Contents

The Iowa Gambling Task

The Iowa Gambling Task is a psychological task thought to simulate real-life decision making. There are 4 decks for participants to choose from. Each time they choose a deck they get either a reward, a penalty or a mixture of both. The aim is to get the most money. Two of the decks contain more penalties and two contain more rewards. It was found that most people tended to stick to the “good” decks after roughly 50 selections.

Decks A and B are known as the “good” decks because they have a positive net outcome whereas decks C and D have a negative net outcome. There are slight differences in these outcomes in each study. There are three different payloads:

1. In Fridberg, Maia, and Worthy, deck C has a variable loss of either −25, −50, or −75.
2. In Horstmann, Streingroever, and Wetzels, they maintain the loss of deck C as a constant of -50.
3. In Kjome, Premkumar, and Wood, the schedules of rewards and losses in such a way that the net outcome of the “bad” decks decreases by 150 every 10 cards and the net outcome of the “good” decks increases by 25 every 10 cards.

My objective was to clean and process the data and perform clustering analysis on it to gain insights into the Iowa Gambling Task and real-life decision making. Another target was to find a way to perform clustering on the data while preserving the privacy of each individual lab.

Data Processing

This is where I cleaned and processed the datasets that were given. I did this in three steps:

1. Data cleaning
2. Merging the Data
3. Aggregating the Data

```
import pandas as pd
import numpy as np
```

Firstly, I read in each dataset and used `pandas.DataFrame.head()` to get an idea of what the datasets looked like.

```
choice_100 = pd.read_csv("./data/choice_100.csv")
choice_150 = pd.read_csv("./data/choice_150.csv")
choice_95 = pd.read_csv("./data/choice_95.csv")
index_100 = pd.read_csv("./data/index_100.csv")
index_150 = pd.read_csv("./data/index_150.csv")
index_95 = pd.read_csv("./data/index_95.csv")
lo_100 = pd.read_csv("./data/lo_100.csv")
lo_150 = pd.read_csv("./data/lo_150.csv")
lo_95 = pd.read_csv("./data/lo_95.csv")
wi_100 = pd.read_csv("./data/wi_100.csv")
wi_150 = pd.read_csv("./data/wi_150.csv")
wi_95 = pd.read_csv("./data/wi_95.csv")
```

```
print(choice_100.head())
print(index_100.head())
print(lo_100.head())
print(wi_100.head())
```

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

        Choice_8  Choice_9  Choice_10  ...  Choice_91  Choice_92  Choice_93  \
Subj_1         2         4          2  ...          1          1          1
Subj_2         2         1          2  ...          4          2          3
Subj_3         4         4          3  ...          3          2          1
Subj_4         2         2          2  ...          4          2          3
Subj_5         1         4          1  ...          2          2          2

        Choice_94  Choice_95  Choice_96  Choice_97  Choice_98  Choice_99  \
Subj_1          2          2          2          4          2          4
Subj_2          4          2          4          4          2          2
Subj_3          4          2          2          2          4          2
Subj_4          4          3          4          1          4          3
Subj_5          2          3          3          3          3          4

        Choice_100
Subj_1           2
Subj_2           4
Subj_3           2
Subj_4           4
Subj_5           4

[5 rows x 100 columns]
   Subj      Study
0     1  Horstmann
1     2  Horstmann
2     3  Horstmann
3     4  Horstmann
4     5  Horstmann
        Losses_1  Losses_2  Losses_3  Losses_4  Losses_5  Losses_6  Losses_7  \
Subj_1      -200      -150         0      -250         0         0         0
Subj_2         0         0         0         0         0         0       -50
Subj_3         0         0       -50      -300         0     -1250         0
Subj_4      -250       -50         0         0      -200         0         0
Subj_5         0         0         0     -1250         0       -50         0

        Losses_8  Losses_9  Losses_10  ...  Losses_91  Losses_92  Losses_93  \
Subj_1         0         0          0  ...          0          0       -350
Subj_2         0         0          0  ...          0          0          0
Subj_3         0         0        -50  ...        -50          0       -200
Subj_4         0         0      -1250  ...          0          0          0
Subj_5         0         0          0  ...          0          0          0

        Losses_94  Losses_95  Losses_96  Losses_97  Losses_98  Losses_99  \
Subj_1          0          0          0          0          0          0
Subj_2       -250      -1250          0          0          0          0
Subj_3          0          0          0          0          0          0
Subj_4          0        -50          0          0          0        -50
Subj_5      -1250          0          0          0        -50          0

        Losses_100
Subj_1       -1250
Subj_2           0
Subj_3           0
Subj_4           0
Subj_5           0

[5 rows x 100 columns]
        Wins_1  Wins_2  Wins_3  Wins_4  Wins_5  Wins_6  Wins_7  Wins_8  \
Subj_1     100     100     100      50      50     100     100     100
Subj_2     100     100      50      50      50     100      50     100
Subj_3      50     100      50     100      50     100      50      50
Subj_4      50      50      50     100     100      50      50     100
Subj_5     100     100     100     100     100      50      50     100

        Wins_9  Wins_10  ...  Wins_91  Wins_92  Wins_93  Wins_94  Wins_95  \
Subj_1      50      100  ...      100      100      100      100      100
Subj_2     100      100  ...       50      100       50       50      100
Subj_3      50       50  ...       50      100      100       50      100
Subj_4     100      100  ...       50      100       50       50       50
Subj_5      50      100  ...      100      100      100      100       50

        Wins_96  Wins_97  Wins_98  Wins_99  Wins_100
Subj_1      100       50      100       50       100
Subj_2       50       50      100      100        50
Subj_3      100      100       50      100       100
Subj_4       50      100       50       50        50
Subj_5       50       50       50       50        50

[5 rows x 100 columns]
```

1. Data Cleaning

My first step was data cleaning. I went through each individual dataframe to see if there were any obvious errors, data type differences or any nulls that needed to be changed.

```
print(choice_95.info(verbose=False))
print(choice_100.info(verbose=False))
print(choice_150.info(verbose=False))
print(index_95.info(verbose=False))
print(index_100.info(verbose=False))
print(index_150.info(verbose=False))
print(lo_95.info(verbose=False))
print(lo_100.info(verbose=False))
print(lo_150.info(verbose=False))
print(wi_95.info(verbose=False))
print(wi_100.info(verbose=False))
print(wi_150.info(verbose=False))
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 15 entries, Subj_1 to Subj_15
Columns: 95 entries, Choice_1 to Choice_95
dtypes: int64(95)
memory usage: 11.2+ KB
None
<class 'pandas.core.frame.DataFrame'>
Index: 504 entries, Subj_1 to Subj_504
Columns: 100 entries, Choice_1 to Choice_100
dtypes: int64(100)
memory usage: 397.7+ KB
None
<class 'pandas.core.frame.DataFrame'>
Index: 98 entries, Subj_1 to Subj_98
Columns: 150 entries, Choice_1 to Choice_150
dtypes: int64(150)
memory usage: 115.6+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Columns: 2 entries, Subj to Study
dtypes: int64(1), object(1)
memory usage: 368.0+ bytes
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 504 entries, 0 to 503
Columns: 2 entries, Subj to Study
dtypes: int64(1), object(1)
memory usage: 8.0+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98 entries, 0 to 97
Columns: 2 entries, Subj to Study
dtypes: int64(1), object(1)
memory usage: 1.7+ KB
None
<class 'pandas.core.frame.DataFrame'>
Index: 15 entries, Subj_1 to Subj_15
Columns: 95 entries, Losses_1 to Losses_95
dtypes: int64(95)
memory usage: 11.2+ KB
None
<class 'pandas.core.frame.DataFrame'>
Index: 504 entries, Subj_1 to Subj_504
Columns: 100 entries, Losses_1 to Losses_100
dtypes: int64(100)
memory usage: 397.7+ KB
None
<class 'pandas.core.frame.DataFrame'>
Index: 98 entries, Subj_1 to Subj_98
Columns: 150 entries, Losses_1 to Losses_150
dtypes: int64(150)
memory usage: 115.6+ KB
None
<class 'pandas.core.frame.DataFrame'>
Index: 15 entries, Subj_1 to Subj_15
Columns: 95 entries, Wins_1 to Wins_95
dtypes: int64(95)
memory usage: 11.2+ KB
None
<class 'pandas.core.frame.DataFrame'>
Index: 504 entries, Subj_1 to Subj_504
Columns: 100 entries, Wins_1 to Wins_100
dtypes: int64(100)
memory usage: 397.7+ KB
None
<class 'pandas.core.frame.DataFrame'>
Index: 98 entries, Subj_1 to Subj_98
Columns: 150 entries, Wins_1 to Wins_150
dtypes: int64(150)
memory usage: 115.6+ KB
None
```

```python
print(choice_100.isnull().values.any())
print(choice_150.isnull().values.any())
print(choice_95.isnull().values.any())
print(index_100.isnull().values.any())
print(index_150.isnull().values.any())
print(index_95.isnull().values.any())
print(lo_100.isnull().values.any())
print(lo_150.isnull().values.any())
print(lo_95.isnull().values.any())
print(wi_100.isnull().values.any())
print(wi_150.isnull().values.any())
print(wi_95.isnull().values.any())
```

```
False
False
False
False
False
False
False
False
False
False
False
False
```

All columns were non-null and of the same type (int64), so I did not need to change any datatypes. The data was already clean.

2. Merging the Datasets

In this step, I merged the datasets to include choice, amount won/lost, and index.

I began by changing the column names in the wins and losses datasets to 'Total_{n}'. This made the merging of the two dataframes easier because I was then able to use `pandas.DataFrame.add()`to get the total amount won or lost for each choice.

```
ld, wd = {}, {}
for i in range(1, 96):
    ld['Losses_' + str(i)] = 'Total_' + str(i)
    wd['Wins_' + str(i)] = 'Total_' + str(i)

lo_95.rename(columns=ld, inplace=True)
wi_95.rename(columns=wd, inplace=True)

total_95 = wi_95.add(lo_95)
total_95
```

	Total_1	Total_2	Total_3	Total_4	Total_5	Total_6	Total_7	Total_8	Total_9	Tot
Subj_1	100	100	100	100	100	100	100	100	-1150	
Subj_2	100	100	50	100	100	100	100	100	100	
Subj_3	50	50	50	100	100	100	100	-50	100	
Subj_4	50	50	100	100	-50	100	100	50	100	
Subj_5	100	100	50	50	50	100	-50	100	100	
Subj_6	100	100	100	100	-50	100	100	50	50	
Subj_7	100	100	50	50	50	50	100	100	100	
Subj_8	50	100	100	50	100	50	100	100	-50	
Subj_9	100	100	100	100	50	50	-50	50	50	
Subj_10	50	100	100	100	50	100	100	50	50	
Subj_11	50	50	0	100	50	100	-50	100	100	
Subj_12	50	50	50	50	100	100	100	100	100	
Subj_13	100	100	50	50	50	100	100	100	-50	
Subj_14	100	100	50	50	100	100	-50	100	50	
Subj_15	100	100	50	50	50	50	50	100	100	

15 rows × 95 columns

I then joined 'total_95' with 'choice_95' so that the dataframe, 'all_95', now contains the total amount won or lost and the choice that was made for each round. The columns alternate between total and choice because it is easier to understand that way.

```
all_95 = total_95.join(choice_95)

cols = all_95.columns.tolist()
cols = sorted(cols, key = lambda x: int(x.split('_')[-1]))
all_95 = all_95[cols]
all_95
```

	Total_1	Choice_1	Total_2	Choice_2	Total_3	Choice_3	Total_4	Choice_4	Tota
Subj_1	100	2	100	2	100	2	100	2	
Subj_2	100	1	100	2	50	3	100	2	
Subj_3	50	3	50	4	50	3	100	2	
Subj_4	50	4	50	3	100	1	100	1	
Subj_5	100	1	100	2	50	3	50	4	
Subj_6	100	1	100	2	100	1	100	2	
Subj_7	100	1	100	2	50	3	50	4	
Subj_8	50	4	100	2	100	1	50	3	
Subj_9	100	1	100	2	100	1	100	2	
Subj_10	50	4	100	2	100	2	100	1	
Subj_11	50	3	50	3	0	3	100	1	
Subj_12	50	4	50	4	50	4	50	4	
Subj_13	100	1	100	1	50	4	50	3	
Subj_14	100	1	100	2	50	3	50	4	
Subj_15	100	1	100	2	50	3	50	4	

15 rows × 190 columns

I changed the cells in ‘index_95’ to be in the form ‘Subj_{}’ and set that column as the index. I then joined it with ‘all_95’.

```
index_95['Subj'] = index_95['Subj'].apply(lambda x: 'Subj_' + str(x))
index_95.set_index('Subj', inplace=True)
index_95.index.name = None
all_95 = all_95.join(index_95)
```

Finally, I added multilevel columns to make each trial separate. I then exported the file in csv format to the data folder in my book.

```
l = []
for i in range(1, 96):
    l.append([i, 'Total_' + str(i)])
    l.append([i, 'Choice_' + str(i)])
l.append(['Name', 'Study'])
all_95.columns = pd.MultiIndex.from_tuples(l)

all_95.to_csv(("./data/all_95.csv"))
print(all_95.head())
```

```
             1                2                3                4           \
       Total_1 Choice_1 Total_2 Choice_2 Total_3 Choice_3 Total_4 Choice_4
Subj_1     100        2     100        2     100        2     100        2
Subj_2     100        1     100        2      50        3     100        2
Subj_3      50        3      50        4      50        3     100        2
Subj_4      50        4      50        3     100        1     100        1
Subj_5     100        1     100        2      50        3      50        4

             5           ...        91       92                93            \
       Total_5 Choice_5  ... Choice_91 Total_92 Choice_92 Total_93 Choice_93
Subj_1     100        2  ...         4       50         4     -200         4
Subj_2     100        2  ...         2       50         3       50         4
Subj_3     100        2  ...         4     -200         4       50         4
Subj_4     -50        1  ...         3       25         3       50         4
Subj_5      50        3  ...         4       50         4       50         4

              94                 95                Name
       Total_94 Choice_94 Total_95 Choice_95      Study
Subj_1       50         4       50         4  Fridberg
Subj_2       50         4       25         3  Fridberg
Subj_3       50         4       50         4  Fridberg
Subj_4       50         4       50         4  Fridberg
Subj_5       50         4       50         4  Fridberg

[5 rows x 191 columns]
```

I repeated this process for the other datasets.

```python
ld, wd = {}, {}
for i in range(1, 101):
    ld['Losses_' + str(i)] = 'Total_' + str(i)
    wd['Wins_' + str(i)] = 'Total_' + str(i)

lo_100.rename(columns=ld, inplace=True)
wi_100.rename(columns=wd, inplace=True)

total_100 = wi_100.add(lo_100)

all_100 = total_100.join(choice_100)

cols = all_100.columns.tolist()
cols = sorted(cols, key = lambda x: int(x.split('_')[-1]))
all_100 = all_100[cols]

index_100['Subj'] = index_100['Subj'].apply(lambda x: 'Subj_' + str(x))
index_100.set_index('Subj', inplace=True)
index_100.index.name = None
all_100 = all_100.join(index_100)

l = []
for i in range(1, 101):
    l.append([i, 'Total_' + str(i)])
    l.append([i, 'Choice_' + str(i)])
l.append(['Name', 'Study'])
all_100.columns = pd.MultiIndex.from_tuples(l)

all_100.to_csv(("./data/all_100.csv"))
print(all_100.head())
```

```
              1                2                3                4            \
        Total_1 Choice_1 Total_2 Choice_2 Total_3 Choice_3 Total_4 Choice_4
 Subj_1    -100        1     -50        1     100        2    -200        4
 Subj_2     100        2     100        1      50        4      50        4
 Subj_3      50        4     100        2       0        3    -200        1
 Subj_4    -200        4       0        3      50        4     100        2
 Subj_5     100        1     100        2     100        2   -1150        2

              5           ...       96       97                 98            \
        Total_5 Choice_5  ... Choice_96 Total_97 Choice_97 Total_98 Choice_98
 Subj_1      50        3  ...         2       50         4      100         2
 Subj_2      50        3  ...         4       50         4      100         2
 Subj_3      50        4  ...         2      100         2       50         4
 Subj_4    -100        1  ...         4      100         1       50         4
 Subj_5     100        2  ...         3       50         3        0         3

              99                 100                   Name
        Total_99 Choice_99 Total_100 Choice_100      Study
 Subj_1       50         4     -1150          2  Horstmann
 Subj_2      100         2        50          4  Horstmann
 Subj_3      100         2       100          2  Horstmann
 Subj_4        0         3        50          4  Horstmann
 Subj_5       50         4        50          4  Horstmann

 [5 rows x 201 columns]
```

```
ld, wd = {}, {}
for i in range(1, 151):
    ld['Losses_' + str(i)] = 'Total_' + str(i)
    wd['Wins_' + str(i)] = 'Total_' + str(i)

lo_150.rename(columns=ld, inplace=True)
wi_150.rename(columns=wd, inplace=True)

total_150 = wi_150.add(lo_150)

all_150 = total_150.join(choice_150)

cols = all_150.columns.tolist()
cols = sorted(cols, key = lambda x: int(x.split('_')[-1]))
all_150 = all_150[cols]

index_150['Subj'] = index_150['Subj'].apply(lambda x: 'Subj_' + str(x))
index_150.set_index('Subj', inplace=True)
index_150.index.name = None
all_150 = all_150.join(index_150)

l = []
for i in range(1, 151):
    l.append([i, 'Total_' + str(i)])
    l.append([i, 'Choice_' + str(i)])
l.append(['Name', 'Study'])
all_150.columns = pd.MultiIndex.from_tuples(l)

all_150.to_csv(("./data/all_150.csv"))
print(all_150.head())
```

```
             1                2                3                4          \
       Total_1 Choice_1 Total_2 Choice_2 Total_3 Choice_3 Total_4 Choice_4
Subj_1    -200        4     100        1    -250        1     100        1
Subj_2    -150        1    -250        1      50        4      50        4
Subj_3     100        2      50        4     100        1      50        3
Subj_4      50        4      50        4      50        4      50        4
Subj_5      50        4      50        4      50        4      50        4

             5           ...         146       147                  148  \
       Total_5 Choice_5  ... Choice_146 Total_147 Choice_147 Total_148
Subj_1      50        3  ...          1         0          3       100
Subj_2      50        4  ...          4      -200          1        50
Subj_3     -50        1  ...          4        50          4        50
Subj_4     -50        1  ...          4        50          4        50
Subj_5      50        4  ...          4        50          4        50

                          149                  150                      Name
       Choice_148 Total_149 Choice_149 Total_150 Choice_150            Study
Subj_1          2        50          4       -50          1  Steingroever2011
Subj_2          4         0          3       100          1  Steingroever2011
Subj_3          4        50          4        50          4  Steingroever2011
Subj_4          4        50          4        50          4  Steingroever2011
Subj_5          4        50          4      -200          4  Steingroever2011

[5 rows x 301 columns]
```

3. Aggregating the Data

Then, I aggregated the ‘all’ dataframes into one dataframe and normalised the data.

I began by adding up each choice option so that I had the total number of times A, B, C and D were picked. For the purposes of this task I assumed that 1 was ‘A’, 2 was ‘B’ etc. I then added the totals so that I could see the total amount won or lost by each participant. I then added these columns as well as study to a dataframe named ‘agg_95’.

```python
data = []
for row in all_95.iterrows():
    total, a, b, c, d = 0,0,0,0,0
    for j in range(0, len(list(row[1])) - 1):

        i = list(row[1])[j]
        if j % 2 != 0:
            if i == 1:
                a += 1
            elif i == 2:
                b += 1
            elif i == 3:
                c += 1
            elif i == 4:
                d += 1

        else:
            total += i
    data.append(['95_' + row[1].name, total, a, b, c, d, list(row[1])[-1]])

agg_95 = pd.DataFrame(data, columns=['Subj', 'Total', 'A', 'B', 'C', 'D', 'Study'])
agg_95
```

	Subj	Total	A	B	C	D	Study
0	95_Subj_1	1150	12	9	3	71	Fridberg
1	95_Subj_2	-675	24	26	12	33	Fridberg
2	95_Subj_3	-750	12	35	10	38	Fridberg
3	95_Subj_4	-525	11	34	12	38	Fridberg
4	95_Subj_5	100	10	24	15	46	Fridberg
5	95_Subj_6	1250	6	18	20	51	Fridberg
6	95_Subj_7	-150	19	31	8	37	Fridberg
7	95_Subj_8	150	12	28	10	45	Fridberg
8	95_Subj_9	-575	10	34	12	39	Fridberg
9	95_Subj_10	1475	3	20	12	60	Fridberg
10	95_Subj_11	-350	14	40	9	32	Fridberg
11	95_Subj_12	-325	15	42	17	21	Fridberg
12	95_Subj_13	450	11	30	14	40	Fridberg
13	95_Subj_14	-425	14	37	17	27	Fridberg
14	95_Subj_15	450	14	17	23	41	Fridberg

This was then repeated for 100 and 150.

```python
data = []
for row in all_100.iterrows():
    total, a, b, c, d = 0,0,0,0,0
    for j in range(0, len(list(row[1])) - 1):

        i = list(row[1])[j]
        if j % 2 != 0:
            if i == 1:
                a += 1
            elif i == 2:
                b += 1
            elif i == 3:
                c += 1
            elif i == 4:
                d += 1

        else:
            total += i
    data.append(['100_' + row[1].name, total, a, b, c, d, list(row[1])[-1]])

agg_100 = pd.DataFrame(data, columns=['Subj','Total', 'A', 'B', 'C', 'D', 'Study'])
agg_100
```

	Subj	Total	A	B	C	D	Study
0	100_Subj_1	-1800	21	42	15	22	Horstmann
1	100_Subj_2	-800	14	35	18	33	Horstmann
2	100_Subj_3	-450	21	42	7	30	Horstmann
3	100_Subj_4	1200	13	24	28	35	Horstmann
4	100_Subj_5	-1300	15	31	28	26	Horstmann
...	...	...	...	...	...	...	...
499	100_Subj_500	75	17	29	28	26	Worthy
500	100_Subj_501	600	14	15	44	27	Worthy
501	100_Subj_502	-1525	27	32	17	24	Worthy
502	100_Subj_503	-750	27	25	23	25	Worthy
503	100_Subj_504	175	10	24	12	54	Worthy

504 rows × 7 columns

```
data = []
for row in all_150.iterrows():
    total, a, b, c, d = 0,0,0,0,0
    for j in range(0, len(list(row[1])) - 1):

        i = list(row[1])[j]
        if j % 2 != 0:
            if i == 1:
                a += 1
            elif i == 2:
                b += 1
            elif i == 3:
                c += 1
            elif i == 4:
                d += 1

        else:
            total += i
    data.append(['150_' + row[1].name, total, a, b, c, d, list(row[1])[-1]])

agg_150 = pd.DataFrame(data, columns=['Subj', 'Total', 'A', 'B', 'C', 'D', 'Study'])
agg_150
```

	Subj	Total	A	B	C	D	Study
0	150_Subj_1	-550	46	37	29	38	Steingroever2011
1	150_Subj_2	-1600	40	57	19	34	Steingroever2011
2	150_Subj_3	900	19	35	8	88	Steingroever2011
3	150_Subj_4	2200	18	11	10	111	Steingroever2011
4	150_Subj_5	1900	13	1	1	135	Steingroever2011
...	...	...	...	...	...	...	...
93	150_Subj_94	300	24	69	13	44	Wetzels
94	150_Subj_95	2150	5	31	46	68	Wetzels
95	150_Subj_96	1450	18	19	37	76	Wetzels
96	150_Subj_97	1200	25	30	44	51	Wetzels
97	150_Subj_98	-1800	11	104	6	29	Wetzels

98 rows × 7 columns

I then normalised them by dividing each dataframe by the amount of trials it contained.

```
agg_150[['Total', 'A', 'B', 'C', 'D']]/= 150
agg_100[['Total', 'A', 'B', 'C', 'D']]/= 100
agg_95[['Total', 'A', 'B', 'C', 'D']]/= 95
```

I then added in a ‘Good’ column (the sum of C and D), a ‘Bad’ column (the sum of A and B), a column for the numeric representation of each study and a column with the payload of the study. I then saved the dataframe as a csv file in the data folder under ‘agg_all.csv’.

```
agg_all = pd.concat([agg_95, agg_100, agg_150])
agg_all.reset_index(inplace=True, drop=True)
agg_all['Bad'] = agg_all['A'] + agg_all['B']
agg_all['Good'] = agg_all['C'] + agg_all['D']

agg_all['StudyNo'] = ''
agg_all['Payload'] = ''

stud_d = {'Fridberg': 1, 'Horstmann': 2, 'Kjome': 3, 'Maia': 4, 'Premkumar': 5,
'Steingroever2011': 6,
    'SteingroverInPrep': 7, 'Wetzels': 8, 'Wood': 9, 'Worthy': 10}
payload_d = {'Fridberg': 1, 'Horstmann': 2, 'Kjome': 3, 'Maia': 1, 'Premkumar': 3,
'Steingroever2011': 2,
    'SteingroverInPrep': 2, 'Wetzels': 2, 'Wood': 3, 'Worthy': 1}

for i in range(0, len(agg_all)):
    agg_all.loc[i, 'StudyNo'] = stud_d[agg_all['Study'][i]]
    agg_all.loc[i, 'Payload'] = payload_d[agg_all['Study'][i]]


agg_all.to_csv("./data/agg_all.csv")
agg_all
```

	Subj	Total	A	B	C	D	Study	Bad
0	95_Subj_1	12.105263	0.126316	0.094737	0.031579	0.747368	Fridberg	0.221053
1	95_Subj_2	-7.105263	0.252632	0.273684	0.126316	0.347368	Fridberg	0.526316
2	95_Subj_3	-7.894737	0.126316	0.368421	0.105263	0.400000	Fridberg	0.494737
3	95_Subj_4	-5.526316	0.115789	0.357895	0.126316	0.400000	Fridberg	0.473684
4	95_Subj_5	1.052632	0.105263	0.252632	0.157895	0.484211	Fridberg	0.357895
...	...	...	...	...	...	...	...	...
612	150_Subj_94	2.000000	0.160000	0.460000	0.086667	0.293333	Wetzels	0.620000
613	150_Subj_95	14.333333	0.033333	0.206667	0.306667	0.453333	Wetzels	0.240000
614	150_Subj_96	9.666667	0.120000	0.126667	0.246667	0.506667	Wetzels	0.246667
615	150_Subj_97	8.000000	0.166667	0.200000	0.293333	0.340000	Wetzels	0.366667
616	150_Subj_98	-12.000000	0.073333	0.693333	0.040000	0.193333	Wetzels	0.766667

617 rows × 11 columns

The data was then ready for me to use in my cluster analysis.

Cluster Analysis

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group are more similar to each other than to those in other groups.

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
sns.set_palette('gnuplot2', n_colors=10)
from sklearn.cluster import KMeans
from collections import Counter
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

```
all_95 = pd.read_csv("./data/all_95.csv", header = [0,1], index_col=0)
all_100 = pd.read_csv("./data/all_100.csv", header = [0,1], index_col=0)
all_150 = pd.read_csv("./data/all_150.csv", header = [0,1], index_col=0)
df = pd.read_csv("./data/agg_all.csv", index_col=0)
df
```

	Subj	Total	A	B	C	D	Study	Bad
0	95_Subj_1	12.105263	0.126316	0.094737	0.031579	0.747368	Fridberg	0.221053
1	95_Subj_2	-7.105263	0.252632	0.273684	0.126316	0.347368	Fridberg	0.526316
2	95_Subj_3	-7.894737	0.126316	0.368421	0.105263	0.400000	Fridberg	0.494737
3	95_Subj_4	-5.526316	0.115789	0.357895	0.126316	0.400000	Fridberg	0.473684
4	95_Subj_5	1.052632	0.105263	0.252632	0.157895	0.484211	Fridberg	0.357895
...	...	...	...	...	...	...	...	...
612	150_Subj_94	2.000000	0.160000	0.460000	0.086667	0.293333	Wetzels	0.620000
613	150_Subj_95	14.333333	0.033333	0.206667	0.306667	0.453333	Wetzels	0.240000
614	150_Subj_96	9.666667	0.120000	0.126667	0.246667	0.506667	Wetzels	0.246667
615	150_Subj_97	8.000000	0.166667	0.200000	0.293333	0.340000	Wetzels	0.366667
616	150_Subj_98	-12.000000	0.073333	0.693333	0.040000	0.193333	Wetzels	0.766667

617 rows × 11 columns

1. The Optimal Number of Clusters

There are two methods to find the optimal number of clusters for a dataset, the Elbow Method and the Silhouette Method.

1.1 The Elbow Method

This is the most common method for determining the optimal number of clusters. To do this you must calculate the Within-Cluster-Sum of Squared Errors (WSS) for different values of *k*, and choose the *k* for which WSS first starts to diminish. In a plot of WSS-versus-k, this is visible as an elbow.

```
def elbow_score(x):
    distortions = []
    K = range(1,11)
    for k in K:
        kmeanModel = KMeans(n_clusters=k)
        kmeanModel.fit(x)
        distortions.append(kmeanModel.inertia_)
    plt.plot(K, distortions, 'bx-')
    plt.xlabel('k')
    plt.ylabel('Distortion')
    plt.title('The Elbow Method showing the optimal k')
    plt.show()
```

1.2 The Silhouette Method

This is a method to find the optimal number of clusters *k*. The silhouette value measures how similar a point is to its own cluster (cohesion) compared to other clusters (separation). A high value is desirable. This is the formula:

$$ s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))} $$

> ***NOTE:*** *s(i)* is defined to be equal to zero if *i* is the only point in the cluster. This is to prevent the number of clusters from increasing significantly with many single-point clusters.

a(i) is the measure of the similarity of the point *i* to its own cluster.

$$ a(i) = \frac{1}{\lvert C_{i}\rvert - 1} \sum_{j{\in}C_{i}, i{\neq}j} d(i,j) $$

Similarly, *b(i)* is the measure of dissimilarity of *i* from points in other clusters.

$$ b(i) = \min_{i{\neq}j} \frac{1}{\lvert C_{j}\rvert} \sum_{j{\in}C_{j}} d(i,j) $$

Where *d(i,j)* is the euclidean distance between the two points.

```
def sil_value(x):
    sil = []
    kmax = 10

    # dissimilarity would not be defined for a single cluster, thus, minimum number of
clusters should be 2
    for k in range(3, kmax+1):
      kmeans = KMeans(n_clusters = k).fit(x)
      labels = kmeans.labels_
      sil.append(silhouette_score(x, labels, metric = 'euclidean'))

    return sil

# https://medium.com/analytics-vidhya/how-to-determine-the-optimal-k-for-k-means-
708505d204eb
```

2. Clustering the Data

I chose to perform clustering on the total amount won or lost compared to the amount the participants chose the deck ‘C’. I thought this would be interesting because C was a “good” deck but had frequent losses. The losses also depended on the payload of the study. I wanted to see if the clusters had any relationship to the payload.

I also wanted to cluster the data comparing the total amount won or lost to the amount of times the participants chose a “good” deck. This would be interesting to see if there was anything in particular that defined each of these clusters.

2.1 Performing Clustering on Deck C

I began by making a dataframe with just ‘Total’ and ‘C’ columns.

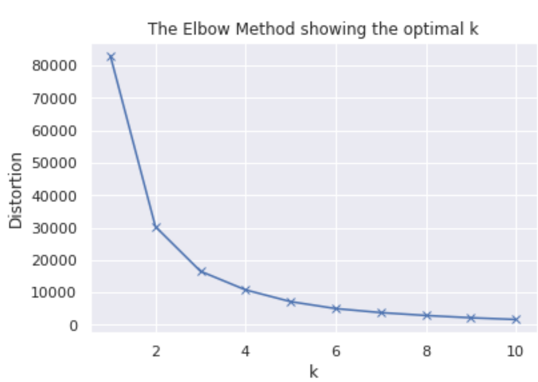
```
c = df.iloc[:,[1,4]]
c
```

	Total	C
0	12.105263	0.031579
1	-7.105263	0.126316
2	-7.894737	0.105263
3	-5.526316	0.126316
4	1.052632	0.157895
...	...	...
612	2.000000	0.086667
613	14.333333	0.306667
614	9.666667	0.246667
615	8.000000	0.293333
616	-12.000000	0.040000

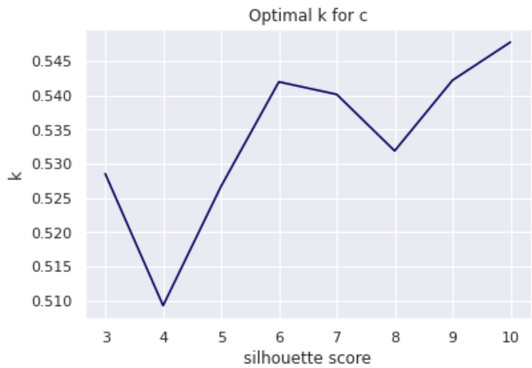
617 rows × 2 columns

I then used the elbow method and the silhouette method to find the optimal number of clusters.

```
elbow_score(c)
```



```
c_sil = sil_value(c)

plt.plot([3,4,5,6,7,8,9,10], c_sil)
plt.title('Optimal k for c')
plt.xlabel('silhouette score')
plt.ylabel('k')
plt.show()
```



It was not completely clear from the elbow method whick value I should use for *k*. It looked to be between 3 and 7. In the silhouette method the peak was at 10 but there was also a slightly lower peak at 7. For this reason I chose 7 as my *k* and made 7 clusters.

I used `sklearn.cluster.KMeans` to create the clusters and fit them to the data.

```
c_kmeans = KMeans(7)
c_kmeans.fit(c)
```

```
KMeans(n_clusters=7)
```

I then find the identified clusters in c.

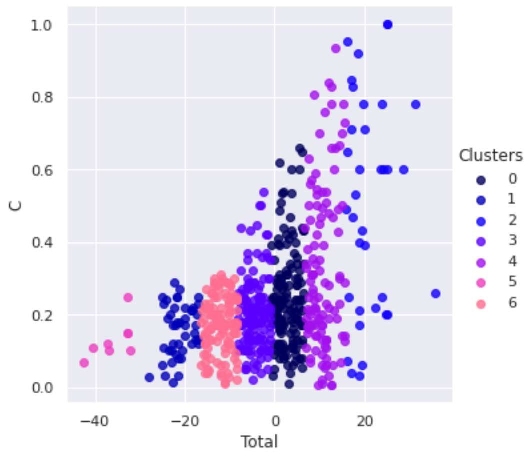
```
c_identified_clusters = c_kmeans.fit_predict(c)
c_identified_clusters
```

```
array([4, 3, 3, 3, 0, 4, 3, 0, 3, 4, 3, 3, 0, 3, 0, 1, 3, 3, 4, 6, 6, 0,
       6, 4, 3, 0, 0, 0, 3, 0, 0, 6, 6, 3, 0, 3, 0, 6, 0, 4, 4, 3, 0, 3,
       0, 4, 3, 0, 3, 4, 3, 0, 6, 1, 4, 0, 0, 0, 6, 6, 3, 3, 3, 0, 1, 0,
       6, 1, 4, 0, 0, 3, 3, 3, 6, 0, 2, 3, 4, 6, 0, 0, 0, 4, 4, 6, 2, 0,
       4, 3, 0, 6, 0, 3, 4, 0, 3, 0, 4, 6, 3, 4, 1, 3, 6, 4, 0, 4, 4, 3,
       0, 3, 3, 0, 0, 4, 0, 6, 4, 0, 0, 4, 3, 4, 4, 0, 1, 0, 0, 3, 0, 6,
       1, 3, 4, 0, 4, 4, 6, 0, 0, 3, 4, 4, 0, 3, 4, 3, 3, 4, 4, 4, 0, 6,
       6, 0, 2, 0, 3, 3, 6, 4, 3, 4, 3, 4, 3, 6, 3, 0, 0, 0, 1, 0, 6, 0,
       0, 1, 3, 6, 3, 4, 4, 0, 3, 6, 1, 0, 3, 5, 0, 4, 4, 3, 0, 1, 0, 0,
       0, 0, 0, 3, 4, 4, 3, 0, 0, 0, 2, 0, 4, 3, 3, 3, 4, 3, 4, 3, 4, 3,
       2, 4, 4, 6, 2, 0, 0, 6, 2, 4, 3, 0, 3, 0, 4, 0, 0, 6, 0, 2, 6, 6,
       3, 0, 4, 3, 6, 0, 4, 3, 3, 4, 4, 6, 4, 0, 3, 3, 6, 0, 4, 3, 4, 1,
       0, 6, 6, 6, 4, 0, 0, 1, 4, 4, 0, 6, 4, 3, 0, 6, 3, 6, 3, 2, 1, 3,
       4, 4, 6, 0, 0, 0, 3, 0, 4, 6, 3, 6, 6, 4, 6, 2, 1, 2, 3, 4, 5, 1,
       6, 2, 6, 2, 1, 2, 4, 0, 2, 0, 6, 2, 3, 4, 0, 2, 6, 2, 0, 6, 4, 1,
       2, 3, 6, 1, 3, 6, 3, 1, 3, 0, 3, 6, 1, 6, 6, 6, 2, 4, 1, 6, 6, 3,
       5, 3, 3, 6, 6, 0, 4, 3, 3, 1, 6, 4, 0, 6, 0, 6, 3, 3, 6, 6, 1, 0,
       3, 0, 4, 6, 6, 3, 3, 3, 1, 1, 6, 6, 3, 6, 6, 3, 1, 4, 6, 4, 4, 4,
       1, 2, 3, 0, 6, 3, 6, 5, 0, 0, 3, 6, 3, 3, 6, 6, 6, 4, 3, 1, 0, 6,
       3, 6, 4, 6, 6, 1, 6, 3, 4, 6, 0, 6, 3, 2, 3, 5, 1, 0, 2, 3, 6, 3,
       1, 6, 0, 6, 3, 0, 6, 6, 6, 5, 6, 3, 3, 6, 1, 1, 3, 1, 6, 3, 1, 6,
       6, 4, 6, 3, 3, 2, 3, 6, 1, 1, 1, 2, 5, 5, 3, 1, 6, 2, 6, 1, 4, 1,
       3, 4, 3, 0, 1, 3, 3, 3, 6, 3, 3, 3, 3, 6, 3, 3, 3, 3, 3, 1, 6, 3,
       0, 6, 6, 2, 0, 3, 3, 3, 0, 0, 6, 3, 0, 3, 6, 0, 4, 4, 6, 0, 2, 0,
       2, 4, 3, 4, 6, 0, 4, 3, 4, 0, 4, 4, 3, 6, 3, 0, 6, 4, 6, 1, 0, 6,
       6, 6, 2, 1, 0, 4, 2, 0, 0, 0, 0, 0, 3, 6, 4, 0, 0, 3, 6, 0, 4, 3,
       0, 2, 6, 2, 3, 0, 2, 0, 3, 4, 4, 6, 0, 0, 0, 4, 4, 2, 0, 0, 0, 4,
       0, 4, 0, 4, 4, 6, 0, 6, 3, 3, 4, 3, 0, 4, 4, 4, 6, 4, 0, 4, 4, 4,
       6], dtype=int32)
```

This is a graph of the data with the different clusters shown in different colours.

```
c_data_with_clusters = df.copy()
c_data_with_clusters['Clusters'] = c_identified_clusters

sns.lmplot(data=c_data_with_clusters, x='Total', y='C', hue='Clusters',
           fit_reg=False, legend=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb5582df650>
```



Then, for a comparison, I plotted the data using colour to differentiate payload.

```
sns.lmplot(data=c_data_with_clusters, x='Total', y='C', hue='Payload',
           fit_reg=False, legend=True, palette='rainbow')
```

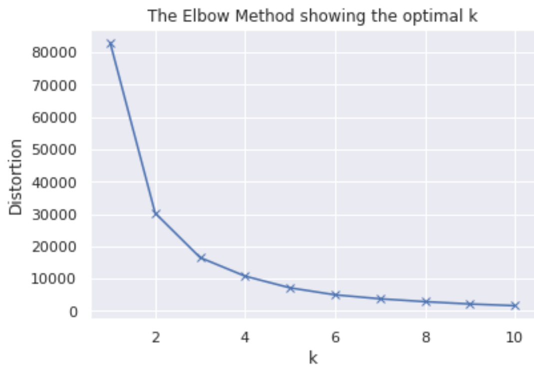
```
<seaborn.axisgrid.FacetGrid at 0x7fb5582bf090>
```

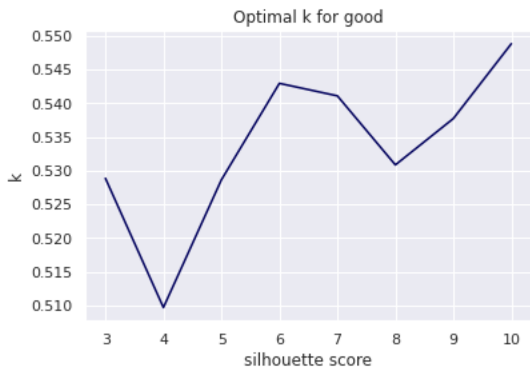
As you can see the datapoints in cluster 6 all chose c less and had the highest loss in the study. All of these data points were in payload 3. This makes sense because these people obviously chose C less because they were getting frequent losses from it and those losses were growing as they went. Most of the people who chose C the most were in payload 2 which is where the losses in C were constant.

2.2 Performing Clustering on “Good” Decks

Now I would like to cluster the data based on the “good” card decks to see if there is any insight to be gleamed from it.

```
good = df.iloc[:,[1,7]]

elbow_score(good)

good_sil = sil_value(good)
good_k = good_sil.index(max(good_sil)) + 2

plt.plot([3,4,5,6,7,8,9,10], good_sil)
plt.title('Optimal k for good')
plt.xlabel('silhouette score')
plt.ylabel('k')
plt.show()
```

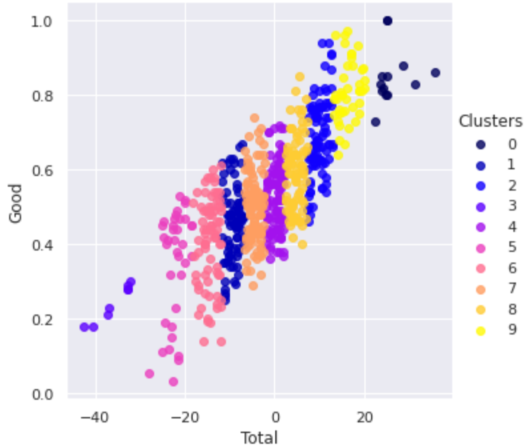


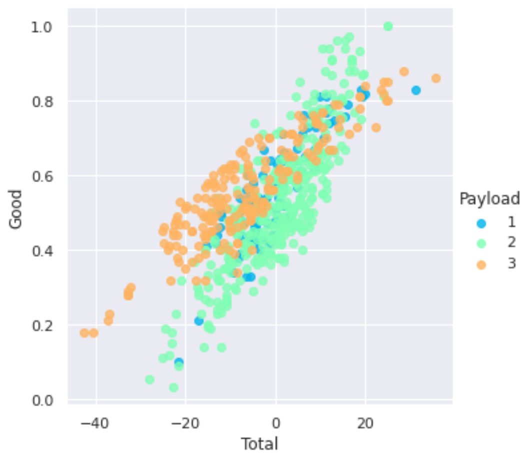


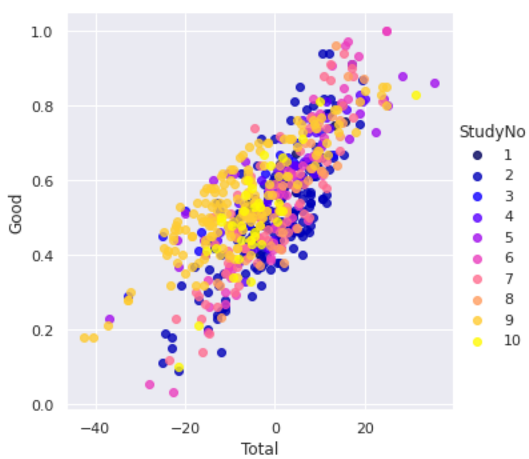
There is no clear *k* from the elbow plot, and the peak at 10 is a fair bit higher than any other peak in the silhouette plot. Therefore, I chose to make 10 clusters this time.

```
good_kmeans = KMeans(10)
good_kmeans.fit(good)
good_identified_clusters = good_kmeans.fit_predict(good)

good_data_with_clusters = df.copy()
good_data_with_clusters['Clusters'] = good_identified_clusters
sns.lmplot(data=good_data_with_clusters, x='Total', y='Good', hue='Clusters',
                   fit_reg=False, legend=True)


sns.lmplot(data=good_data_with_clusters, x='Total', y='Good', hue='Payload',
                   fit_reg=False, legend=True, palette='rainbow')


sns.lmplot(data=good_data_with_clusters, x='Total', y='Good', hue='StudyNo',
                   fit_reg=False, legend=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb55800f590>
```







As you can see from the two graphs, there is obviously a strong correlation between picking “good” decks and winning more money. There does not seem to be any correlation between payload and “good” decks though. Maia et al. frequently asked the participants about their knowledge of the decks at regular intervals during the task. This is study number 4 and is shown in yellow on the graph. Nothing major stands out about this study except there were less people who chose the “bad” decks. It is hard to see because the datapoints for study 4 are all in the centre of the graph but there are a lot less of their datapoints under 0.4 than the other studies.

2.3 Further Analysis of “Good” Decks

I would now like to look more closely at clusters 7 and 9 because they are the most extreme on each side. It is said that after about 50 choices most people begin to see the patterns ang go for the “good” decks more consistently. I want to see if the people in cluster 9 seemed to figure out the pattern and the people in cluster 7 did not.

I began by getting the number of the good cluster (9) and the bad cluster (7). This is just in case the cluster numbers change at some point.

```
good_cluster = good_data_with_clusters[good_data_with_clusters['Total'] ==
max(good_data_with_clusters['Total'])]['Clusters'].iloc(0)[0]
bad_cluster = good_data_with_clusters[good_data_with_clusters['Total'] ==
min(good_data_with_clusters['Total'])]['Clusters'].iloc(0)[0]
```

I then found the subjects of the cluster so that I could find out whether they belonged to 95, 100, or 150. I then looked at the choices they made after the first 50 rounds and did some analysis on that.

```
good_subj_1 = list(good_data_with_clusters[good_data_with_clusters['Clusters'] ==
bad_cluster]['Subj'])
good_subj_1

good_lst = []
for i in good_subj_1:
    n, idx = i.split('_', 1)
    if n == '100':
        good_lst.append(list(all_100[all_100.index == idx].values[0])[101::2])
    elif n == '150':
        good_lst.append(list(all_150[all_150.index == idx].values[0])[101::2])
    elif n == '95':
        good_lst.append(list(all_95[all_95.index == idx].values[0])[101::2])

good_d = {'a' : 0, 'b' : 0, 'c' : 0, 'd' : 0}
good_l, no_bad = [], []
for i in good_lst:
    c = Counter(i)
    good_d['a'] += c[1]
    good_d['b'] += c[2]
    good_d['c'] += c[3]
    good_d['d'] += c[4]
    if c[1] + c[2] != 0:
        good_l.append((c[3] + c[4]) / (c[1] + c[2]))
    else:
        no_bad.append(c)
print(good_d)
print(len([x for x in good_l if x < .5]))
print(len(good_l))
print(no_bad)
print(np.mean(good_l))
```

```
{'a': 67, 'b': 239, 'c': 46, 'd': 48}
6
8
[]
0.3690945336047337
```

This cluster chose ‘B’ the most and 6 out of 8 people were twice as likely to choose “bad” decks over “good” decks. This shows that the people in this cluster probably did not see the pattern and got distracted by the infrequent losses in Deck B.

The mean of the ratios between good and bad is 0.369, which shows that the participants in the cluster were far less likely to choose good decks. If the mean was 1 it would mean that the participants chose fairly evenly between the “good” and “bad” decks.

```
good_subj_2 = list(good_data_with_clusters[good_data_with_clusters['Clusters'] ==
good_cluster]['Subj'])
good_subj_2

good_lst = []
for i in good_subj_2:
    n, idx = i.split('_', 1)
    if n == '100':
        good_lst.append(list(all_100[all_100.index == idx].values[0])[101::2])
    elif n == '150':
        good_lst.append(list(all_150[all_150.index == idx].values[0])[101::2])
    elif n == '95':
        good_lst.append(list(all_95[all_95.index == idx].values[0])[101::2])

good_d = {'a' : 0, 'b' : 0, 'c' : 0, 'd' : 0}
good_l, no_bad = [], []
for i in good_lst:
    c = Counter(i)
    good_d['a'] += c[1]
    good_d['b'] += c[2]
    good_d['c'] += c[3]
    good_d['d'] += c[4]
    if c[1] + c[2] != 0:
        good_l.append((c[3] + c[4]) / (c[1] + c[2]))
    else:
        no_bad.append(c)

print(good_d)
print(len([x for x in good_l if x < 1]))
print(len(good_l))
print(no_bad)
print(np.mean(good_l))
```

```
{'a': 28, 'b': 18, 'c': 516, 'd': 188}
0
8
[Counter({3: 32, 4: 18}), Counter({3: 37, 4: 13}), Counter({3: 50}), Counter({3: 100}),
Counter({3: 100})]
15.049107142857144
```

In this cluster 'C' was chosen the most despite its frequent losses. None of the participants chose the "bad" decks more frequently than the "good" decks, with some people never choosing the bad decks at all. The mean was 15.049 which is very high. This means that the "bad" were barely chosen in comparison to the "good" decks.

For these reasons I think it is safe to assume that a lot of these participants figured out that the decks C and D contained more rewards than penalties.

3. Preserving the Privacy of Each Lab

Next, I wanted to find a way to obtain similar clustering results while preserving the privacy of each lab.

3.1 Using PCA

Principal Component Analysis, or PCA, is a way to reduce the dimensionality of a dataset. It also is able to provide some degree of privacy to the data.

First I dropped the columns that were not numeric and useful.

```
x = df.drop(columns = ['Subj','Study', 'StudyNo', 'Payload'])
```

I then performed PCA for two components and added them to a dataframe with the study number and payload.

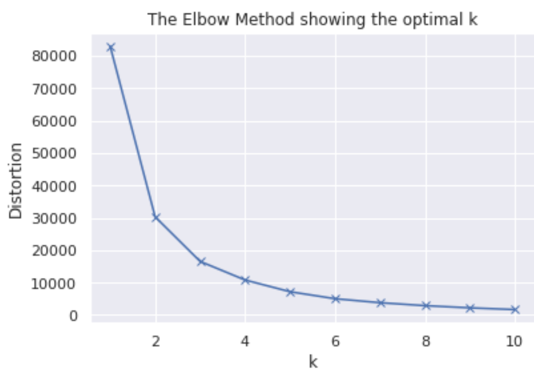
```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
pca_df = pd.concat([principalDf, df[['StudyNo', 'Payload']]], axis = 1)
# https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60

pca_df
```

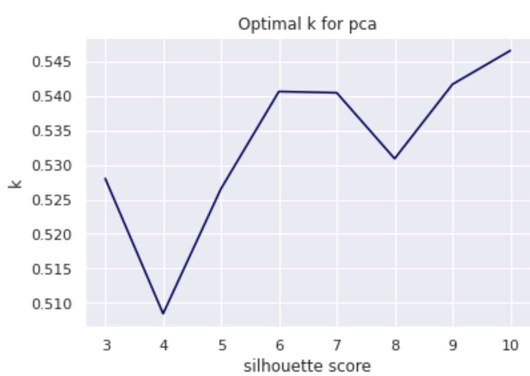
	principal component 1	principal component 2	StudyNo	Payload
0	-13.875027	-0.317474	1	1
1	5.341599	-0.071472	1	1
2	6.130536	-0.110633	1	1
3	3.761851	-0.107813	1	1
4	-2.819895	-0.101765	1	1
...	...	...	...	...
612	-3.757980	-0.293730	8	2
613	-16.102065	-0.053414	8	2
614	-11.436426	-0.046602	8	2
615	-9.766207	-0.009333	8	2
616	10.245009	-0.271539	8	2

617 rows × 4 columns

I then got the elbow and silhouette scores and chose 7 as my optimal *k*.

```
elbow_score(principalDf)

pca_sil = sil_value(principalDf)
pca_k = pca_sil.index(max(pca_sil)) + 2

plt.plot([3,4,5,6,7,8,9,10], pca_sil)
plt.title('Optimal k for pca')
plt.xlabel('silhouette score')
plt.ylabel('k')
plt.show()
```

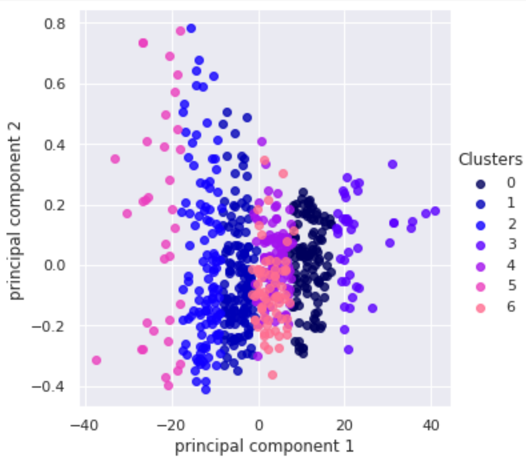


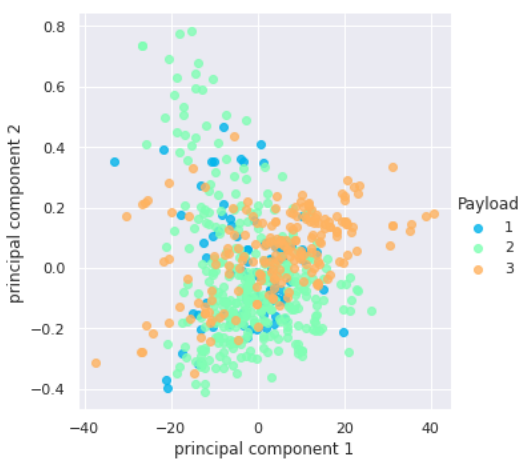


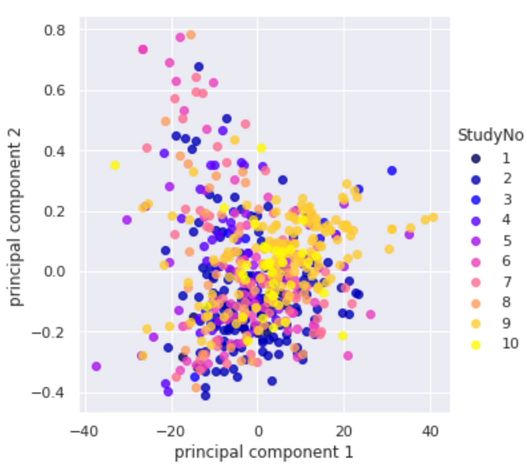
I then clustered the data and made graphs showing the clusters, the payload, and the study.

```
pca_kmeans = KMeans(7)
pca_kmeans.fit(pca_df)
pca_identified_clusters = pca_kmeans.fit_predict(pca_df)

pca_data_with_clusters = pca_df.copy()
pca_data_with_clusters['Clusters'] = pca_identified_clusters
sns.lmplot(data=pca_data_with_clusters, x='principal component 1', y='principal
component 2', hue='Clusters',
                   fit_reg=False, legend=True)


sns.lmplot(data=pca_data_with_clusters, x='principal component 1', y='principal
component 2', hue='Payload',
                   fit_reg=False, legend=True, palette='rainbow')



sns.lmplot(data=pca_df, x='principal component 1', y='principal component 2',
hue='StudyNo',
                   fit_reg=False, legend=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb55a4ce4d0>
```







These graphs do not look like the graphs I made before. There is more data in the top left quadrant of the graph. There does not seem to be much correlation between the clusters and the payoff but cluster 4 seems to be made up of majority payload 3 studies. There is no clear connection between these clusters and the studies.

3.2 Using Centroids

To do this I decided to perform clustering on each lab's results separately and then cluster their centroids to see if it gave similar results to clustering the data from all of the labs together. For this example I will use the same analysis as before, the 'Total' and the "good" decks.

I looped through the studies and got the silhouette score for each study. I then performed k-means clustering on the data from each study. I added the centroid of each cluste to a dataframe called 'centroids'.

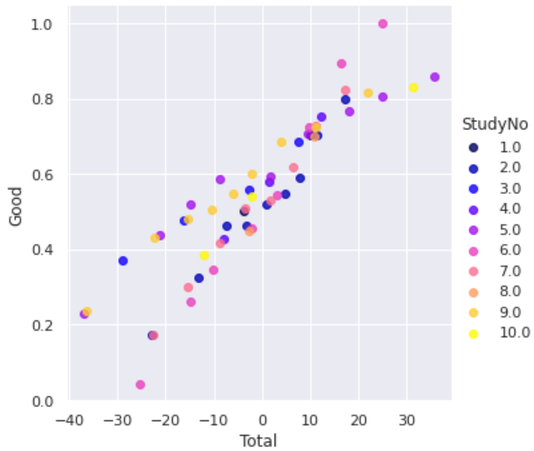
```
centroids = pd.DataFrame(columns = ['Total', 'Good', 'StudyNo'])
for study in range(1,11):

    study_df =  df[df['StudyNo'] == study]

    study_good = study_df.iloc[:,[1,7]]
    study_good_sil = sil_value(study_good)
    study_good_k = study_good_sil.index(max(study_good_sil)) + 2

#     plt.plot([3,4,5,6,7,8,9,10], study_good_sil)
#     plt.title('Optimal k for {}'.format(study_df['Study'].iloc(0)[0]))
#     plt.xlabel('silhouette score')
#     plt.ylabel('k')
#     plt.show()

    study_good_kmeans = KMeans(study_good_k)
    study_good_kmeans.fit(study_good)
    study_good_identified_clusters = study_good_kmeans.fit_predict(study_good)

    centers = np.array(study_good_kmeans.cluster_centers_)
    for i in centers:
        centroids = centroids.append({'Total': i[0], 'Good': 1-i[1], 'StudyNo': study},
ignore_index=True)

print(centroids)
```

```
        Total      Good  StudyNo
0   -3.710526  0.503158      1.0
1   10.052632  0.705263      1.0
2   11.500000  0.703333      2.0
3   -7.441176  0.462941      2.0
4    0.803571  0.518571      2.0
5  -13.261905  0.326190      2.0
6    4.703704  0.548889      2.0
7  -23.000000  0.173333      2.0
8   17.200000  0.800000      2.0
9   -3.208333  0.465000      2.0
10   7.894737  0.591579      2.0
11  -2.741667  0.558333      3.0
12 -28.975000  0.370000      3.0
13   7.485714  0.685714      3.0
14 -16.375000  0.477500      3.0
15   1.541667  0.581667      4.0
16  12.303571  0.753571      4.0
17  -8.000000  0.426250      4.0
18  25.050000  0.807500      5.0
19  -8.712500  0.587500      5.0
20 -21.150000  0.440000      5.0
21   1.700000  0.592500      5.0
22 -36.950000  0.230000      5.0
23  18.016667  0.766667      5.0
24  35.700000  0.860000      5.0
25   9.366667  0.706667      5.0
26 -14.825000  0.520000      5.0
27  -2.212121  0.456364      6.0
28   9.666667  0.725926      6.0
29 -10.111111  0.346667      6.0
30  25.000000  1.000000      6.0
31   3.095238  0.546190      6.0
32 -25.333333  0.043333      6.0
33  16.285714  0.893333      6.0
34 -14.888889  0.262222      6.0
35 -15.300000  0.301000      7.0
36   6.562500  0.618750      7.0
37  -3.450000  0.511000      7.0
38  17.285714  0.825714      7.0
39  -8.807692  0.417692      7.0
40   1.700000  0.532000      7.0
41 -22.750000  0.175000      7.0
42  11.000000  0.728000      7.0
43  10.933333  0.699333      8.0
44  -2.793651  0.450476      8.0
45  11.034615  0.726154      9.0
46 -15.393548  0.480323      9.0
47 -10.405769  0.506154      9.0
48 -22.326471  0.432353      9.0
49 -36.383333  0.238333      9.0
50  22.042857  0.815714      9.0
51  -2.050000  0.600000      9.0
52  -5.973214  0.547500      9.0
53   3.970833  0.685833      9.0
54  -2.152174  0.540870     10.0
55  31.250000  0.830000     10.0
56 -12.068182  0.387273     10.0
```

Here is a graph of all the centroids. As you can see, it is similar to the graph for all datapoints when plotting ‘Total’ and ‘Good’. It also still captures that some studies had more variation in their ‘Total’ or in the amount a “good” deck was chosen. For example, the 6th study had people choosing “good” decks all the time and some people never choosing them. In contrast, the people in study 4 mostly chose a “good” deck somewhere between 40% and 80% of the time.

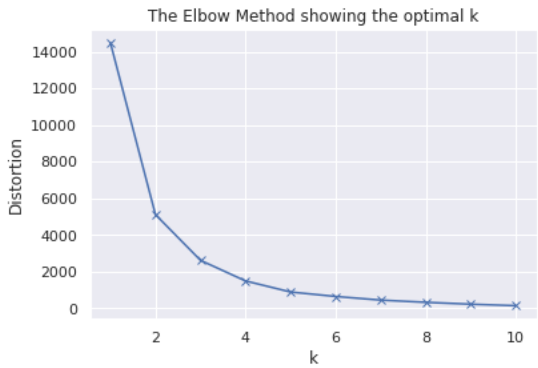
```
sns.lmplot(data=centroids, x='Total', y='Good', hue='StudyNo',
           fit_reg=False, legend=True, palette='gnuplot2')
```

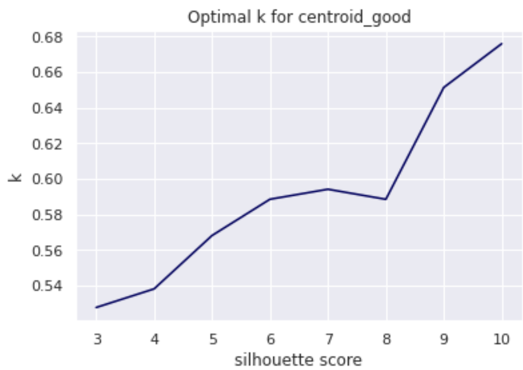
```
<seaborn.axisgrid.FacetGrid at 0x7fb544c461d0>
```



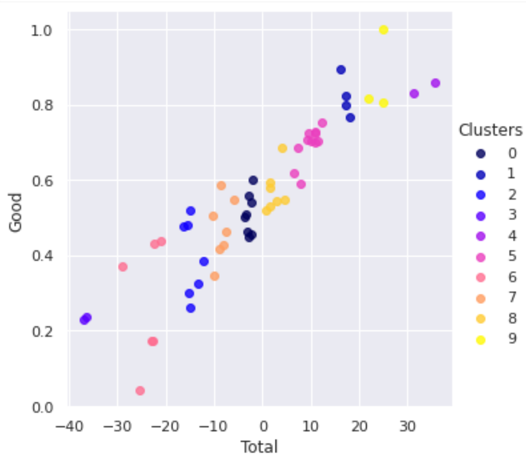
This is where I got the optimal *k* for the centroids data and performed clustering on it.

```
centroid_good = centroids.iloc[:,[0,1]]

elbow_score(centroid_good)

centroid_good_sil = sil_value(centroid_good)
centroid_good_k = centroid_good_sil.index(max(centroid_good_sil)) + 2

plt.plot([3,4,5,6,7,8,9,10], centroid_good_sil)
plt.title('Optimal k for centroid_good')
plt.xlabel('silhouette score')
plt.ylabel('k')
plt.show()

centroid_good_kmeans = KMeans(10)
centroid_good_kmeans.fit(centroid_good)
centroid_good_identified_clusters = centroid_good_kmeans.fit_predict(centroid_good)

centroid_good_data_with_clusters = centroids.copy()
centroid_good_data_with_clusters['Clusters'] = centroid_good_identified_clusters
sns.lmplot(data=centroid_good_data_with_clusters, x='Total', y='Good', hue='Clusters',
           fit_reg=False, legend=True)
```






```
<seaborn.axisgrid.FacetGrid at 0x7fb544ac0bd0>
```



The clusters show the same as the original clusters that the smaller clusters are on the edge because there are less datapoints there and the ones in the middle are bigger.

Overall, I would say that this method of preserving privacy is fine if you just want to understand the overall picture of the data and see roughly what way the data would look if it was clustered. However if you want to perform deeper analysis on the data it would not be very useful.

Conclusion

There are some interesting insights to be learned from this data when it has been clustered. There are relationships between the amount won or lost and the decks chosen, and the payload for the amount deck C was rewarding or penalising may have impacted the way that participants made their decisions. It is also harder to gain insight into the data as a whole when the privacy of each study is preserved.