
Times 2 was chosen= 4402
Times 2 was a loss= 490
Times 2 was a win= 4402
Average amount lost when choosing 2=  -174.6365288505225
Average amount won when choosing 2=  111.7014993184916
Times 3 was chosen= 3523
Times 3 was a loss= 2029
Times 3 was a win= 3523
Average amount lost when choosing 3=  -26.078626170877094
Average amount won when choosing 3=  54.53306840760715
Times 4 was chosen= 4811
Times 4 was a loss= 458
Times 4 was a win= 4811
Average amount lost when choosing 4=  -27.03699854500104
Average amount won when choosing 4=  56.877987944294325
```

# Creating graphs to see differences

## Risk of a Penalty per choice per Payoff Scheme

```
# Risk of loss per choice per study

payoff1_risk = [383/1400,
                153/2516,
                694/2232,
                169/2777]
payoff2_risk = [2039/5793,
                851/13683,
                3182/10485,
                825/12339]
payoff3_risk = [1402/2564,
                490/4402,
                2029/3523,
                458/4811]
labels = [1, 2, 3, 4]
```

```
fig, ax = plt.subplots()
x = np.arange(len(labels))  # the label locations
width = 0.25

rects1 = ax.bar(x, payoff1_risk, width, label='Payoff 1')
rects2 = ax.bar(x +.25, payoff2_risk, width, label='Payoff 2')
rects2 = ax.bar(x + .5, payoff3_risk, width, label='Payoff 3')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Chance of receiving a loss')
ax.set_xlabel('Choices')
ax.set_title('Risk of Receiving a Loss per Choice per Payoff Scheme')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

fig.tight_layout()

plt.show()
```



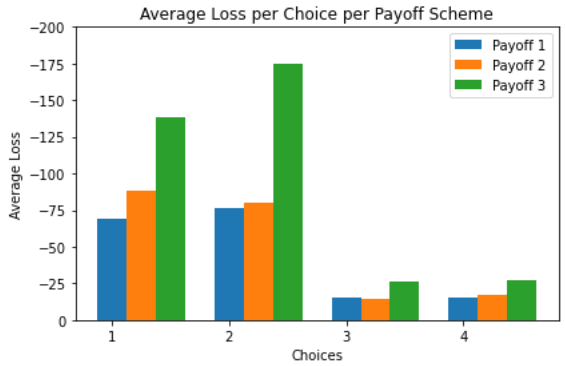

As shown in the above graph and indicated in this paper, the decks 1 and 3 have more frequent penalties for its subjects. All three payoff schemes seem to be very similar.

In payoff scheme 1, deck 3 holds a higher penalty risk than deck 1, although from the graph we can see that for payoff scheme 2, it is riskier to choose from 1. This could explain why decks 1 and 3 are chosen the least amount of times, even though deck 3 is considered a 'good deck'

## Average Penalty Received per Choice per Payoff Scheme

```
# Average penalty per choice per study

payoff1_avg_penalty = [-69.32142857142857,
                       -76.01351351351352,
                       -15.367383512544803,
                       -15.214259992797983]
payoff2_avg_penalty = [-88.14949076471603,
                -79.9714974786231,
                -14.82832618025751,
                -17.240051868060622]
payoff3_avg_penalty = [-138.59204368174727,
                -174.6365288505225,
                -26.078626170877094,
                -27.03699854500104]
labels = [1, 2, 3, 4]
```

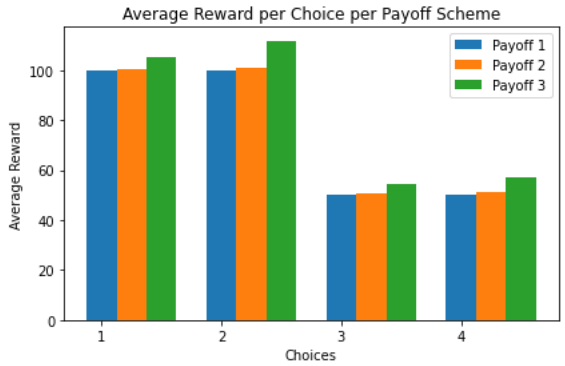
```
fig, ax = plt.subplots()
x = np.arange(len(labels))  # the label locations
width = 0.25

rects1 = ax.bar(x, payoff1_avg_penalty, width, label='Payoff 1')
rects2 = ax.bar(x +.25, payoff2_avg_penalty, width, label='Payoff 2')
rects2 = ax.bar(x + .5, payoff3_avg_penalty, width, label='Payoff 3')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Average Loss')
ax.set_xlabel('Choices')
ax.set_title('Average Loss per Choice per Payoff Scheme')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.set_ylim(0, -200)
ax.legend()

fig.tight_layout()

plt.show()
```



### Average Reward Received per Choice per Payoff Scheme

```
# Average reward per choice per study

payoff1_avg_reward = [100,
                      100,
                      50,
                      50]
payoff2_avg_reward = [100.62316588986708,
                100.71548636994811,
                50.59418216499762,
                51.030472485614716]
payoff3_avg_reward = [104.9804992199688,
                111.7014993184916,
                54.53306840760715,
                56.877987944294325]
labels = [1, 2, 3, 4]
```

```
fig, ax = plt.subplots()
x = np.arange(len(labels))  # the label locations
width = 0.25

rects1 = ax.bar(x, payoff1_avg_reward, width, label='Payoff 1')
rects2 = ax.bar(x +.25, payoff2_avg_reward, width, label='Payoff 2')
rects2 = ax.bar(x + .5, payoff3_avg_reward, width, label='Payoff 3')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Average Reward')
ax.set_xlabel('Choices')
ax.set_title('Average Reward per Choice per Payoff Scheme')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

fig.tight_layout()

plt.show()
```




It's clear that there is not much deviation between the reward values for each payoff scheme. Payoff Scheme 3 seems to offer the largest reward by a small margin.

# Exploratory Work

## Importing Relevant Packages and Datasets

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from functions import *
```

```
choice_95 = pd.DataFrame(pd.read_csv('../data/choice_95.csv'))
win_95 = pd.DataFrame(pd.read_csv('../data/wi_95.csv'))
loss_95 = pd.DataFrame(pd.read_csv('../data/lo_95.csv'))

choice_100 = pd.DataFrame(pd.read_csv('../data/choice_100.csv'))
win_100 = pd.DataFrame(pd.read_csv('../data/wi_100.csv'))
loss_100 = pd.DataFrame(pd.read_csv('../data/lo_100.csv'))

choice_150 = pd.DataFrame(pd.read_csv('../data/choice_150.csv'))
win_150 = pd.DataFrame(pd.read_csv('../data/wi_150.csv'))
loss_150 = pd.DataFrame(pd.read_csv('../data/lo_150.csv'))
```

```
datasets = [choice_95, win_95, loss_95,
          choice_100, win_100, loss_100,
          choice_150, win_150, loss_150]

for dataset in datasets:
    change_columns_to_int(dataset)
```

Changing the column names from strings to integers allows for plotting on a graph

## Popularity of each choice throughout

We know that there are certain decks that yield a higher return than others, but the subjects participating in the task do not.

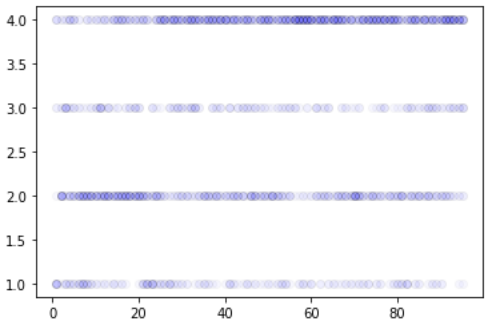
The following code snippets will be exploring the popularity of each choice throughout the game.

```
array_95 = create_plottable_array(choice_95)
array_100_pt1 = create_plottable_array(choice_100.iloc[:168])
array_100_pt2 = create_plottable_array(choice_100.iloc[168:336])
array_100_pt3 = create_plottable_array(choice_100.iloc[336:])
array_150 = create_plottable_array(choice_150)
```

As there were 504 subjects contained in the 'array_100' dataset, they have been split into three groups to allow for clearer analysis of the graphs

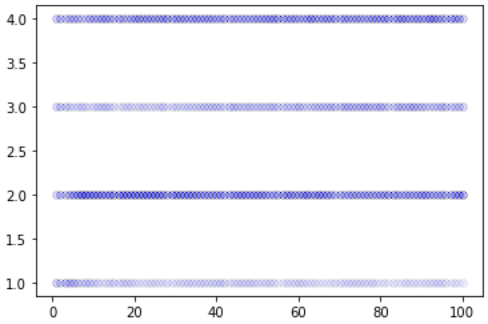
```
# plot the data points
plt.scatter(
   array_95[:, 0], array_95[:, 1],
   c='blue', alpha=0.03,
   edgecolor='black'
)
```

```
<matplotlib.collections.PathCollection at 0x1bb9ba112b0>
```



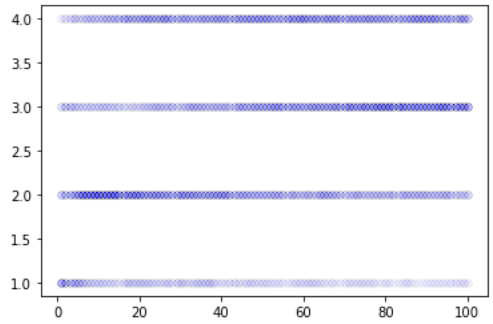

```
# plot the data points
plt.scatter(
   array_100_pt1[:, 0], array_100_pt1[:, 1],
   c='blue', alpha=0.002,
   edgecolor='black'
)
```

```
<matplotlib.collections.PathCollection at 0x1bb9bf28df0>
```



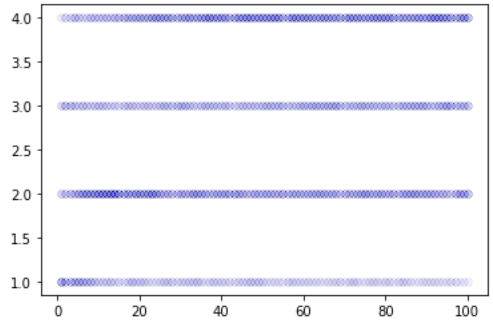
```
# plot the data points
plt.scatter(
   array_100_pt2[:, 0], array_100_pt2[:, 1],
   c='blue', alpha=0.002,
   edgecolor='black'
)
```

```
<matplotlib.collections.PathCollection at 0x1bb9b8d5dc0>
```



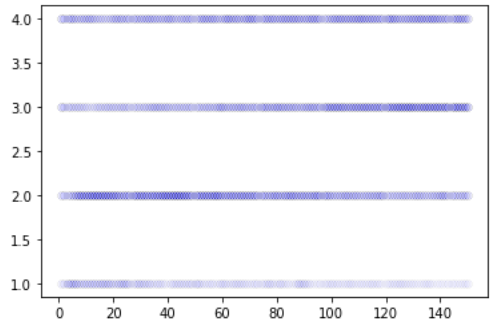

```
# plot the data points
plt.scatter(
   array_100_pt3[:, 0], array_100_pt3[:, 1],
   c='blue', alpha=0.002,
   edgecolor='black'
)
```

```
<matplotlib.collections.PathCollection at 0x1bb9c283cd0>
```




```
# plot the data points
plt.scatter(
   array_150[:, 0], array_150[:, 1],
   c='blue', alpha=0.005,
   edgecolor='black'
)
```

```
<matplotlib.collections.PathCollection at 0x1bb9c2ef730>
```




The above graphs are very similar, this implies that the popularity of choices is independent of the number of trials the participant played.

Choice 1 seems to be at it's most popular at the beginning of the game, but is much less likely to be chosen as the trials continue.

Choice 2 is very popular at the beginning, but decreases slightly towards the end.

Choice 3 is not very popular at the beginning, but gains some popularity towards the end of the task.

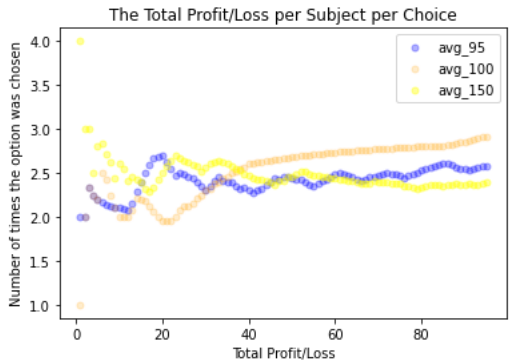
Choice 4 is consistently popular throughout the entire task.

## Viewing the running average of a selected group of subjects

```
running_average_95 = create_running_average_array(choice_95, 1)
running_average_100 = create_running_average_array(choice_100, 452)
running_average_150 = create_running_average_array(choice_150, 83)
```

```
plt.scatter(
   running_average_95[:, 0], running_average_95[:, 1],
    s=25, alpha=0.3, c='blue',
    marker='o',
    label='avg_95'
)

plt.scatter(
   running_average_100[:, 0], running_average_100[:, 1],
    s=25, alpha=0.2, c='orange',
    marker='o',
    label='avg_100'
)

plt.scatter(
   running_average_150[:, 0], running_average_150[:, 1],
    s=25, alpha=0.4, c='yellow',
    marker='o',
    label='avg_150'
)

plt.title("The Total Profit/Loss per Subject per Choice")
plt.xlabel("Total Profit/Loss")
plt.ylabel("Number of times the option was chosen")
plt.legend(scatterpoints=1)
```

```
<matplotlib.legend.Legend at 0x1bb9c349a60>
```



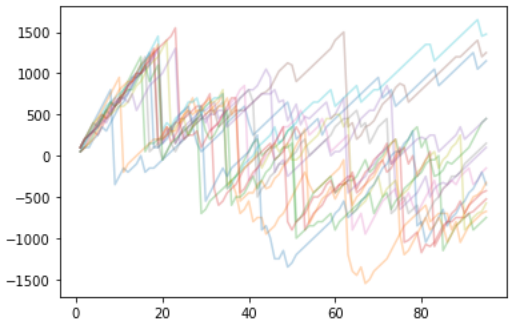

The graphs above are not representative of the entire group of participants, as plotting the choices of 617 people on a chart would not be legible.

The few selected subjects are showing an expected scattered plot at the beginning of the task, with the direction of the plot becoming narrower towards the end.

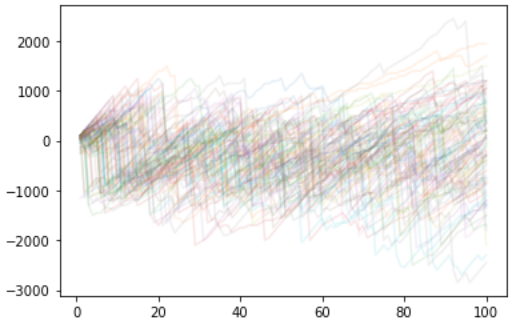
These figures will not be useful for clustering as the influence of the choice on the running average becomes smaller and smaller and is not hugely representative of their choices towards the end, although it is an interesting observation.

## Analysing Net Profit for Each Round

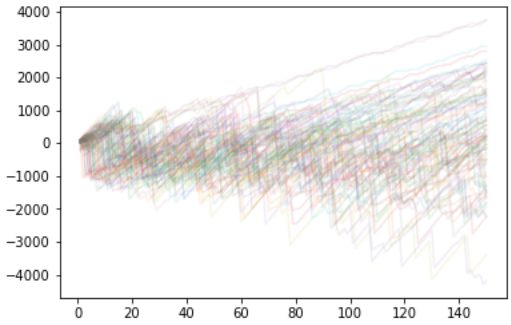
```
trials_95 = win_95.columns.tolist()
for subject in range(0,len(win_95)):
    net_profits = create_net_profit_loss_list(win_95, loss_95, subject)
    plt.plot(trials_95, net_profits, alpha = 0.3)
plt.show()
```




```
trials_100 = win_100.columns.tolist()
for subject in range(0,len(trials_100)):
    net_profits = create_net_profit_loss_list(win_100, loss_100, subject)
    plt.plot(trials_100, net_profits, alpha=0.1)
plt.show()
```




```
trials_150 = win_150.columns.tolist()
for subject in range(0,len(win_150)):
    net_profits = create_net_profit_loss_list(win_150, loss_150, subject)
    plt.plot(trials_150, net_profits, alpha = 0.1)
plt.show()
```




It's difficult to draw conclusions from the above diagrams. There seems to be a slight increase for majority of subjects until around the 10-20 trial mark when it takes a sharp decline. This could be due to the 'bad decks' having quite low penalties at the beginning and gradually getting higher penalties, as outlined in this paper.

For the diagrams shown in the 100-trial and 150-trial datasets, it seems there are an equal distribution of people's net profits/losses around the 0 mark. This shows that not everyone in the trial identified the 'good decks'.

# Cluster 1: Profit and Loss vs. Times Chosen per Choice

This cluster aims to spot a trend/correlation between the total net profit/loss made by each subject for each choice option and the proportion of times they chose that option.

Each subject will have the total profits/losses for choice 1, 2, 3 and 4, meaning there are 2468 [number of subjects=617] x [4 choices] datapoints in the graph.

## Importing Relevant Packages and Datasets

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from functions import *
```

```
choice_95 = pd.DataFrame(pd.read_csv('../data/choice_95.csv'))
win_95 = pd.DataFrame(pd.read_csv('../data/wi_95.csv'))
loss_95 = pd.DataFrame(pd.read_csv('../data/lo_95.csv'))

choice_100 = pd.DataFrame(pd.read_csv('../data/choice_100.csv'))
win_100 = pd.DataFrame(pd.read_csv('../data/wi_100.csv'))
loss_100 = pd.DataFrame(pd.read_csv('../data/lo_100.csv'))

choice_150 = pd.DataFrame(pd.read_csv('../data/choice_150.csv'))
win_150 = pd.DataFrame(pd.read_csv('../data/wi_150.csv'))
loss_150 = pd.DataFrame(pd.read_csv('../data/lo_150.csv'))
```

## Assigning Variables for Net Profit/Loss and Choice Proportions and Visualising them

```
## Creating initial lists with only the 95-trial dataset
output_95 = create_net_profit_vs_count_list(choice_95, win_95, loss_95)
profit_loss_list = output_95[0]
chosen_count = output_95[1]

# Adding the 100-trial dataset to the lists
output_100 = create_net_profit_vs_count_list(choice_100, win_100, loss_100,
                                             profit_loss_list, chosen_count)
profit_loss_list = output_100[0]
chosen_count = output_100[1]

# Adding the 150-trial dataset and having our final list
output_150 = create_net_profit_vs_count_list(choice_150, win_150, loss_150,
                                             profit_loss_list, chosen_count)
profit_loss_list = output_150[0]
chosen_count = output_150[1]
```

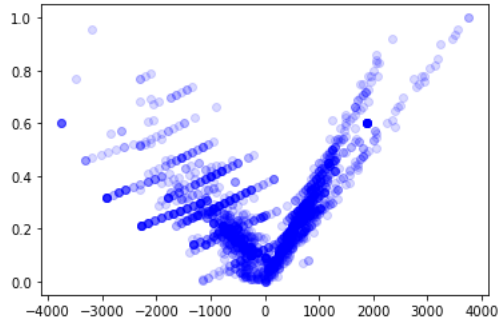
```
tmp_list = []
for subject in range(0,len(profit_loss_list)):
    for choice in range(0,4):
        tmp_list.append([profit_loss_list[subject][choice],chosen_count[subject][choice]])
cluster_array = np.array(tmp_list)
```

The above code snippet is used to arrange the data into arrays, to be passed to clustering algorithms at a later stage.

The graph below shows the scatterplot for the proportion of times each choice was picked by a subject and the net profit or loss the same subject made with that choice in total.

```
plt.scatter(
   cluster_array[:, 0], cluster_array[:, 1],
   c='blue', alpha = 0.15
)
```

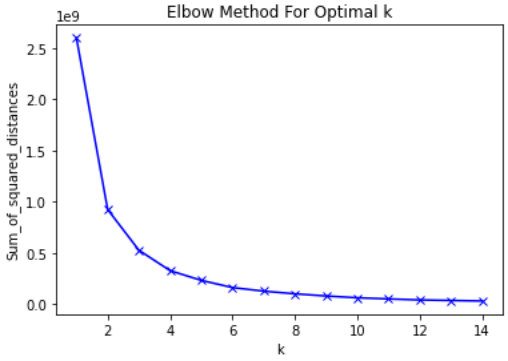
```
<matplotlib.collections.PathCollection at 0x2143fa58430>
```




## Using Elbow Method to Identify Optimal K for K-Means Algorithm

```
Sum_of_squared_distances = []
K = range(1,15)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(cluster_array)
    Sum_of_squared_distances.append(km.inertia_)
```

```
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```




From the above graph, the 'elbow' point appears to be at k = 4.

As there are four choices that the datapoints are classed as, having k = 4 makes the most sense to allow comparisons between predicted clusters and the actual classes of each datapoint.

## Creating a Cluster for each Choice Option using the K-Means Algorithm

```
kmeans = KMeans(
    n_clusters=4, init='random',
    n_init=1,
    tol=1e-04, random_state=2
)

y_km = kmeans.fit_predict(cluster_array)
```

```
plt.scatter(
    cluster_array[y_km == 0, 0], cluster_array[y_km == 0, 1],
    s=25, c='lightgreen',
    marker='o', alpha = 0.2,
    label='cluster 1'
)

plt.scatter(
    cluster_array[y_km == 1, 0], cluster_array[y_km == 1, 1],
    s=25, c='orange',
    marker='o', alpha = 0.2,
    label='cluster 2'
)

plt.scatter(
    cluster_array[y_km == 2, 0], cluster_array[y_km == 2, 1],
    s=25, c='yellow',
    marker='o', alpha = 0.2,
    label='cluster 3'
)

plt.scatter(
    cluster_array[y_km == 3, 0], cluster_array[y_km == 3, 1],
    s=25, c='blue',
    marker='o', alpha = 0.2,
    label='cluster 4'
)

# plot the centroids
plt.scatter(
    kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
    s=250, marker='*',
    c='red', edgecolor='black',
    label='centroids'
)
plt.title("The Total Profit/Loss per Subject per Choice")
plt.xlabel("Total Profit/Loss")
plt.ylabel("Number of times the option was chosen")
plt.legend(scatterpoints=1)
```

```
<matplotlib.legend.Legend at 0x21440d7d760>
```


## Creating Same Clusters using the K-Means Algorithm (where maximum iterations is 300)

```
kmeans2 = KMeans(
    n_clusters=4, init='random',
    n_init=1, max_iter = 300,
)

y_km2 = kmeans2.fit_predict(cluster_array)
```

```
# plot the 4 clusters
plt.scatter(
    cluster_array[y_km2 == 0, 0], cluster_array[y_km2 == 0, 1],
    s=25, c='orange',
    marker='o', alpha = 0.2,
    label='cluster 1'
)

plt.scatter(
    cluster_array[y_km2 == 1, 0], cluster_array[y_km2 == 1, 1],
    s=25, c='blue',
    marker='o', alpha = 0.2,
    label='cluster 2'
)

plt.scatter(
    cluster_array[y_km2 == 2, 0], cluster_array[y_km2 == 2, 1],
    s=25, c='lightgreen',
    marker='o', alpha = 0.2,
    label='cluster 3'
)

plt.scatter(
    cluster_array[y_km2 == 3, 0], cluster_array[y_km2 == 3, 1],
    s=25, c='yellow',
    marker='o', alpha = 0.2,
    label='cluster 4'
)

# plot the centroids
plt.scatter(
    kmeans2.cluster_centers_[:, 0], kmeans2.cluster_centers_[:, 1],
    s=250, marker='*',
    c='red', edgecolor='black',
    label='centroids'
)
plt.title("The Total Profit/Loss per Subject per Choice")
plt.xlabel("Total Profit/Loss")
plt.ylabel("Number of times the option was chosen")
plt.legend(scatterpoints=1)
```

```
<matplotlib.legend.Legend at 0x21440e4edc0>
```


## Comparing clustering algorithm to the original choices

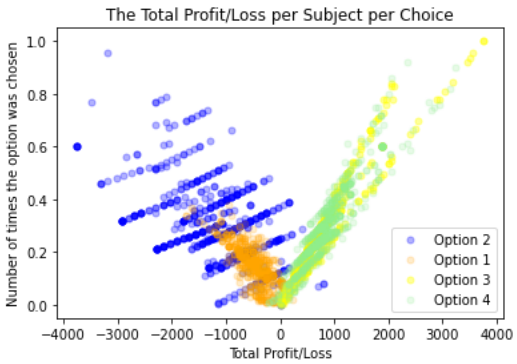
```
option = [1,2,3,4]*(len(profit_loss_list))
option_array = np.array(option)
```

```
option_array
```

```
array([1, 2, 3, ..., 2, 3, 4])
```

```
plt.scatter(
    cluster_array[option_array == 2, 0], cluster_array[option_array == 2, 1],
    s=25, alpha=0.3, c='blue',
    marker='o',
    label='Option 2'
)

plt.scatter(
    cluster_array[option_array == 1, 0], cluster_array[option_array == 1, 1],
    s=25, alpha=0.2, c='orange',
    marker='o',
    label='Option 1'
)

plt.scatter(
    cluster_array[option_array == 3, 0], cluster_array[option_array == 3, 1],
    s=25, alpha=0.4, c='yellow',
    marker='o',
    label='Option 3'
)

plt.scatter(
    cluster_array[option_array == 4, 0], cluster_array[option_array == 4, 1],
    s=25, alpha=0.2, c='lightgreen',
    marker='o',
    label='Option 4'
)

plt.title("The Total Profit/Loss per Subject per Choice")
plt.xlabel("Total Profit/Loss")
plt.ylabel("Number of times the option was chosen")
plt.legend(scatterpoints=1)
```

```
<matplotlib.legend.Legend at 0x21440e97f70>
```




# Conclusions of Cluster Analyses

The clusters produced above are quite accurate in comparison to the real class it belongs to, in this case, the deck options 1, 2, 3 and 4.

There is quite a noticeable split in the graph (at y=0), where there are two sets of data points on the left that are sloped downward, and two sets of datapoints on the right that are sloping upwards. This is an interesting observations as the datasets left and right of y=0 are split by what are considered the 'good decks' and 'bad decks'.

The 'bad decks' are deck 1 and 2, whilst the good decks are deck 3 and 4. Option 1 was the least chosen deck amongst participants, this is explained not only by it being considered a 'bad deck', but also as it is a deck with 'frequent losses', as stated here.

The use of the elbow method helped identify that the optimal number of clusters was 4, which was the original amount of classes, allowing for easier comparison. The cluster with 300 maximum iterations outperforms the other cluster. The k-means algorithm worked quite well in identifying the four classes (four choice options), as shown below.

## Real Cluster Assignments

cluster1_real_values.png

## Predicted Clustered Assignments

# Cluster 2: Average Profit and Loss per Study

This cluster is intended to identify the study based on the total profits and losses made by the participating subject.

## Importing Relevant Packages and Datasets

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from functions import *
```

```
index_95 = pd.DataFrame(pd.read_csv('../data/index_95.csv'))
win_95 = pd.DataFrame(pd.read_csv('../data/wi_95.csv'))
loss_95 = pd.DataFrame(pd.read_csv('../data/lo_95.csv'))

index_100 = pd.DataFrame(pd.read_csv('../data/index_100.csv'))
win_100 = pd.DataFrame(pd.read_csv('../data/wi_100.csv'))
loss_100 = pd.DataFrame(pd.read_csv('../data/lo_100.csv'))

index_150 = pd.DataFrame(pd.read_csv('../data/index_150.csv'))
win_150 = pd.DataFrame(pd.read_csv('../data/wi_150.csv'))
loss_150 = pd.DataFrame(pd.read_csv('../data/lo_150.csv'))
```

## Creating Separate Datasets per Study

```
study_fridberg = index_95
study_hortsmann = index_100.iloc[:162]
study_kjome = index_100.iloc[162:181]
study_maia = index_100.iloc[181:221]
study_steingrover_inprep = index_100.iloc[221:291]
study_premkumar = index_100.iloc[291:316]
study_wood = index_100.iloc[316:469]
study_worthy = index_100.iloc[469:]
study_steingrover_2011 = index_150.iloc[:57]
study_wetzels = index_150.iloc[57:]
```

```
win_fridberg = win_95
win_hortsmann = win_100.iloc[:162]
win_kjome = win_100.iloc[162:181]
win_maia = win_100.iloc[181:221]
win_steingrover_inprep = win_100.iloc[221:291]
win_premkumar = win_100.iloc[291:316]
win_wood = win_100.iloc[316:469]
win_worthy = win_100.iloc[469:]
win_steingrover_2011 = win_150.iloc[:57]
win_wetzels = win_150.iloc[57:]
```

```
loss_fridberg = loss_95
loss_hortsmann = loss_100.iloc[:162]
loss_kjome = loss_100.iloc[162:181]
loss_maia = loss_100.iloc[181:221]
loss_steingrover_inprep = loss_100.iloc[221:291]
loss_premkumar = loss_100.iloc[291:316]
loss_wood = loss_100.iloc[316:469]
loss_worthy = loss_100.iloc[469:]
loss_steingrover_2011 = loss_150.iloc[:57]
loss_wetzels = loss_150.iloc[57:]
```

These datasets were created manually by checking the structure of each dataset and the studies contained within them.

## Creating Arrays to be Clustered

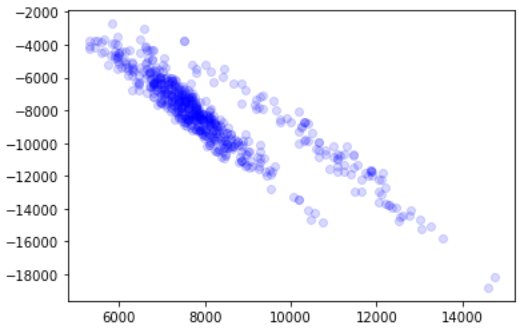
```
win_loss_datasets = [[win_fridberg, loss_fridberg],
                     [win_hortsmann, loss_hortsmann],
                     [win_kjome, loss_kjome],
                     [win_maia, loss_maia],
                     [win_steingrover_inprep, loss_steingrover_inprep],
                     [win_premkumar, loss_premkumar],
                     [win_wood, loss_wood],
                     [win_worthy, loss_worthy],
                     [win_steingrover_2011, loss_steingrover_2011],
                     [win_wetzels, loss_wetzels]]
```

```
profit_and_loss = []
for dataset_pair in win_loss_datasets:
    profit_and_loss = create_avg_profit_loss_list(dataset_pair, profit_and_loss)
```

## Plotting the data by Profits and Losses

```
cluster_array = np.array(profit_and_loss)

plt.scatter(
   cluster_array[:, 0], cluster_array[:, 1],
   c='blue', alpha = 0.15
)
```

```
<matplotlib.collections.PathCollection at 0x21469fe2ac0>
```

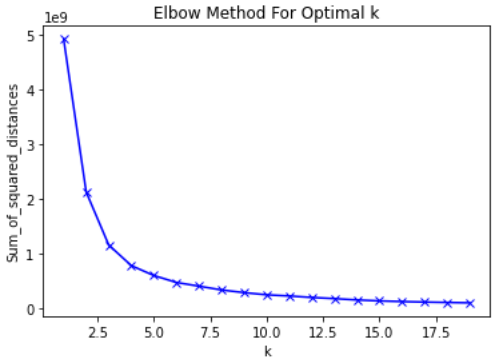



## Using Elbow Method to Identify Optimal K for K-Means Algorithm

```
Sum_of_squared_distances = []
K = range(1,20)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(cluster_array)
    Sum_of_squared_distances.append(km.inertia_)
```

```
C:\Users\laram\AppData\Local\Continuum\anaconda3\envs\ca4015\lib\site-
packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is known to have a memory leak on
Windows with MKL, when there are less chunks than available threads. You can avoid it by
setting the environment variable OMP_NUM_THREADS=3.
  warnings.warn(
```

```
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



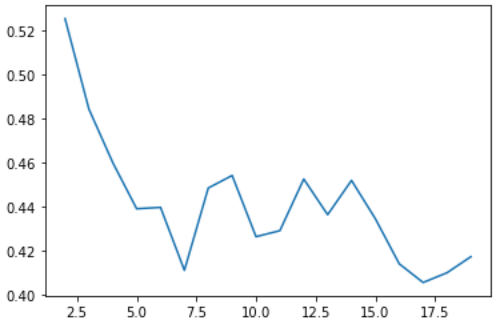

It appears that the elbow point is at approximately k=5. We will use k=5 in our cluster creation.

## Silhouette values in comparison

```
from sklearn.metrics import silhouette_score
silhouette_scores = []
K = range(2,20)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(cluster_array)
    silhouette = silhouette_score(cluster_array, km.labels_, metric='euclidean')
    silhouette_scores.append(silhouette)
```

```
plt.plot(range(2,20),silhouette_scores)
```

```
[<matplotlib.lines.Line2D at 0x2146aa94a30>]
```

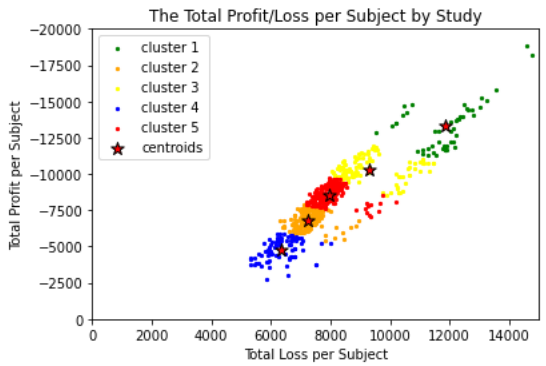


## Creating 5 Clusters Using K-Means Algorithm

```
kmeans_5 = KMeans(
    n_clusters=5, init='random',
    n_init=1, max_iter = 300,
)

y_km_5 = kmeans_5.fit_predict(cluster_array)
```

```
plt.scatter(
    cluster_array[y_km_5 == 0, 0], cluster_array[y_km_5 == 0, 1],
    s=5, c='green',
    marker='o',
    label='cluster 1'
)

plt.scatter(
    cluster_array[y_km_5 == 1, 0], cluster_array[y_km_5 == 1, 1],
    s=5, c='orange',
    marker='o',
    label='cluster 2'
)

plt.scatter(
    cluster_array[y_km_5 == 2, 0], cluster_array[y_km_5 == 2, 1],
    s=5, c='yellow',
    marker='o',
    label='cluster 3'
)

plt.scatter(
    cluster_array[y_km_5 == 3, 0], cluster_array[y_km_5 == 3, 1],
    s=5, c='blue',
    marker='o',
    label='cluster 4'
)

plt.scatter(
    cluster_array[y_km_5 == 4, 0], cluster_array[y_km_5 == 4, 1],
    s=5, c='red',
    marker='o',
    label='cluster 5'
)

# plot the centroids
plt.scatter(
    kmeans_5.cluster_centers_[:, 0], kmeans_5.cluster_centers_[:, 1],
    s=100, marker='*',
    c='red', edgecolor='black',
    label='centroids'
)

plt.title("The Total Profit/Loss per Subject by Study")
plt.xlabel("Total Loss per Subject")
plt.ylabel("Total Profit per Subject")
plt.xlim(0,15000)
plt.ylim(0,-20000)
plt.legend(scatterpoints=1)
```

```
<matplotlib.legend.Legend at 0x2146aafc7f0>
```



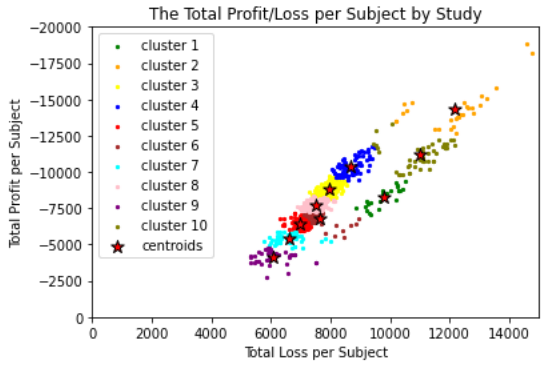

It is not clear just yet if this is an accurate representation of the data groupings. Below we will try out the same clustering technique for k=10, to allow k to equal the same number of classes (studies) that exist.

## Creating a Cluster for each Study using the K-Means Algorithm

```
kmeans = KMeans(
    n_clusters=10, init='random',
    n_init=1, max_iter = 300,
)

y_km = kmeans.fit_predict(cluster_array)
```

```
plt.scatter(
    cluster_array[y_km == 0, 0], cluster_array[y_km == 0, 1],
    s=5, c='green',
    marker='o',
    label='cluster 1'
)

plt.scatter(
    cluster_array[y_km == 1, 0], cluster_array[y_km == 1, 1],
    s=5, c='orange',
    marker='o',
    label='cluster 2'
)

plt.scatter(
    cluster_array[y_km == 2, 0], cluster_array[y_km == 2, 1],
    s=5, c='yellow',
    marker='o',
    label='cluster 3'
)

plt.scatter(
    cluster_array[y_km == 3, 0], cluster_array[y_km == 3, 1],
    s=5, c='blue',
    marker='o',
    label='cluster 4'
)

plt.scatter(
    cluster_array[y_km == 4, 0], cluster_array[y_km == 4, 1],
    s=5, c='red',
    marker='o',
    label='cluster 5'
)

plt.scatter(
    cluster_array[y_km == 5, 0], cluster_array[y_km == 5, 1],
    s=5, c='brown',
    marker='o',
    label='cluster 6'
)

plt.scatter(
    cluster_array[y_km == 6, 0], cluster_array[y_km == 6, 1],
    s=5, c='cyan',
    marker='o',
    label='cluster 7'
)

plt.scatter(
    cluster_array[y_km == 7, 0], cluster_array[y_km == 7, 1],
    s=5, c='pink',
    label='cluster 8'
)

plt.scatter(
    cluster_array[y_km == 8, 0], cluster_array[y_km == 8, 1],
    s=5, c='purple',
    label='cluster 9'
)

plt.scatter(
    cluster_array[y_km == 9, 0], cluster_array[y_km == 9, 1],
    s=5, c='olive',
    label='cluster 10'
)

# plot the centroids
plt.scatter(
    kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
    s=100, marker='*',
    c='red', edgecolor='black',
    label='centroids'
)

plt.title("The Total Profit/Loss per Subject by Study")
plt.xlabel("Total Loss per Subject")
plt.ylabel("Total Profit per Subject")
plt.xlim(0,15000)
plt.ylim(0,-20000)
plt.legend(scatterpoints=1)
```

```
<matplotlib.legend.Legend at 0x2146abc23d0>
```



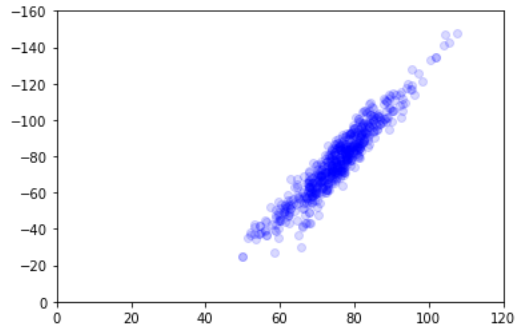

In the above graph, we can see that the total profit per subject or the total loss per subject is done by their overall totals. This can cause an unintended skew in the data, as the subjects who participated in 150-trial studies will naturally have a better opportunity to earn (or lose) more money.

Below we will see the same graph with the data normalised, instead of total profit and loss for all trials, it will be the average profit or loss that the subject made per trial.

## Creating Data that represents the average profit and loss per choice

```
avg_profit_and_loss_per_choice = []
for dataset_pair in win_loss_datasets:
    avg_profit_and_loss_per_choice = create_avg_profit_loss__per_choice_list(dataset_pair,
avg_profit_and_loss_per_choice)
```

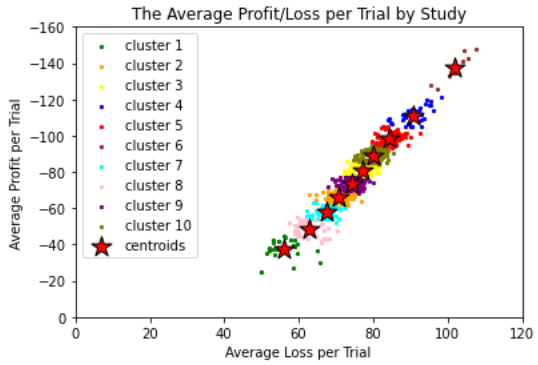
```
cluster2_array = np.array(avg_profit_and_loss_per_choice)

plt.scatter(
   cluster2_array[:, 0], cluster2_array[:, 1],
   c='blue', alpha = 0.15
)
plt.ylim(0,-160)
plt.xlim(0,120)
plt.show()
```




```
kmeans2 = KMeans(
    n_clusters=10, init='random',
    n_init=1, max_iter = 300,
)

y_km2 = kmeans.fit_predict(cluster2_array)
```

```
plt.scatter(
    cluster2_array[y_km2 == 0, 0], cluster2_array[y_km2 == 0, 1],
    s=5, c='green',
    marker='o',
    label='cluster 1'
)

plt.scatter(
    cluster2_array[y_km2 == 1, 0], cluster2_array[y_km2 == 1, 1],
    s=5, c='orange',
    marker='o',
    label='cluster 2'
)

plt.scatter(
    cluster2_array[y_km2 == 2, 0], cluster2_array[y_km2 == 2, 1],
    s=5, c='yellow',
    marker='o',
    label='cluster 3'
)

plt.scatter(
    cluster2_array[y_km2 == 3, 0], cluster2_array[y_km2 == 3, 1],
    s=5, c='blue',
    marker='o',
    label='cluster 4'
)

plt.scatter(
    cluster2_array[y_km2 == 4, 0], cluster2_array[y_km2 == 4, 1],
    s=5, c='red',
    marker='o',
    label='cluster 5'
)

plt.scatter(
    cluster2_array[y_km2 == 5, 0], cluster2_array[y_km2 == 5, 1],
    s=5, c='brown',
    marker='o',
    label='cluster 6'
)

plt.scatter(
    cluster2_array[y_km2 == 6, 0], cluster2_array[y_km2 == 6, 1],
    s=5, c='cyan',
    marker='o',
    label='cluster 7'
)

plt.scatter(
    cluster2_array[y_km2 == 7, 0], cluster2_array[y_km2 == 7, 1],
    s=5, c='pink',
    label='cluster 8'
)

plt.scatter(
    cluster2_array[y_km2 == 8, 0], cluster2_array[y_km2 == 8, 1],
    s=5, c='purple',
    label='cluster 9'
)

plt.scatter(
    cluster2_array[y_km2 == 9, 0], cluster2_array[y_km2 == 9, 1],
    s=5, c='olive',
    label='cluster 10'
)

# plot the centroids
plt.scatter(
    kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
    s=250, marker='*',
    c='red', edgecolor='black',
    label='centroids'
)

plt.title("The Average Profit/Loss per Trial by Study")
plt.xlabel("Average Loss per Trial")
plt.ylabel("Average Profit per Trial")
plt.xlim(0,120)
plt.ylim(0,-160)
plt.legend(scatterpoints=1)
```

```
<matplotlib.legend.Legend at 0x2146ac95ac0>
```

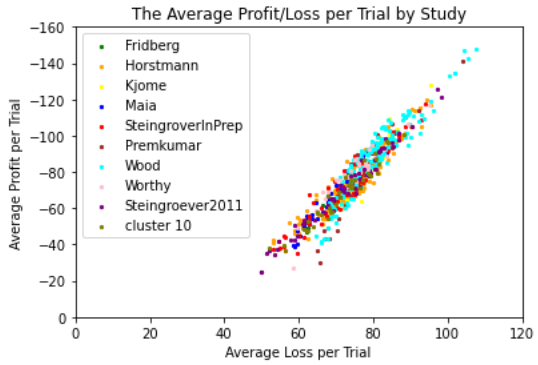



## Comparing Predicted Clusters to the Original Studies

```
study_datasets = [study_fridberg, study_hortsmann,
                  study_kjome, study_maia,
                  study_steingrover_inprep, study_premkumar,
                  study_wood, study_worthy,
                  study_steingrover_2011, study_wetzels]

study_list = []
for dataset in study_datasets:
    for study in dataset.iloc[:,1]:
        study_list.append(study)
study_array = np.array(study_list)
```

```
plt.scatter(
    cluster2_array[study_array == 'Fridberg',0], cluster2_array[study_array == 'Fridberg',1],
    s=5, c='green',
    marker='o',
    label='Fridberg'
)

plt.scatter(
    cluster2_array[study_array == 'Horstmann',0], cluster2_array[study_array == 'Horstmann',1],
    s=5, c='orange',
    marker='o',
    label='Horstmann'
)

plt.scatter(
    cluster2_array[study_array == 'Kjome',0], cluster2_array[study_array == 'Kjome',1],
    s=5, c='yellow',
    marker='o',
    label='Kjome'
)

plt.scatter(
    cluster2_array[study_array == 'Maia',0], cluster2_array[study_array == 'Maia',1],
    s=5, c='blue',
    marker='o',
    label='Maia'
)

plt.scatter(
    cluster2_array[study_array == 'SteingroverInPrep',0], cluster2_array[study_array ==
'SteingroverInPrep',1],
    s=5, c='red',
    marker='o',
    label='SteingroverInPrep'
)

plt.scatter(
    cluster2_array[study_array == 'Premkumar',0], cluster2_array[study_array == 'Premkumar',1],
    s=5, c='brown',
    marker='o',
    label='Premkumar'
)

plt.scatter(
    cluster2_array[study_array == 'Wood',0], cluster2_array[study_array == 'Wood',1],
    s=5, c='cyan',
    marker='o',
    label='Wood'
)

plt.scatter(
    cluster2_array[study_array == 'Worthy',0], cluster2_array[study_array == 'Worthy',1],
    s=5, c='pink',
    label='Worthy'
)

plt.scatter(
    cluster2_array[study_array == 'Steingroever2011',0], cluster2_array[study_array ==
'Steingroever2011',1],
    s=5, c='purple',
    label='Steingroever2011'
)

plt.scatter(
    cluster2_array[study_array == 'Wetzels',0], cluster2_array[study_array == 'Wetzels',1],
    s=5, c='olive',
    label='cluster 10'
)


plt.title("The Average Profit/Loss per Trial by Study")
plt.xlabel("Average Loss per Trial")
plt.ylabel("Average Profit per Trial")
plt.xlim(0,120)
plt.ylim(0,-160)
plt.legend(scatterpoints=1)
```

```
<matplotlib.legend.Legend at 0x2146ad4ad90>
```




The above graph shows no obvious clusters. The choices made by the subjects seem to be generally coherent throughout all of the studies.

This means that the clustering algorithm was limited in it's performance. The k-means clustering algorithm was not the most suitable predictive algortihm to use for this data, as the data is not clustered in its natural form.

This cluster could be expanded more by possibly introducing more features such as the amount of times they chose each deck throughout the trials, this could differentiate the studies and introduce organic clusters as the studies use different payout schemes.

## Real Cluster Assignments

cluster2_real_values.png

## Predicted Cluster Assignments

cluster2_cluster_values.png

# Conclusion

## Data Exploration Conclusion

The data exploration involved in these datasets were related to the payoff schemes and the general choices made by the subjects.

The results from exploring the data based on the payoff schemes used by the studies was in line with the description of the payoff schemes in this papaer. It is useful to visualise the payoff schemes to try and justify differences in subjects' choices.

The results from exploring the choice popularity was quite interesting, as the popularity of each choice seems to change throughout the duration of the trials. The good decks and bad decks begin with a small difference in rewards and losses, that gradually become more drastic. The popularity increasing/decreasing are representative of the decks that become better or worse as the trials go on.

## Clustering Conclusion

The two cluster graphs created are created to see if the choice of decks or the study can be identifed.

The first clustering algorithm was used to identify the deck chosen by a subject based on the net profit/loss they made and the amount of times they chose that deck. This yielded quite an accurate set of clusters that are similar to the original classes they belong to. The clusters identified the bad decks as deck 1 and 2, and the good decks as 3 and 4 (identical to the real datasets).

The second clustering algorithm was used to identify the study that each participant belonged to depending on the profits and loss made by each subject in all of the trials. The results of this clustering technique were not very ipressive, as when compared to the original data, the clusters had correlation. This was due to the nature of the data. The profits and

losses made by each subject was distributed in a specific way irrespective of the study they participated in.

---

By Lara Murphy