Contents

Introduction

The Iowa gambling task is a study that was created to model human decision making. It seeks to explore our ability to learn and how we may react to varying levels of rewards and punishments due to the decisions we have made. The Iowa Gambling involves four decks of cards (A, B, C and D). Participants must choose a card from these four decks. Each deck choice is a single trial and participants complete a task of ninety five, one hundred or one hundred and fifty trials, depending on which study they are part of. The four decks are not all equal, and although there are variations from study to study, the premise of the study is that there are good decks and bad decks. The good decks reward participants for consistently choosing cards from these decks while the bad decks will result in a loss of money over time if they are consistently selected. There are other intricacies from deck to deck such as a variable win and variable loss function that changes over time. The participants are told their task is to maximise their profit by choosing correctly.

Uses of the Iowa Gambling Task

The Iowa Gambling Task has been used to test the decision making of healthy participants vs participants with damage to the prefontal cortex which may impact their decision making. The theory proposed is that healthy participants show an ability to learn from their choices and will tend towards the decks which maximise their earning potential. Conversely the unhealthy participants may not learn from previous choices or may even tend towards more severe punishments from certain bad decks.

Our analysis

We seek to explore the Iowa Gambling task using a clustering algorithm. The data used in this project comes from 617 participants across 10 studies. We explore the data to understand any insights or trends and then further investigate any beliefes we have around potential clustering with the k-means clustering algorithm.

Data Cleaning

We load in the relevant files for inspection and carry out a number of data manipulations, visualisations and processes to understand and analyse our data before proceeding with the clustering algorithm. We also format our data into a single source for our clustering algorithm.

Importing required libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
```

Data Import

Our data is split across 12 files. These are split into trials of length 95, length 100 and length 150. The wi_95 file contains all wins from the 95 trials for each subject, lo_95 contains the losses, index_95 contains which study each subject relates to and choice_95 details what deck each subject chose across each trial.

These files contain data from 617 healthy subjects who have participated in the Iowa Gambling Task. These participants are split accross 10 independant studies with a variety of participants and length of trial:

Study	Participants	Trials
Fridberg et al.	15	95
Horstmannb	162	100
Kjome et al.	19	100
Maia & McClelland	40	100
Premkumar et al.	25	100
Steingroever et al.	70	100
Steingroever et al.	57	150
Wetzels et al.	41	150
Wood et al.	153	100
Worthy et al.	35	100

```
#import raw data into dataframes
wi_95 = pd.read_csv('../data/wi_95.csv')
wi_100 = pd.read_csv('../data/wi_100.csv')
wi_150 = pd.read_csv('../data/wi_150.csv')
lo_95 = pd.read_csv('../data/lo_95.csv')
lo_100 = pd.read_csv('../data/lo_100.csv')
lo_150 = pd.read_csv('../data/lo_150.csv')
index_95 = pd.read_csv('../data/index_95.csv')
index_100 = pd.read_csv('../data/index_100.csv')
index_150 = pd.read_csv('../data/index_150.csv')
choice_95 = pd.read_csv('../data/choice_95.csv')
choice_100 = pd.read_csv('../data/choice_100.csv')
choice_150 = pd.read_csv('../data/choice_150.csv')
```

Inspection of the raw data

We display the wins, losses, index and choices for the trials of length 95 to understand the current format of the data

```
#Display wins
wi_95.head()
```

	Wins_1	Wins_2	Wins_3	Wins_4	Wins_5	Wins_6	Wins_7	Wins_8	Wins_9	Win
Subj_1	100	100	100	100	100	100	100	100	100	
Subj_2	100	100	50	100	100	100	100	100	100	
Subj_3	50	50	50	100	100	100	100	100	100	
Subj_4	50	50	100	100	100	100	100	50	100	
Subj_5	100	100	50	50	50	100	100	100	100	

5 rows × 95 columns

```
#Display losses
lo_95.head()
```

	Losses_1	Losses_2	Losses_3	Losses_4	Losses_5	Losses_6	Losses_7	Losses_8
Subj_1	0	0	0	0	0	0	0	0
Subj_2	0	0	0	0	0	0	0	0
Subj_3	0	0	0	0	0	0	0	-150
Subj_4	0	0	0	0	-150	0	0	0
Subj_5	0	0	0	0	0	0	-150	0

5 rows × 95 columns

```
#Display index
index_95.head()
```

	Subj	Study
0	1	Fridberg
1	2	Fridberg
2	3	Fridberg
3	4	Fridberg
4	5	Fridberg

```
#Display choices
choice_95.head()
```

	Choice_1	Choice_2	Choice_3	Choice_4	Choice_5	Choice_6	Choice_7	Choice_8
Subj_1	2	2	2	2	2	2	2	2
Subj_2	1	2	3	2	2	2	2	2
Subj_3	3	4	3	2	2	1	1	1
Subj_4	4	3	1	1	1	2	2	3
Subj_5	1	2	3	4	3	1	1	2

5 rows × 95 columns

Data manipulations

We want to consolidate the data into a single dataframe with certain features. The first step is to aggregate the choices for all subjects into single columns for each card deck.

```
#Count values for each choice by deck
agg_choice_95 = choice_95.apply(pd.Series.value_counts, axis=1)

agg_choice_100 = choice_100.apply(pd.Series.value_counts, axis=1)

agg_choice_150 = choice_150.apply(pd.Series.value_counts, axis=1)
```

```
#Display aggregated choice data
agg_choice_95.head()
```

	1	2	3	4
Subj_1	12	9	3	71
Subj_2	24	26	12	33
Subj_3	12	35	10	38
Subj_4	11	34	12	38
Subj_5	10	24	15	46

We add columns labelled with the total wins and total losses for each subject calculated from the wins and losses raw data we previously imported

```
#calculate total wins and total losses for each subject
agg_choice_95["tot_win"] = wi_95.sum(axis=1)
agg_choice_95["tot_los"] = lo_95.sum(axis=1)

agg_choice_100["tot_win"] = wi_100.sum(axis=1)
agg_choice_100["tot_los"] = lo_100.sum(axis=1)

agg_choice_150["tot_win"] = wi_150.sum(axis=1)
agg_choice_150["tot_los"] = lo_150.sum(axis=1)

#resetting index for concatination in the next cell
agg_choice_95.reset_index(inplace=True)
agg_choice_100.reset_index(inplace=True)
agg_choice_150.reset_index(inplace=True)
```

We then add the index dataframe so as to know which Study each subject is from which will aid with our analysis and visualisations later on

```
agg_choice_95.head()
```

	index	1	2	3	4	tot_win	tot_los
0	Subj_1	12	9	3	71	5800	-4650
1	Subj_2	24	26	12	33	7250	-7925
2	Subj_3	12	35	10	38	7100	-7850
3	Subj_4	11	34	12	38	7000	-7525
4	Subj_5	10	24	15	46	6450	-6350

```
#Concatenate index to identify the study
final_95 = pd.concat([agg_choice_95, index_95], axis=1)
final_100 = pd.concat([agg_choice_100, index_100], axis=1)
final_150 = pd.concat([agg_choice_150, index_150], axis=1)
```

```
final_95.head()
```

	index	1	2	3	4	tot_win	tot_los	Subj	Study
0	Subj_1	12	9	3	71	5800	-4650	1	Fridberg
1	Subj_2	24	26	12	33	7250	-7925	2	Fridberg
2	Subj_3	12	35	10	38	7100	-7850	3	Fridberg
3	Subj_4	11	34	12	38	7000	-7525	4	Fridberg
4	Subj_5	10	24	15	46	6450	-6350	5	Fridberg

We inspect the new dataframes to see if our aggregation has created any null values

```
#Display summary info for our data types accross the three dataframes we have created
final_95.info()
final_100.info()
final_150.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   index    15 non-null     object
 1   1        15 non-null     int64 
 2   2        15 non-null     int64 
 3   3        15 non-null     int64 
 4   4        15 non-null     int64 
 5   tot_win  15 non-null     int64 
 6   tot_los  15 non-null     int64 
 7   Subj     15 non-null     int64 
 8   Study    15 non-null     object
dtypes: int64(7), object(2)
memory usage: 1.2+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 504 entries, 0 to 503
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   index    504 non-null    object
 1   1        504 non-null    int64 
 2   2        504 non-null    int64 
 3   3        504 non-null    int64 
 4   4        504 non-null    int64 
 5   tot_win  504 non-null    int64 
 6   tot_los  504 non-null    int64 
 7   Subj     504 non-null    int64 
 8   Study    504 non-null    object
dtypes: int64(7), object(2)
memory usage: 35.6+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98 entries, 0 to 97
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   index    98 non-null     object 
 1   1        96 non-null     float64
 2   2        96 non-null     float64
 3   3        98 non-null     float64
 4   4        96 non-null     float64
 5   tot_win  98 non-null     int64  
 6   tot_los  98 non-null     int64  
 7   Subj     98 non-null     int64  
 8   Study    98 non-null     object 
dtypes: float64(4), int64(3), object(2)
memory usage: 7.0+ KB
```

This has shown the presence of six null values within the final_150 dataframe. Further inspection of these null values may show us an insight into some participants choices in this task

```
sample = final_150[final_150[1].isnull()]
sample.head(2)
```

	index	1	2	3	4	tot_win	tot_los	Subj	Study
7	Subj_8	NaN	NaN	150.0	NaN	7500	-3750	8	Steingroever2011
56	Subj_57	NaN	NaN	150.0	NaN	7500	-3750	57	Steingroever2011

Note

Interestingly this step has shown that two participants, both from the Steingroever2011 study chose deck 3 for 150 consecutive trials. It would be interesting to know whether this was a tactical choice with prior knowledge of how the Iowa gambling task is set up - i.e. with decks 3 and 4 giving the best returns. This may have also been a tactic to quickly finish the game from a participant who was uninterested in the game itself. Either way this slightly varies from the expected results of the Iowa Gambling task in which we expect subjects to show a variety of choices but ultimately learn which of the decks are more rewarding.

We replace these cells marked as Null with 0 across the data

```
#replace all NaN values with '0'
final_150[1] = final_150[1].fillna(0)
final_150[2] = final_150[2].fillna(0)
final_150[4] = final_150[4].fillna(0)
```

We change all columns to type int from type float

```
#convert float to int
final_150[1] = final_150[1].astype(int)
final_150[2] = final_150[2].astype(int)
final_150[3] = final_150[3].astype(int)
final_150[4] = final_150[4].astype(int)
```

We then bring all three dataframes together into a single dataframe named 'final'. This is comprised of:

- Trials of length 95
- Trials of length 100
- Trials of length 150

We can then do manipulations across all of this data all at once

```
#creation of a single dataframe with all 617 subjects accross all studies
temp = final_95.append(final_100)
final = temp.append(final_150)
final
```

```
       index    1    2    3    4  tot_win  tot_los  Subj     Study
 0    Subj_1   12    9    3   71     5800    -4650     1  Fridberg
 1    Subj_2   24   26   12   33     7250    -7925     2  Fridberg
 2    Subj_3   12   35   10   38     7100    -7850     3  Fridberg
 3    Subj_4   11   34   12   38     7000    -7525     4  Fridberg
 4    Subj_5   10   24   15   46     6450    -6350     5  Fridberg
...      ...  ...  ...  ...  ...      ...      ...   ...       ...
93   Subj_94   24   69   13   44    12150   -11850    94   Wetzels
94   Subj_95    5   31   46   68     9300    -7150    95   Wetzels
95   Subj_96   18   19   37   76     9350    -7900    96   Wetzels
96   Subj_97   25   30   44   51    10250    -9050    97   Wetzels
97   Subj_98   11  104    6   29    13250   -15050    98   Wetzels
617 rows × 9 columns
```

As can be seen from the above output - the dataframe contains 617 rows (one for each of the the participants). However we dont have a unique identifier for each subject. We also have the issue that there is Subject 1 for each of the three length trials data. We add a column with a unique ID for each of the 617 participants to overcome this issue. We also calculate the balance for each participant after the completion of their trials which is the total wins added to the total losses.
\[wins + losses = balance\]

```
#calculate balance and add Unique ID for participants
final['Unique_ID'] = (np.arange(len(final))+1)
final["balance"] = final["tot_win"] + final["tot_los"]

final = final.rename(columns={1:'Deck_A', 2:'Deck_B', 3:'Deck_C', 4:'Deck_D'})
final
```

	index	Deck_A	Deck_B	Deck_C	Deck_D	tot_win	tot_los	Subj	Study	Unique_
0	Subj_1	12	9	3	71	5800	-4650	1	Fridberg	
1	Subj_2	24	26	12	33	7250	-7925	2	Fridberg	
2	Subj_3	12	35	10	38	7100	-7850	3	Fridberg	
3	Subj_4	11	34	12	38	7000	-7525	4	Fridberg	
4	Subj_5	10	24	15	46	6450	-6350	5	Fridberg	
...	...	...	...	...	...	...	...	...	...	
93	Subj_94	24	69	13	44	12150	-11850	94	Wetzels	6
94	Subj_95	5	31	46	68	9300	-7150	95	Wetzels	6
95	Subj_96	18	19	37	76	9350	-7900	96	Wetzels	6
96	Subj_97	25	30	44	51	10250	-9050	97	Wetzels	6
97	Subj_98	11	104	6	29	13250	-15050	98	Wetzels	6

617 rows × 11 columns

Add Payoff to Dataframe

We are interested in the payoffs used in the various trials for the Iowa Gambling task study. There were three different kinds of payoff used in all the studies.

Payoff 1

This scheme gives +250 result from 10 choices of the good decks (C&D) while giving a -250 result from 10 choices of the bad decks (A&B). Payoff 1 also has a feature in which decks A&C result in frequent losses while decks B&D result in infrequent losses. Another feature of payoff 1 is that deck C has varialble loss between -25, -50 or -75. The final feature of Payoff 1 is that wins and losses are in a fixed sequence

Payoff 2

Payoff 2 is a variant on the first payoff with a number of changes. The first is that the loss from Deck C is constant at -50. The second change involves a random sequence of wins and losses through the trials.

Payoff 3

Payoff 3 is another variety which still contains the base feature of bad decks A&B and good decks C&D with A&C having frequent losses and B&D having infrequent losses. This scheme differs by having decks that change every 10 trials. This means the bad decks (A&B) produce a -250 result from 10 trials at the start but this decreases by 150 for every 10 trials resulting in a final result of -1000 for 10 card chosen fromn the bad decks. Similarly the good decks start with a +250 result and increase by 25 with every 10 trials finishing with a reward of 375 for every 10 trials. This means both choices are magnified as the trials progress but also the punishment for choosing a bad deck increases far more than the reward for choosing a good deck. The final feature of payoff scheme 3 is that there is a difference in wins between decks and the sequence of wins and losses is fixed.

This variety in reward/punishment from payoff to payoff is an interesting factor in the outcomes of our project and so we will further explore how much of an impact this variety has within our clustering analysis.

```
#payoff data
data = [['Fridberg', 1],['Horstmann', 2],['Kjome', 3],['Maia', 1],['SteingroverInPrep',
2],['Premkumar', 3],['Wood', 3],['Worthy', 1],['Steingroever2011', 2],['Wetzels', 2]]

# Create the pandas DataFrame
payoff = pd.DataFrame(data, columns = ['Study', 'Payoff'])

# print dataframe.
payoff
```

	Study	Payoff
0	Fridberg	1
1	Horstmann	2
2	Kjome	3
3	Maia	1
4	SteingroverInPrep	2
5	Premkumar	3
6	Wood	3
7	Worthy	1
8	Steingroever2011	2
9	Wetzels	2

We join the final dataframe with the dataframe containing the payoff values

```
final = final.join(payoff.set_index('Study'), on='Study')
```

Export data

This final dataframe is in the appropriate format for clustering and for some initial visualisations and analysis. We export the data for use in the clustering notebook and then proceed with our exploration.

```
final.to_csv('../data/cleaned_data.csv')
```

Exploration of the data

We want to further understand the data set to see if there are any insights or trends that we can explore during our clustering. A number of methods and graphs can help us to understand the make up of this data and what questions we want to answer within this project.

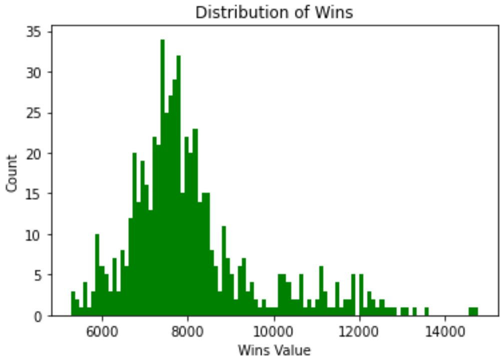
```
final.describe()
```

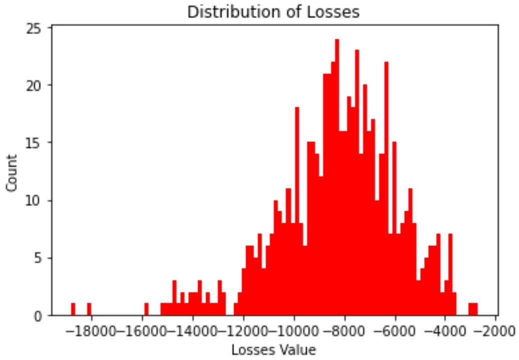
	Deck_A	Deck_B	Deck_C	Deck_D	tot_win	tot_los
count	617.000000	617.000000	617.000000	617.000000	617.000000	617.000000
mean	15.813614	33.388979	26.320908	32.296596	8087.252836	-8244.084279
std	7.999550	17.599469	21.428470	18.170402	1555.720966	2357.633329
min	0.000000	0.000000	1.000000	0.000000	5300.000000	-18800.000000
25%	10.000000	22.000000	14.000000	21.000000	7150.000000	-9550.000000
50%	16.000000	31.000000	21.000000	29.000000	7750.000000	-8100.000000
75%	21.000000	41.000000	30.000000	40.000000	8480.000000	-6650.000000
max	48.000000	143.000000	150.000000	135.000000	14750.000000	-2725.000000

We can see that mean choices for decks B, C & D are higher than Deck A showing a clear trend of participants away from A and towards the good decks C&D. Interestingly the mean total wins is less than the mean average losses and hence participants mean balance was -156.83.

Analysis by Individual

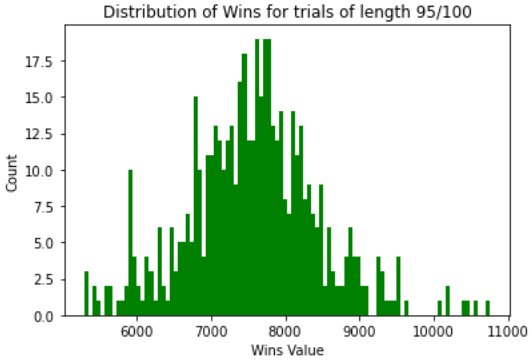
```
#create histogram for wins
plt.hist(final["tot_win"], bins=100, color="green")
plt.title("Distribution of Wins")
plt.xlabel("Wins Value")
plt.ylabel("Count")
plt.show()

#create histogram for losses
plt.hist(final["tot_los"], bins=100, color="red")
plt.title("Distribution of Losses")
plt.xlabel("Losses Value")
plt.ylabel("Count")
plt.show()
```

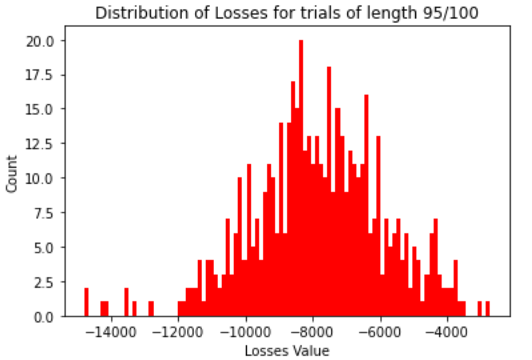




We can see wins has a right skew distribution with a small number of subjects with wins above 10,000. The losses is similarly distributed but with a left skew distribution. The presence of trials of length 150 in this data alongside the data of trials with length 95/100 would be partly the cause of this.

```
#create histogram for wins

to_keep=
['Fridberg','Horstmann','Kjome','Maia','SteingroverInPrep','Premkumar','Wood','Worthy']
filtered_df = final[final['Study'].isin(to_keep)]
plt.hist(filtered_df["tot_win"], bins=100, color="green")
plt.title("Distribution of Wins for trials of length 95/100")
plt.xlabel("Wins Value")
plt.ylabel("Count")
plt.show()

#create histogram for losses
plt.hist(filtered_df["tot_los"], bins=100, color="red")
plt.title("Distribution of Losses for trials of length 95/100")
plt.xlabel("Losses Value")
plt.ylabel("Count")
plt.show()

filtered_df.describe()
```





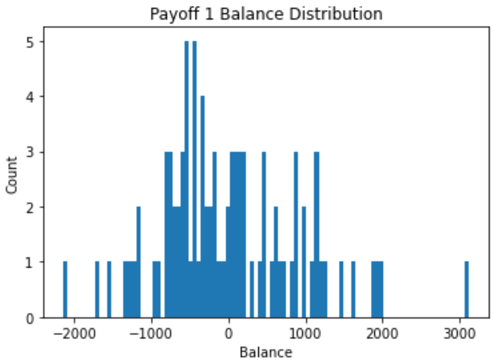
	Deck_A	Deck_B	Deck_C	Deck_D	tot_win	tot_los
count	519.000000	519.000000	519.000000	519.000000	519.000000	519.000000
mean	15.132948	30.901734	23.354528	30.466281	7575.211946	-7832.080925
std	7.125167	13.933238	14.957389	14.430446	881.667949	1960.716200
min	1.000000	1.000000	1.000000	1.000000	5300.000000	-14800.000000
25%	10.000000	22.000000	13.000000	21.000000	7045.000000	-9012.500000
50%	15.000000	30.000000	20.000000	28.000000	7600.000000	-7800.000000
75%	20.000000	39.000000	28.000000	38.000000	8100.000000	-6512.500000
max	38.000000	79.000000	84.000000	92.000000	10750.000000	-2725.000000

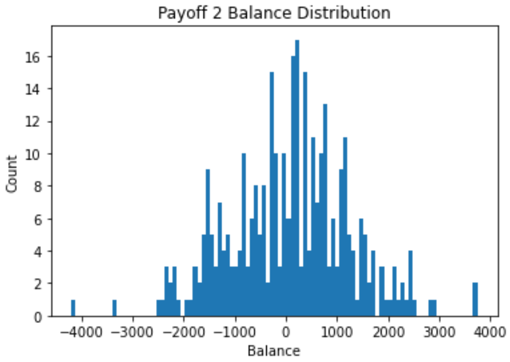
We can see once we remove the trials of length 150 the level of right skew in the wins histogram reduces as does the level of left skew in the losses histogram.

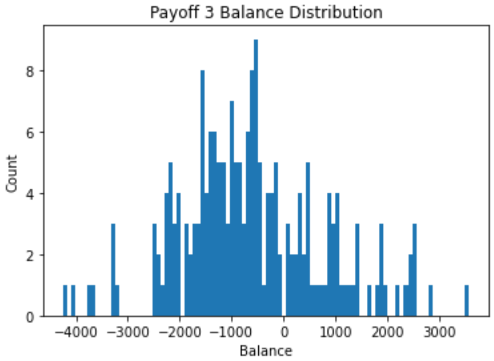
Analysis by Payoff

We now look at the distribution of the balance across the different payoff schemes.

```
i = 1
while i < 4:
    pay1 = final['Payoff'] == i
    temp = final[pay1]
    plt.hist(temp['balance'], bins = 100)
    plt.title("Payoff "+ str(i) + " Balance Distribution")
    plt.xlabel("Balance")
    plt.ylabel('Count')
    plt.show()
    i += 1
    temp.describe()
```

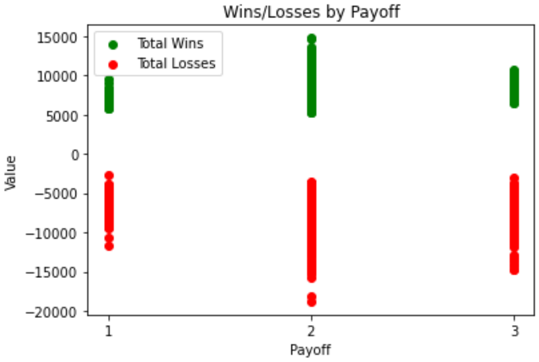






We can see payoff 2 has distribution that centers just above 0 while payoff 1 and 3 have a negative center - payoff 3 showing the worst results. This may be due to the sharp increase in penalty that occurs with every 10 cards chosen in payoff 3.

```
#Show wins and losses by Payoff
plt.scatter(final["Payoff"], final["tot_win"], label = "Total Wins", color="g")
plt.scatter(final["Payoff"], final["tot_los"], label = "Total Losses", color="r")

plt.legend(ncol=1, loc='upper left')
plt.title("Wins/Losses by Payoff")
plt.xlabel("Payoff")
plt.ylabel("Value")
plt.xticks([1, 2, 3])
plt.show()
```

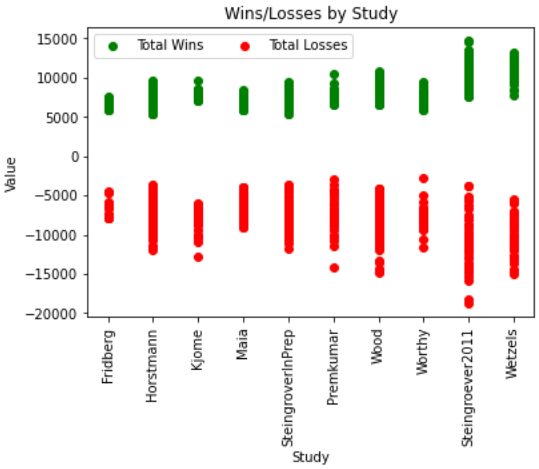


Payoff 2 shows the widest distribution across wins and losses while payoff 3 again shows a very tight distribution of wins but a wide variety of losses due to this evolving reward/punishment system every 10 cards.

Analysis by Study

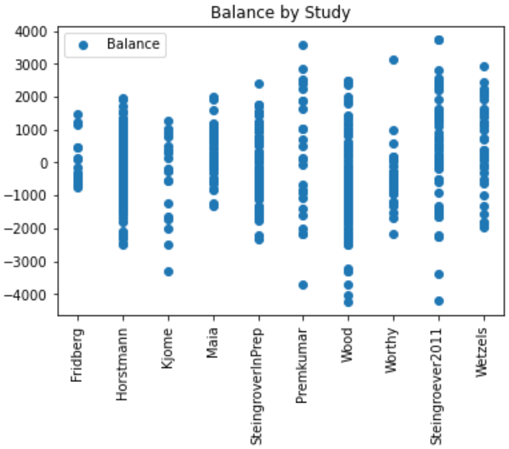
Another theory we have is that the study in which the subject is part of may affect the participants results. We will explore this data with each study in miond to see if any obvious trends emerge. This may also be a useful clustering excercise to see if subjects will cluster with others in their study

```
#Show wins and losses by Study
plt.scatter(final["Study"], final["tot_win"], label = "Total Wins", color="g")
plt.scatter(final["Study"], final["tot_los"], label = "Total Losses", color="r")

plt.xticks(rotation=90)
plt.legend(ncol=2)
plt.title("Wins/Losses by Study")
plt.xlabel("Study")
plt.ylabel("Value")
plt.show()
```



Steingroever2011 and Wetzels, as the only studies with trials of length 150, show the greatest variety in wins and losses across their participants. The Fridberg study group shows limited wins but limited losses also. This group has 15 participants so this may be a factor although Kjome shows a wider variety of wins and losses with 19 participants.

```
#show balance by study
plt.scatter(final["Study"], final["balance"], label = "Balance")
plt.xticks(rotation=90)
plt.legend()
plt.title("Balance by Study")
plt.show()
```

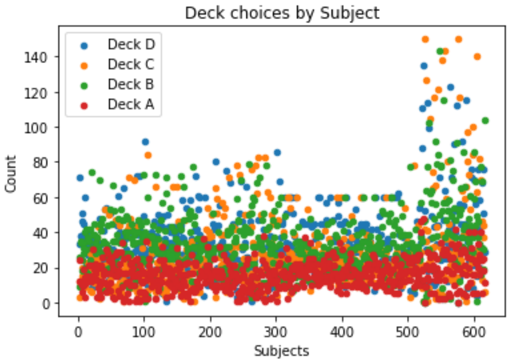


Steingroever2011, Wood and Premkumar show the widest variety in terms of positive and negative balance of its participants. Fridberg and Kjome again vary despite their similar small group size. We can also see that the two participants from Steingroever2011 which chose deck 3 150 times had the greatest profit of 3,750 amongst all participants.

Analysis of deck choices

One of the features we have included in our dataframe is the number of choices of each deck. We know there are good decks and bad decks with varying rewards and punishments so this will definitely be a feature that will affect our clustering algorithm.

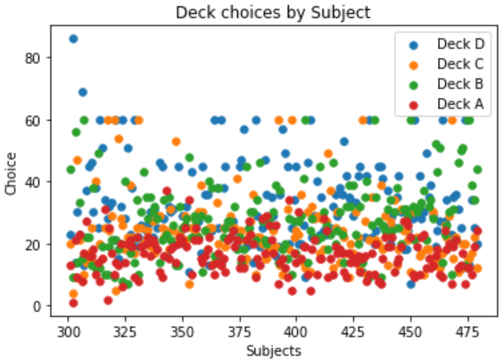
```
plt.scatter(final["Unique_ID"], final['Deck_D'], label = "Deck D", s=20)
plt.scatter(final["Unique_ID"], final['Deck_C'], label = "Deck C", s=20)
plt.scatter(final["Unique_ID"], final['Deck_B'], label = "Deck B", s=20)
plt.scatter(final["Unique_ID"], final['Deck_A'], label = "Deck A", s=20)


plt.title("Deck choices by Subject")
plt.xlabel("Subjects")
plt.ylabel("Count")
plt.legend()
plt.show()
```



We can see above the tendency of participants to choice Decks 2/3/4 more than 60 times where deck 1 was never chosen this many times by any participant. We can also see the trend of a number of participants who have chose the same deck close to 100% of the trials. The large increase in choices from Subject 500 onwards is due to the trials of length 150.

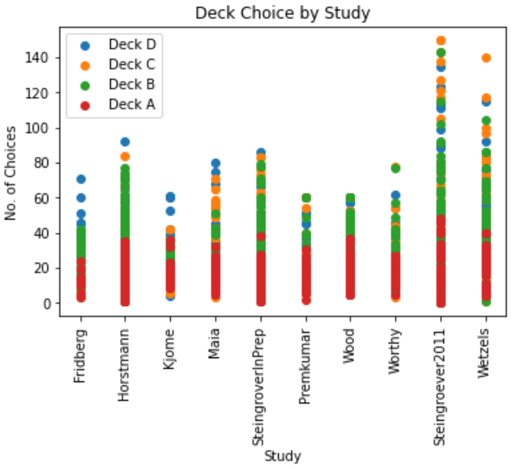
There seems to be a unnatural distribution of choices between subjects 300-480 with a consistent number of choices being exactly 60. We will explore this data in more detail:

```
filter1 = final["Unique_ID"]>300
filter2 = final["Unique_ID"]<480
final_zoom = final.where(filter1&filter2, inplace=False)
plt.scatter(final_zoom["Unique_ID"], final_zoom['Deck_D'], label = "Deck D", s=30)
plt.scatter(final_zoom["Unique_ID"], final_zoom['Deck_C'], label = "Deck C", s=30)
plt.scatter(final_zoom["Unique_ID"], final_zoom['Deck_B'], label = "Deck B", s=30)
plt.scatter(final_zoom["Unique_ID"], final_zoom['Deck_A'], label = "Deck A", s=30)

plt.title("Deck choices by Subject")
plt.xlabel("Subjects")
plt.ylabel("Choice")
plt.legend()
plt.show()
```



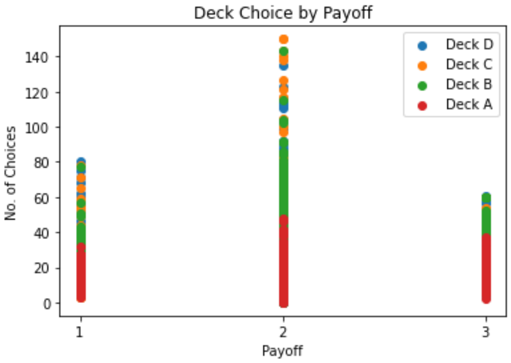
Further inspection on a subset of participants shows an unusual trend to choose a specific deck exactly 60 times. This is across decks 2/3/4 but again deck 1 is neglected in this. This may have been a stipulation of the study that no deck was allowed to be chosen more than 60 times.

```
plt.scatter(final["Study"], final['Deck_D'], label = "Deck D")
plt.scatter(final["Study"], final['Deck_C'], label = "Deck C")
plt.scatter(final["Study"], final['Deck_B'], label = "Deck B")
plt.scatter(final["Study"], final['Deck_A'], label = "Deck A")
plt.xticks(rotation=90)
plt.legend()
plt.title("Deck Choice by Study")
plt.xlabel("Study")
plt.ylabel("No. of Choices")
plt.show()
```



Steingroever2011 had the two most successful participants and also had the two highest choices of a single deck with two members selecting deck C 150/150 trials. There were also a number of other participants who chose a single deck more than 100 times in their study.

```
plt.scatter(final["Payoff"], final['Deck_D'], label = "Deck D")
plt.scatter(final["Payoff"], final['Deck_C'], label = "Deck C")
plt.scatter(final["Payoff"], final['Deck_B'], label = "Deck B")
plt.scatter(final["Payoff"], final['Deck_A'], label = "Deck A")
plt.legend()
plt.xticks([1, 2, 3])
plt.title("Deck Choice by Payoff")
plt.xlabel("Payoff")
plt.ylabel("No. of Choices")
plt.show()
```



Interestingly payoff 2 shows a much higher selection of choices compared to payoff 1 or payoff 3. The distribution across these three is clearly quite different and so it will be interesting to see if clustering will indicate this also.

Data Preprocessing, Standardization, and K-means clustering

We take our cleaned and formatted data and perform a number of steps to the data to preprocess and standardize before it is ready to use with our k-means clustering algorithm.

Import libraries

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
import matplotlib
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
from matplotlib.pyplot import figure
```

Import Data

```
clean_data = pd.read_csv('../data/cleaned_data.csv')
```

Preprocessing

Label Encoding/One Hot Encoding

We want to have our data as numerical type for the k-means clustering algorithm. Categorical data such as the column for "Study" pose a problem as the k-means algorithm cannot assign an appropriate value to a string. Two techniques that can solve this problem are label encoding and one hot encoding. Label encoding assigns a numerical value but this does not work for our dataset as it will assign values (1, 2, 3, etc.) which may imply a hierarchy that is not present in our data between each Study. We therefore explore the second option of one hot encoding which creates a new column for each of the 10 studies and then assigns the value 0 if it is not from this study or a 1 if it is. One hot encoding can cause problems when there are many categories but in this case we deemed it appropriate for use.

We will use the label encoding technique to convert categorical variables to numbers and then use the one hot encoding toolkit from scikit-learn to convert these into binary values. Scikit-learn has a useful preprocessing module for this.

Label Encoder

```
# create instance of labelencoder
labelencoder = LabelEncoder()
# Assign numerical values and store in a new column
clean_data['Study_no'] = labelencoder.fit_transform(clean_data['Study'])
```

Display this new column

```
clean_data
```

	Unnamed: 0	index	Deck_A	Deck_B	Deck_C	Deck_D	tot_win	tot_los	Subj	Stı
0	0	Subj_1	12	9	3	71	5800	-4650	1	Fridt
1	1	Subj_2	24	26	12	33	7250	-7925	2	Fridt
2	2	Subj_3	12	35	10	38	7100	-7850	3	Fridt
3	3	Subj_4	11	34	12	38	7000	-7525	4	Fridt
4	4	Subj_5	10	24	15	46	6450	-6350	5	Fridt
...	...	...	...	...	...	...	...	...	...	
612	93	Subj_94	24	69	13	44	12150	-11850	94	Wet:
613	94	Subj_95	5	31	46	68	9300	-7150	95	Wet:
614	95	Subj_96	18	19	37	76	9350	-7900	96	Wet:
615	96	Subj_97	25	30	44	51	10250	-9050	97	Wet:
616	97	Subj_98	11	104	6	29	13250	-15050	98	Wet:

617 rows × 14 columns

One Hot Encoding

```
# create instance of one-hot-encoder
enc = OneHotEncoder(handle_unknown='ignore')

# passing on Study_no column values
enc_df = pd.DataFrame(enc.fit_transform(clean_data[['Study_no']]).toarray())

# rename columns as Study names
enc_df = enc_df.rename(columns={0: 'Fridberg', 1: 'Horstmann', 2: 'Kjome', 3: 'Maia', 6:
'SteingroverInPrep', 4: 'Premkumar', 8: 'Wood', 9: 'Worthy', 5: 'Steingroever2011', 7:
'Wetzels'})

# reset index before concatenation
clean_data.reset_index(inplace=True)
enc = pd.concat([clean_data, enc_df], axis=1)
enc
```

	level_0	Unnamed: 0	index	Deck_A	Deck_B	Deck_C	Deck_D	tot_win	tot_los	Su
0	0	0	Subj_1	12	9	3	71	5800	-4650	
1	1	1	Subj_2	24	26	12	33	7250	-7925	
2	2	2	Subj_3	12	35	10	38	7100	-7850	
3	3	3	Subj_4	11	34	12	38	7000	-7525	
4	4	4	Subj_5	10	24	15	46	6450	-6350	
...	...	...	...	...	...	...	...	...	...	
612	612	93	Subj_94	24	69	13	44	12150	-11850	
613	613	94	Subj_95	5	31	46	68	9300	-7150	
614	614	95	Subj_96	18	19	37	76	9350	-7900	
615	615	96	Subj_97	25	30	44	51	10250	-9050	
616	616	97	Subj_98	11	104	6	29	13250	-15050	

617 rows × 25 columns

Drop columns

```
df = enc.drop(columns=['level_0', 'index', 'Subj', 'Study', 'Study_no', 'Unnamed: 0',
"Unique_ID", "balance"])
df
```

	Deck_A	Deck_B	Deck_C	Deck_D	tot_win	tot_los	Payoff	Fridberg	Horstmann
0	12	9	3	71	5800	-4650	1	1.0	0.0
1	24	26	12	33	7250	-7925	1	1.0	0.0
2	12	35	10	38	7100	-7850	1	1.0	0.0
3	11	34	12	38	7000	-7525	1	1.0	0.0
4	10	24	15	46	6450	-6350	1	1.0	0.0
...	...	...	...	...	...	...	...	...	...
612	24	69	13	44	12150	-11850	2	0.0	0.0
613	5	31	46	68	9300	-7150	2	0.0	0.0
614	18	19	37	76	9350	-7900	2	0.0	0.0
615	25	30	44	51	10250	-9050	2	0.0	0.0
616	11	104	6	29	13250	-15050	2	0.0	0.0

617 rows × 17 columns

Standardization

Generally speaking, learning algorithms perform better with standardized data. In our case there is a large variety in the 0 and 1 values for the studies and the total win and total loss values which vary from 14,750 to -2,725 and so we have chosen to standardize the whole data set. Scitkit-learn again offers a way to do this using the preprocessing module.

```
scaler = preprocessing.StandardScaler().fit(df)
X_scaled = scaler.transform(df)
```

```
#rename columns to clearly represent decks
sd = pd.DataFrame(X_scaled, columns=['Deck_A', 'Deck_B', 'Deck_C', 'Deck_D', 'tot_win',
'tot_los', 'Payoff', 'Fridberg', 'Horstmann', 'Kjome', 'Maia', 'Premkumar',
'Steingroever2011', 'SteingroverInPrep', 'Wetzels', 'Wood', 'Worthy'])
sd
```

	Deck_A	Deck_B	Deck_C	Deck_D	tot_win	tot_los	Payoff	Fridk
0	-0.477115	-1.386904	-1.089197	2.131753	-1.471413	1.525683	-1.778972	6.335
1	1.024186	-0.420182	-0.668854	0.038743	-0.538613	0.135451	-1.778972	6.335
2	-0.477115	0.091612	-0.762264	0.314139	-0.635110	0.167288	-1.778972	6.335
3	-0.602224	0.034746	-0.668854	0.314139	-0.699441	0.305250	-1.778972	6.335
4	-0.727332	-0.533914	-0.528740	0.754773	-1.053262	0.804036	-1.778972	6.335
...	...	...	...	...	...	...	...	
612	1.024186	2.025056	-0.622150	0.644614	2.613607	-1.530705	-0.262914	-0.157
613	-1.352875	-0.135852	0.919107	1.966515	0.780173	0.464437	-0.262914	-0.157
614	0.273535	-0.818244	0.498764	2.407149	0.812338	0.146063	-0.262914	-0.157
615	1.149295	-0.192718	0.825698	1.030169	1.391318	-0.342110	-0.262914	-0.157
616	-0.602224	4.015366	-0.949083	-0.181574	3.321249	-2.889100	-0.262914	-0.157

617 rows × 17 columns

K- means clustering

Clustering involves grouping data points into groups based on their distance to the centroid of each cluster. It is an unsupervised technique using patterns within the data to decide upon the chosen groups. Within k-means, the k represents the number of clusters we require. We choose this value for k and so the best way to determine the most appropriate value of k is to run experiments for different values of k and then see which results in the least error.

K means steps:

- The value of k must be decided
- k random points are selected as the initial centroids
- Each data point is assigned to a cluster based on their distance from the nearest initial centroid.
- A new centroid is then calculated from each cluster
- This process is iterated over until the data points settle in their appropriate cluster

This measure of distance d is the euclidean distance given by:
\[d = √((x1-y1)^2 + (x2-y2)^2)\]

PCA dimensionality reduction

Principal component analysis is a method whereby we can reduce the dimensions used to represent our data. This can help with the curse of dimensionality which can often make clustering algorithms challenging with so many features. PCA helps us by taking all of these features and represents the data within a new set of features (the number of which we can decide). This can often help with the visualisation of cluster as we can reduce the features to 2 components which will healp to plot our points and therefore visualise the clustering algorithm and how well it represents our data.

```
#select how many features we want
pca = PCA(2)
#transform data
df = pca.fit_transform(sd)
```

Choosing the value for k

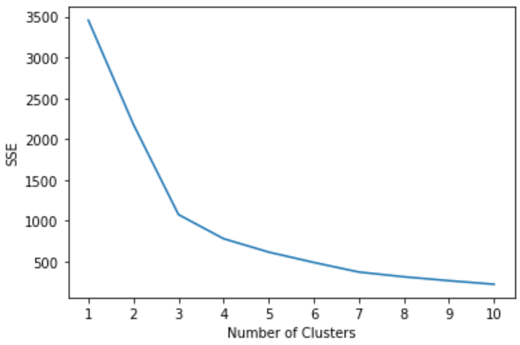
The first step in the k-means algorithm involves calculating what is the most appropriate value for k. We will explore the elbow method and the silhoutte coefficient to determine our value for k within this project.

From our data exploration thus far we are interested in whether the data should be partitioned in 10 groups (k=10) for each study we have, or whether clustering would be best in 3 groups(k=3) to reflect the 3 payoff schemes we have seen.

Elbow Method

The elbow method involves calculating the sum of squared error (SSE) and deciding at what point the modelling of the data is most appropriate. Often we can add further clusters without much gain in SSE and so we must determine at which point any further returns are no longer worthwhile.
\[SSE = \sum_{i=1}^n (y_i-ŷ_i)^2\]

```python
kmeans_kwargs = {
    "init": "random",
    "n_init": 10,
    "max_iter": 300,
    "random_state": 42,
}

# A list holds the SSE values for each k
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(df)
    sse.append(kmeans.inertia_)

plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```



k=3 appears to be a good fit using the elbow method and SSE

Silhouette Coefficient

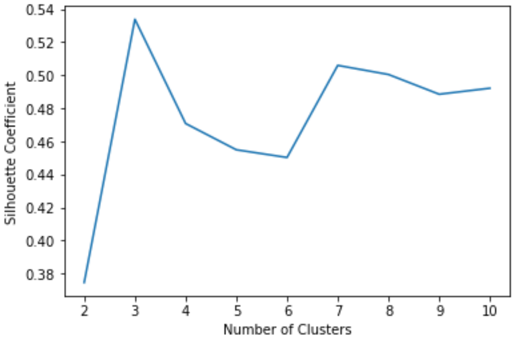
The Silhouette coefficient is another measure of clustering and ranges between -1 and 1

This value is calculated by the following equation:
\[(y-x)/max(x,y)\]
Where:

x = average distance between each point in a cluster

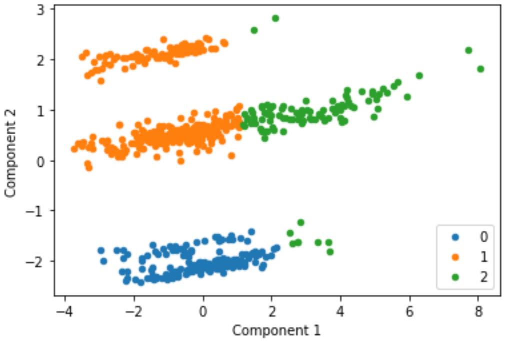
y = average distance between all clusters

```
# A list holds the silhouette coefficients for each k
silhouette_coefficients = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(df)
    score = silhouette_score(df, kmeans.labels_)
    silhouette_coefficients.append(score)

plt.plot(range(2, 11), silhouette_coefficients)
plt.xticks(range(2, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
plt.show()
```



Again we see that k=3 returns the highest value for the silhouette coefficient confirming that payoff scheme may be the most defining feature in the data for the outcome of the Iowa Gambling task. We will proceed to cluster with k=3.

```
kmeans = KMeans(n_clusters= 3)

#predict the labels of clusters.
label = kmeans.fit_predict(df)

#Getting unique labels
u_labels = np.unique(label)

#plotting the results:
for i in u_labels:
    plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i, s=20)
plt.legend()
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.show()
```



Interestingly this instance of k-means has split the data into three clusters which may not intuitively be the ideal clusters. This can often be a limiting factor when using k-means. We now add the Payoff labels back onto the data to visuakise the same points with each payoff scheme identified by a different color.

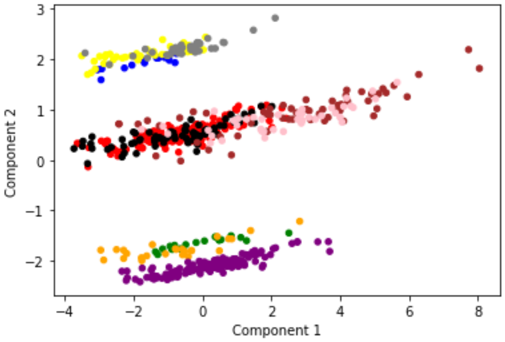
```
tmp = pd.DataFrame(df, columns=['Component_1', 'Component_2'])


original_labels = pd.concat([tmp, clean_data[["Study", "Payoff"]]], axis=1)
```

```
original_labels
```

	Component_1	Component_2	Study	Payoff
0	-2.954758	1.588772	Fridberg	1
1	-0.803481	1.928634	Fridberg	1
2	-1.099105	2.030063	Fridberg	1
3	-1.279225	2.022929	Fridberg	1
4	-2.055069	1.879925	Fridberg	1
...	...	...	...	...
612	4.101098	1.061375	Wetzels	2
613	0.162405	0.589999	Wetzels	2
614	0.561677	0.383470	Wetzels	2
615	1.692721	0.610973	Wetzels	2
616	5.655661	1.538607	Wetzels	2

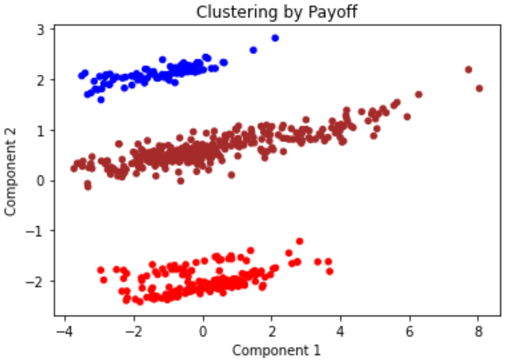
617 rows × 4 columns

```
colors = {'Fridberg': 'blue','Horstmann': 'red','Kjome': 'green','Maia':
'yellow','SteingroverInPrep': 'black','Premkumar': 'orange','Wood': 'purple','Worthy':
'grey','Steingroever2011': 'brown','Wetzels': 'pink'}
#for i in range(618):
figure(figsize=(6, 4))
plt.scatter(original_labels['Component_1'], original_labels["Component_2"], c =
original_labels["Study"].map(colors), s=20)
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.show()

for k in colors:
    print(k + ": " + colors[k])
```



```
Fridberg: blue
Horstmann: red
Kjome: green
Maia: yellow
SteingroverInPrep: black
Premkumar: orange
Wood: purple
Worthy: grey
Steingroever2011: brown
Wetzels: pink
```

```
colors = {1: 'blue',2: 'brown',3:'red'}
#for i in range(618):
figure(figsize=(6, 4))
plt.scatter(original_labels["Component_1"], original_labels["Component_2"],
c=original_labels["Payoff"].map(colors), s=20)
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.title("Clustering by Payoff")
plt.show()
```

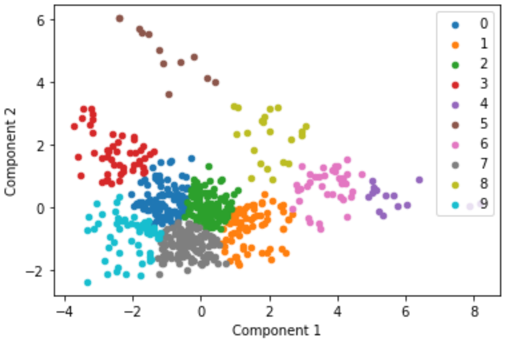


We can see that the payoff data clearly has a large impact on our clusters that we have created. All data points have been assigned to the same cluster as other data points in their payoff.

k=10 (without payoff data)

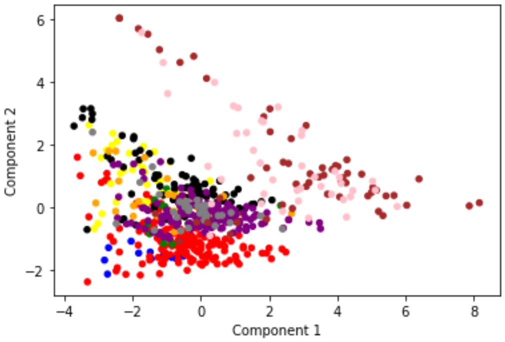
To explore the possibility that our data is impacted by the study in which it was recorded we drop the payoff column and then reduce the dimensionality before computing our clusters with k=10

```
#drop payoff column
w_out_payoff = sd.drop(columns=["Payoff"])

pca = PCA(2)
w_out = pca.fit_transform(w_out_payoff)

kmeans = KMeans(n_clusters= 10)

#predict the labels of clusters.
label = kmeans.fit_predict(w_out)

#Getting unique labels
u_labels = np.unique(label)

#plotting the results:
for i in u_labels:
    plt.scatter(w_out[label == i , 0] , w_out[label == i , 1] , label = i, s=20)
plt.legend()
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.show()
```



We then plot the same data points but clustering by study using a unique color for each

```
t = pd.DataFrame(w_out, columns=['Component_1', 'Component_2'])


t_labels = pd.concat([t, clean_data["Study"]], axis=1)

colors = {'Fridberg': 'blue','Horstmann': 'red','Kjome': 'green','Maia':
'yellow','SteingroverInPrep': 'black','Premkumar': 'orange','Wood': 'purple','Worthy':
'grey','Steingroever2011': 'brown','Wetzels': 'pink'}
#for i in range(618):
figure(figsize=(6, 4))
plt.scatter(t_labels['Component_1'], t_labels["Component_2"], c =
t_labels["Study"].map(colors), s=20)
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.show()

#show study-colors
for k in colors:
    print(k + ": " + colors[k])
```



```
Fridberg: blue
Horstmann: red
Kjome: green
Maia: yellow
SteingroverInPrep: black
Premkumar: orange
Wood: purple
Worthy: grey
Steingroever2011: brown
Wetzels: pink
```

Analysis

The two features we have identified, Payoff scheme and study, have clearly impacted the results of this clustering algorithm. The clusters for payoff show how strongly weighted this was within the clustering algorithm. We can agree from this that the appropriate value for k is 3 and that using k=10 can also tell us an interesting insight into how using study as a features may affect the final clustering result.

Privacy Preserving k-means

Data privacy is an extremely sensitive topic which requires a selection of methods/techniques when processing or using individuals data. Some new approaches to this involve the creation of synthetic data or the move away from sharing any individuals data at all to a single central data source. This second approach involves processing data at source and sharing only the key information centrally. This can mean processing data within a device, or at a central source (i.e. within a single study) and then sharing only the key information that is needed about that data. This section will discuss how information could be shared in this case between studies to gain a central cluster without passing on any individuals data from one study to another.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
import matplotlib
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
from matplotlib.pyplot import figure
import array
```

```
data = pd.read_csv('../data/cleaned_data.csv')
```

```
data
```

	Unnamed: 0	index	Deck_A	Deck_B	Deck_C	Deck_D	tot_win	tot_los	Subj	St
0	0	Subj_1	12	9	3	71	5800	-4650	1	Fridb
1	1	Subj_2	24	26	12	33	7250	-7925	2	Fridb
2	2	Subj_3	12	35	10	38	7100	-7850	3	Fridb
3	3	Subj_4	11	34	12	38	7000	-7525	4	Fridb
4	4	Subj_5	10	24	15	46	6450	-6350	5	Fridb
...	...	...	...	...	...	...	...	...	...	
612	93	Subj_94	24	69	13	44	12150	-11850	94	Wetz
613	94	Subj_95	5	31	46	68	9300	-7150	95	Wetz
614	95	Subj_96	18	19	37	76	9350	-7900	96	Wetz
615	96	Subj_97	25	30	44	51	10250	-9050	97	Wetz
616	97	Subj_98	11	104	6	29	13250	-15050	98	Wetz

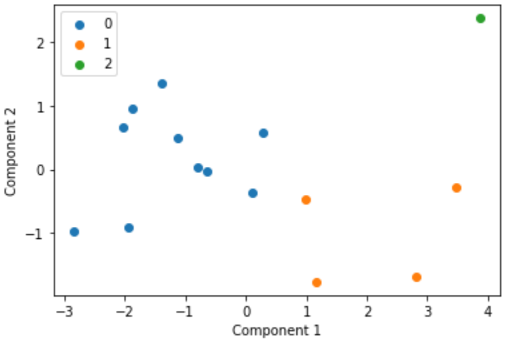
617 rows × 13 columns

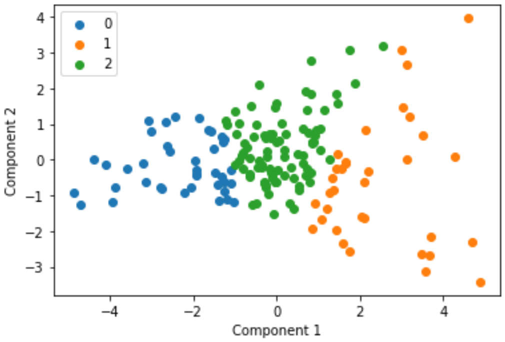
Clustering by study

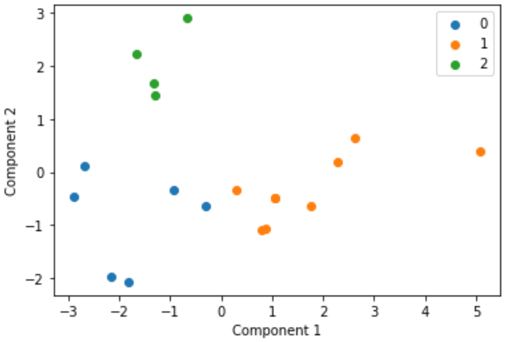
We cluster each study individually with k=3 and display these clusters below

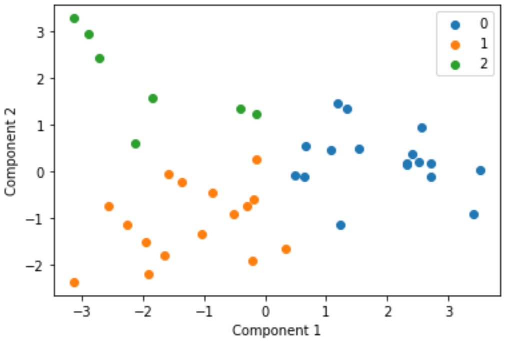
```
Fridberg = data[data['Study']=='Fridberg']
Horstmann = data[data['Study']=='Horstmann']
Kjome = data[data['Study']=='Kjome']
Maia = data[data['Study']=='Maia']
SteingroverInPrep = data[data['Study']=='SteingroverInPrep']
Premkumar = data[data['Study']=='Premkumar']
Wood = data[data['Study']=='Wood']
Worthy = data[data['Study']=='Worthy']
Steingroever2011 = data[data['Study']=='Steingroever2011']
Wetzels = data[data['Study']=='Wetzels']

l = [Fridberg, Horstmann, Kjome, Maia, SteingroverInPrep, Premkumar, Wood, Worthy,
Steingroever2011, Wetzels]
```

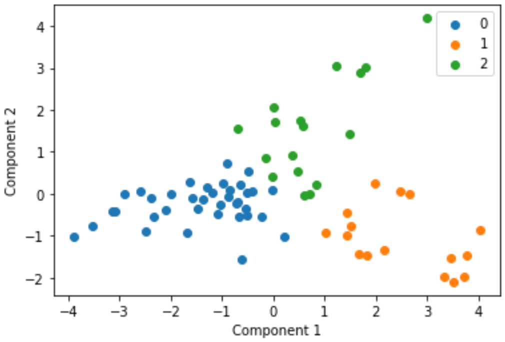
```
clusters = []
labels = []
for s in l:
    df = s.drop(columns=['index', 'Subj', 'Study', 'Unnamed: 0', 'Unique_ID',
'balance'])

    scaler = preprocessing.StandardScaler().fit(df)
    X_scaled = scaler.transform(df)

    #rename columns to clearly represent decks
    sd = pd.DataFrame(X_scaled, columns=['Deck_1', 'Deck_2', 'Deck_3', 'Deck_4',
'tot_win', 'tot_los', 'Payoff'])
    sd

    pca = PCA(2)
    df = pca.fit_transform(sd)

    #w_out = pd.DataFrame(df, columns=['Component_1', 'Component_2'])
    #w_out_payoff = sd.drop(columns=["Payoff"])

    #w_out = pca.fit_transform(w_out_payoff)

    kmeans = KMeans(n_clusters= 3)

    #predict the labels of clusters.
    label = kmeans.fit_predict(df)
    for v in label:
        labels.append(v)

    #Getting unique labels
    u_labels = np.unique(label)

    centroids = kmeans.cluster_centers_
    clusters.append(centroids)

    #plotting the results:
    for i in u_labels:
        plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
    plt.legend()
    plt.xlabel("Component 1")
    plt.ylabel("Component 2")
    plt.show()
```

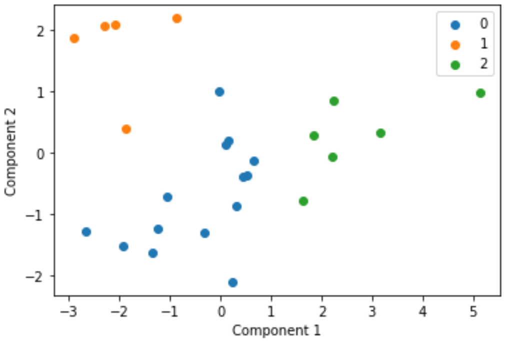

0
1
2
Component 2
Component 1
2
1
0
−1
−3
−2
−1
0
1
2
3
4

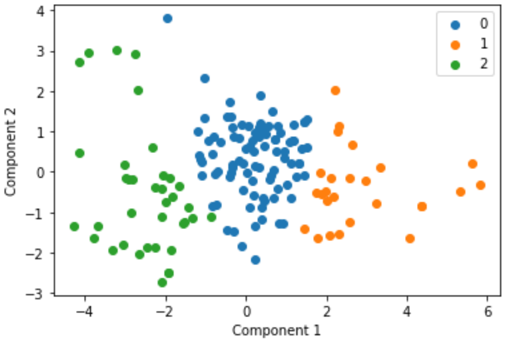

0
1
2
Component 2
Component 1
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4

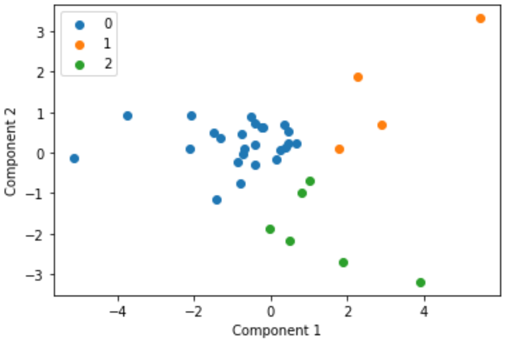

0
1
2
Component 2
Component 1
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
4
5

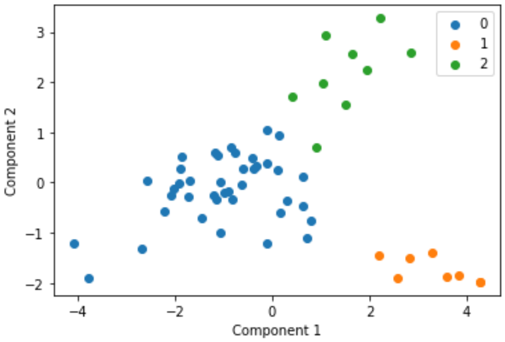

0
1
2
Component 2
Component 1
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

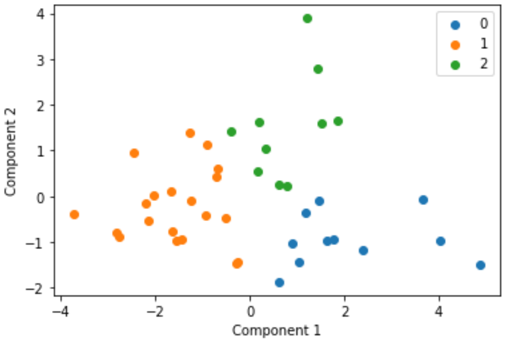

0
1
2
Component 2
Component 1
4
3
2
1
0
−1
−2
−4
−3
−2
−1
0
1
2
3
4







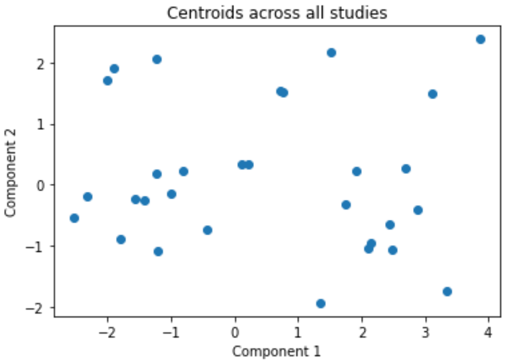




Centroid collection

We make a list called clusters above, with all of the centroids for each of the studies and plot these all together in the next cell. This gives an indication of each studies data distribution without providing any raw data centrally.

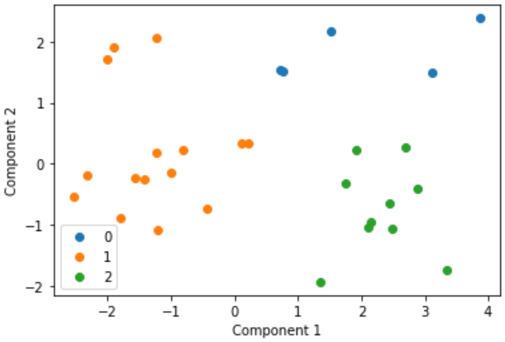
```
arr = np.vstack(clusters)
plt.scatter(arr[:,0], arr[:,1])
plt.title("Centroids across all studies")
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.show()
```



Final clustering

We then cluster these centroids to see what our final distribution is like

```
kmeans = KMeans(n_clusters= 3)

#predict the labels of clusters.
label = kmeans.fit_predict(arr)

#Getting unique labels
u_labels = np.unique(label)


#plotting the results:
for i in u_labels:
    plt.scatter(arr[label == i , 0] , arr[label == i , 1] , label = i)
plt.legend()
#plt.title()
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.show()
```



Analysis

This is an incredibly simplistic and naive method of privacy preserving clustering. The use of centroids causes issues as some centroids will have been created with very few points and others with a large number of data points and hence the clustering will be quite poor in comparison to the original clustering effort. Some more work would need to be done to explore a better way of protecting individuals privacy while allowing us to centrally analyse our data using k-means.

By George Dockrell