Contents

Introduction

Iowa Gambling Task (IGT)

This assignment is based around the studies conducted for the Iowa Gambling Task In this study 617 healthy participants, the participants have no neurological impairments, are being assessed across 10 independent studies. During the task the partipants are are asked to select on of four cards on a screen, they are given a balance of 2000 USD. The cards they are presented with will either award them or deduct money from their balance. The purpose of the game is to win as much money as possible. In the selection of cards some decks are considered to be “good” (decks 3 and 4) and “bad” (decks 3 and 4), meaning some decks will reward more then others.

Datasets used

The data being used for this assignment is taken from many different labs ranging in number of participants and number of trials.

> **Note**
>
> The smallest study uses 15 participants while the biggest study uses 162 participants.

The datasets below used various different pay off schemes

The data is gathered from independent studies

Here is a list of the data sets being used for the assignment:

Study	Number of participants	Number of trials
Fridberg et al. 3	15	95
Horstmannb	162	100
Kjome et al. 5	19	100
Maia & McClelland 6	40	100
Premkumar et al. 7	25	100
Steingroever et al. 8	70	100
Steingroever et al. 9	57	150
Wetzels et al. 15c	41	150
Wood et al. 16	153	100
Worthy et al. 17	35	100

A link to the data is available here

Fridberg et al.3, Worth et al., amd Mai&McClelland all fall under payoff scheme no 1.

Horstmann, Steingroever er al8., Steingroever er al9., and Wetzels et al. all fall under payoff scheme no2.

The rest of the studies are parrt of payoff scheme no 3. (Kjome et al., Premkumar et al., and Wood et al.)

The description of the payoff schemes can be found (here)[https://s3-eu-west-1.amazonaws.com/ubiquity-partner-network/up/journal/jopd/ak/sup-text.pdf]

As part of this assignment, k-means clustering will be used on the data sets above. Clustering is an unsupervised machine learning problem, often used to find interesting patterns in data

K-means clustering is used to assign examples to clusters in an effort to minimize the variance within each cluster. It is used to quickly predict groupings from within an unlabeled dataset.

Our aim is to model the underlying structure of the data in order to learn from data and identify groups of data (segments/clusters) with similar characteristics/behaviours

Data Cleaning and Exploration

The purpose of this section is to clean the data sets and the explore the data that is in them. As it stands there are 12 seperate data sets and that can be very messy for analysing the data. It is easier to join all the data sets together. This will make the data analysis section a lot easier and will also make the data processing section for clustering easier also.

The first step of the data exploration phase is to import any necessary packages

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

For the purpose of this assignment their are several different data sets to be read in and explored.
There are 12 data sets to read in. They are as follows:

- **Wins:** These data sets show how much money participants win during each part of the trial
- **Losses:** These data sets show how much, “Amount_lost”, “Amount_won”h money participants lose during each part of the trial
- **Choice:** These data sets indicate what deck participants chose during each part of the trial
- **Index:** These data sets contain the name of the first author of the study that reports the data of the corresponding participant.

Importing data sets

```
win_95 = pd.read_csv('data/wi_95.csv')
win_100 = pd.read_csv('data/wi_100.csv')
win_150 = pd.read_csv('data/wi_150.csv')
```

```
loss_95 = pd.read_csv('data/lo_95.csv')
loss_100 = pd.read_csv('data/lo_100.csv')
loss_150 = pd.read_csv('data/lo_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')
```

```
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')
```

The next process is to clean the above data

Data cleaning is a very important part of the data exploration process as it will identify and remove errors for machine learning processes in the future

Data Cleaning

My first data cleaning step is to check for null values in the data sets

```
win_95.isna().sum().sum() + win_100.isna().sum().sum() + win_150.isna().sum().sum()
```

```
0
```

There are no null values in the wins data sets

```
loss_95.isna().sum().sum() + loss_100.isna().sum().sum() + loss_150.isna().sum().sum()
```

```
0
```

There are no null values in the losses data sets

```
choice_95.isna().sum().sum() + choice_100.isna().sum().sum() +
choice_150.isna().sum().sum()
```

```
0
```

There are no null values in the choices data sets

```
index_95.isna().sum().sum() + index_100.isna().sum().sum() +
index_150.isna().sum().sum()
```

```
0
```

There are no null values in the choices data sets

Due to the large number of data sets, it might make it simpler to join tables based on the number of trials

Note

Below I am making a new column for each number of trials. The new column shows to total won or lost per person.

```
total_win_95 = win_95.sum(axis=1)
total_loss_95 = loss_95.sum(axis=1)
total_95 = total_win_95 + total_loss_95

total_win_100 = win_100.sum(axis=1)
total_loss_100 = loss_100.sum(axis=1)
total_100 = total_win_100 + total_loss_100

total_win_150 = win_150.sum(axis=1)
total_loss_150 = loss_150.sum(axis=1)
total_150 = total_win_150 + total_loss_150
```

Making totals into pandas dataframes for further analysis

```
total_95 = pd.DataFrame(total_95)
total_95 = total_95.rename(columns={0: 'Total'})

total_100 = pd.DataFrame(total_100)
total_100 = total_100.rename(columns={0: 'Total'})

total_150 = pd.DataFrame(total_150)
total_150 = total_150.rename(columns={0: 'Total'})
```

Adding Study Names to the totals column

```
total_95["Study_Type"] = index_95["Study"].values
total_100["Study_Type"] = index_100["Study"].values
total_150["Study_Type"] = index_150["Study"].values
```

Adding column for number of participants in the trial

```
total_95["No_participants"] = 95
total_100["No_participants"] = 100
total_150["No_participants"] = 150
```

Adding total won and total lost per player over the course of the task

```
total_95["Amount_won"] = win_95.sum(axis=1)
total_95["Amount_lost"] = loss_95.sum(axis=1)

total_100["Amount_won"] = win_100.sum(axis=1)
total_100["Amount_lost"] = loss_100.sum(axis=1)

total_150["Amount_won"] = win_150.sum(axis=1)
total_150["Amount_lost"] = loss_150.sum(axis=1)
```

Adding choice of cards into each data frame

Pandas series, returns a Series containing counts of unique values.

```
total_choice_95 = choice_95.apply(pd.Series.value_counts, axis=1)

total_choice_100 = choice_100.apply(pd.Series.value_counts, axis=1)

total_choice_150 = choice_150.apply(pd.Series.value_counts, axis=1)

total_choices = total_choice_95.append(total_choice_100)
total_choices = total_choices.append(total_choice_150)
```

```
total_choices.shape
```

```
(617, 4)
```

Showing the number of columns in the new datasets shows that no rows have been lost

```
total_95.shape[0] +total_150.shape[0] + total_100.shape[0]
```

```
617
```

Showing the number of columns in the new datasets shows that no rows have been lost

```
total_95.shape[0] +total_150.shape[0] + total_100.shape[0]
```

```
617
```

Joining all the totals datasets together

```
totals = total_95.append(total_100)
totals = totals.append(total_150)
```

```
totals.shape
```

```
(617, 5)
```

The last step of the data cleaning process is to join thte totals dataframe to the total_choices data frame

```
all_data = pd.concat([totals, total_choices], axis=1)
all_data = all_data.fillna(0)
```

```
all_data.shape
```

```
(617, 9)
```

Data Analysis/Exploration

Analysing the wins and losses for each number of trials

Seaborn and Matplotlib will be used to visualise the data

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
all_data.head()
```

	Total	Study_Type	No_participants	Amount_won	Amount_lost	1	2	3
Subj_1	1150	Fridberg	95	5800	-4650	12.0	9.0	3.0
Subj_2	-675	Fridberg	95	7250	-7925	24.0	26.0	12.0
Subj_3	-750	Fridberg	95	7100	-7850	12.0	35.0	10.0
Subj_4	-525	Fridberg	95	7000	-7525	11.0	34.0	12.0
Subj_5	100	Fridberg	95	6450	-6350	10.0	24.0	15.0

Descriptive statistics for the Total column.

```
all_data["Total"].describe()
```

```
count     617.000000
mean     -156.831442
std      1251.585443
min     -4250.000000
25%     -1000.000000
50%      -170.000000
75%       650.000000
max      3750.000000
Name: Total, dtype: float64
```

```
all_data[all_data["Total"] < 0].shape
```

```
(333, 9)
```

333 of the participants of these experiments lost money that is approximately 54% of all the participants.

This is initally surprising considering all the participants are deemed to be “healthy”

Healthy patients should be able to find the most advantageous decks (3 and 4) and stick to them

Now I will check a comparison of wins vs losses for each type of study (95 participants, 100 participants and 150 participants)

Analysis for studies containing 95 participants

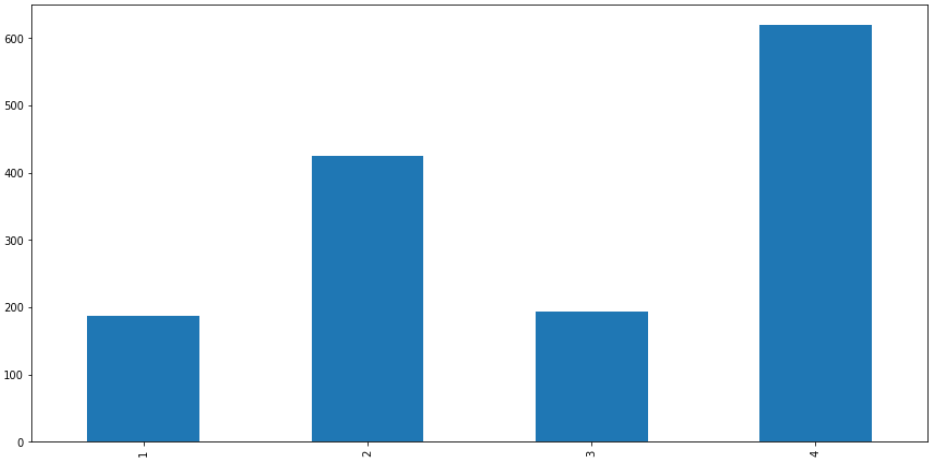
```
all_data_95 = all_data[all_data["No_participants"] == 95]
```

```
all_data_95[[1,2,3,4]].sum()
```

```
1    187.0
2    425.0
3    194.0
4    619.0
dtype: float64
```

```
all_data_95[[1,2,3,4]].sum().plot.bar(figsize=(15,7.5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4885b14fd0>
```




```
all_data_95.head(1)
```

	Total	Study_Type	No_participants	Amount_won	Amount_lost	1	2	3	
Subj_1	1150	Fridberg	95	5800	-4650	12.0	9.0	3.0	71.

```
all_data_95["Study_Type"].value_counts()
```

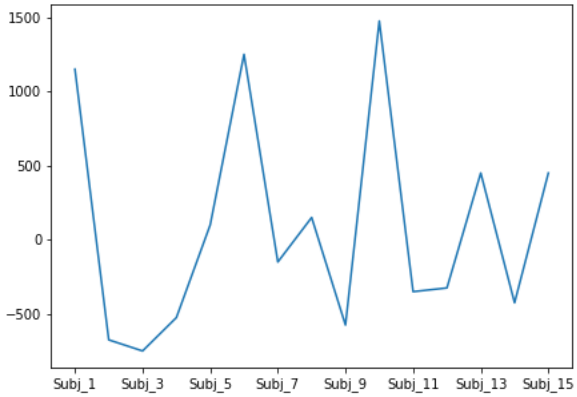
```
Fridberg    15
Name: Study_Type, dtype: int64
```

Note

Fridberg is the only study that had experiments with participants only having 95 trials.

```
all_data_95["Total"].plot(figsize=(7,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f487bae7d50>
```



As there are only 15 participants it is hard to find anything concrete about the amounts won and lost in these trials from the visualisation

```
all_data_95["Total"].describe()
```

```
count      15.000000
mean       83.333333
std       729.460825
min      -750.000000
25%      -475.000000
50%      -150.000000
75%       450.000000
max      1475.000000
Name: Total, dtype: float64
```

```
all_data_95[all_data_95["Total"] <0].shape
```

```
(8, 9)
```

8 of the participants in this trial failed to make any money

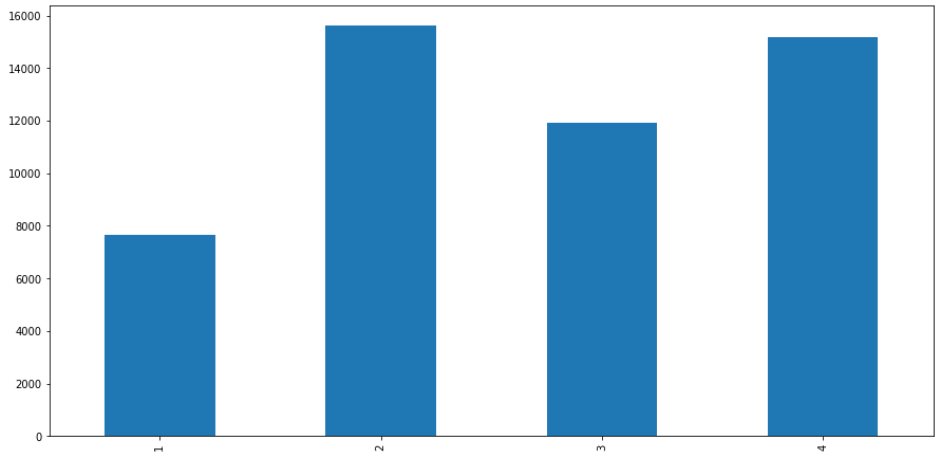
With participants in this study with 95 trials it is difficult to come to any conclusions due to the fact that there are simply not enough participants.

Analysis for studies containing 100 participants

```
all_data_100 = all_data[all_data["No_participants"] == 100]
```

```
all_data_100[[1,2,3,4]].sum().plot.bar(figsize=(15,7.5))
```

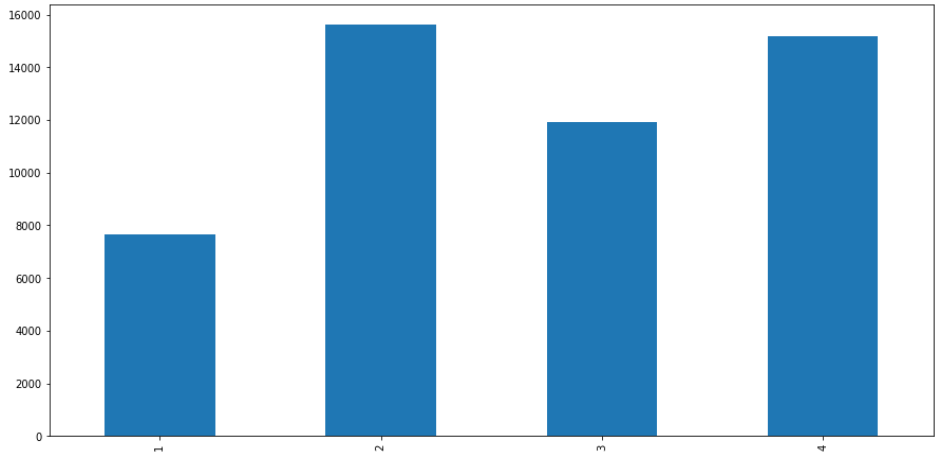
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f487baf83d0>
```



compare total won and lost for each study

```
all_data_100[[1,2,3,4]].sum().plot.bar(figsize=(15,7.5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f487b87f190>
```




```
all_data_100.head(1)
```

	Total	Study_Type	No_participants	Amount_won	Amount_lost	1	2	3
Subj_1	-1800	Horstmann	100	8150	-9950	21.0	42.0	15.0

```
all_data_100["Total"].describe()
```

```
count     504.000000
mean     -266.994048
std      1178.585955
min     -4250.000000
25%     -1050.000000
50%      -300.000000
75%       550.000000
max      3570.000000
Name: Total, dtype: float64
```

```
all_data_100["Study_Type"].value_counts()
```

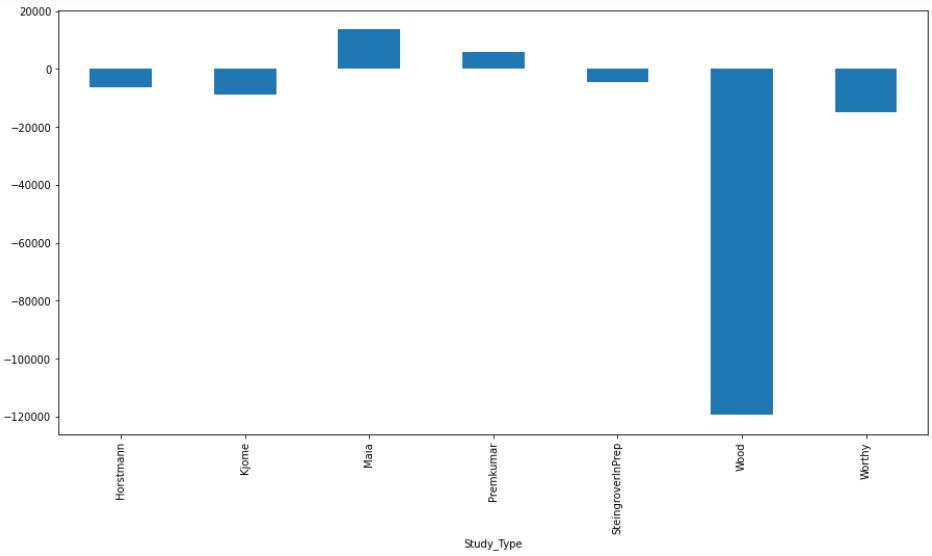
```
Horstmann            162
Wood                 153
SteingroverInPrep     70
Maia                  40
Worthy                35
Premkumar             25
Kjome                 19
Name: Study_Type, dtype: int64
```

```
all_data_100.groupby("Study_Type")["Total"].sum()
```

```
Study_Type
Horstmann             -6200
Kjome                 -8750
Maia                  13600
Premkumar              5995
SteingroverInPrep     -4700
Wood                -119410
Worthy               -15100
Name: Total, dtype: int64
```

```
all_data_100.groupby("Study_Type")["Total"].sum().plot.bar(figsize=(15,7.5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f487b802690>
```



Look into Maia study and see why it was the only one to make any money

Maia was in the payoff 1 scheme

```
all_data_100[all_data_100["Study_Type"] == "Wood"]
```

	Total	Study_Type	No_participants	Amount_won	Amount_lost	1	2	3
Subj_317	-320	Wood	100	8080	-8400	19.0	30.0	25.0
Subj_318	-1030	Wood	100	8870	-9900	17.0	43.0	20.0
Subj_319	-1850	Wood	100	8450	-10300	20.0	35.0	21.0
Subj_320	-775	Wood	100	8150	-8925	22.0	28.0	24.0
Subj_321	-1600	Wood	100	8250	-9850	16.0	35.0	19.0
...	...	...	...	...	...	...	...	..
Subj_465	2400	Wood	100	6600	-4200	5.0	10.0	25.0
Subj_466	-1035	Wood	100	8190	-9225	25.0	27.0	23.0
Subj_467	-2375	Wood	100	8975	-11350	12.0	47.0	19.0
Subj_468	1390	Wood	100	6940	-5550	5.0	16.0	19.0
Subj_469	-2235	Wood	100	8290	-10525	21.0	32.0	29.0

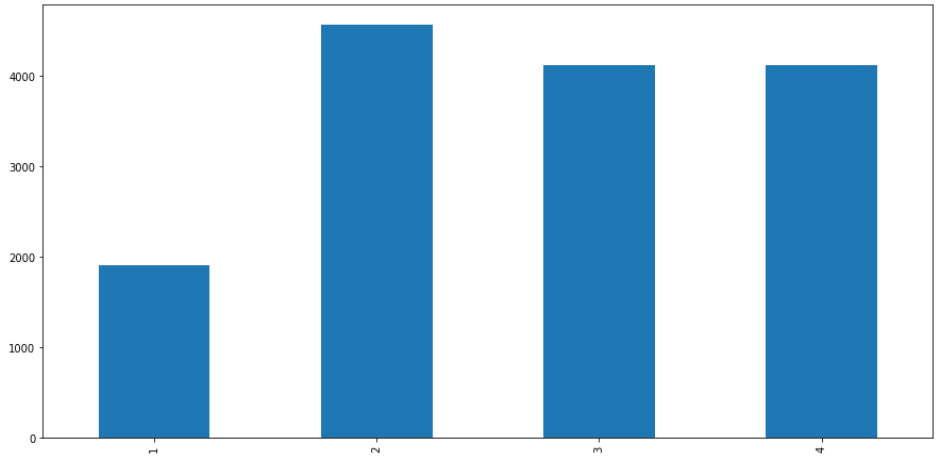
153 rows × 9 columns

Analysis for studies containing 150 participants

```
all_data_150 = all_data[all_data["No_participants"] == 150]
```

```
all_data_150[[1,2,3,4]].sum().plot.bar(figsize=(15,7.5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f487b78bf10>
```



Both of these studies were in the payoff 2 scheme

Compare total won and lost for each study

```
all_data_150["Study_Type"].value_counts()
```

```
Steingroever2011    57
Wetzels             41
Name: Study_Type, dtype: int64
```

```
all_data_150.groupby("Study_Type")["Total"].sum()
```

```
Study_Type
Steingroever2011    12550
Wetzels             24000
Name: Total, dtype: int64
```

```
all_data_150["Total"].describe()
```

```
count      98.000000
mean      372.959184
std      1520.668063
min     -4200.000000
25%      -500.000000
50%       350.000000
75%      1450.000000
max      3750.000000
Name: Total, dtype: float64
```

```
all_data_150.shape
```

```
(98, 9)
```

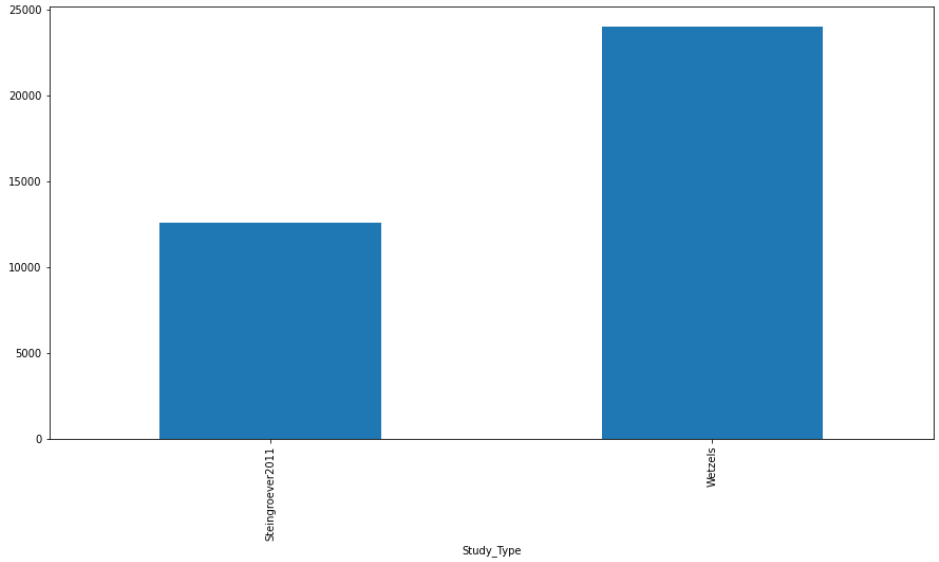
```
len(all_data_150[all_data_150["Total"] > 0])
```

```
62
```

In the case of 150 participants 62 out of 98 partipants made money.

```
all_data_150.groupby("Study_Type")["Total"].sum().plot.bar(figsize=(15,7.5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f487bd1fbd0>
```



Comparing outputs for all 3 trials

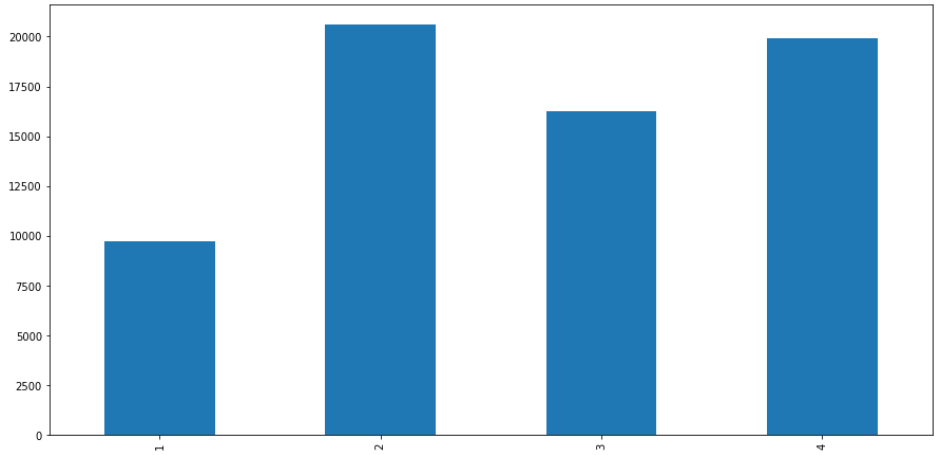
Card deck selection analysis

```
all_data[[1,2,3,4]].sum()
```

```
1     9757.0
2    20601.0
3    16240.0
4    19927.0
dtype: float64
```

```
all_data[[1,2,3,4]].sum().plot.bar(figsize=(15,7.5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f487bcfce90>
```



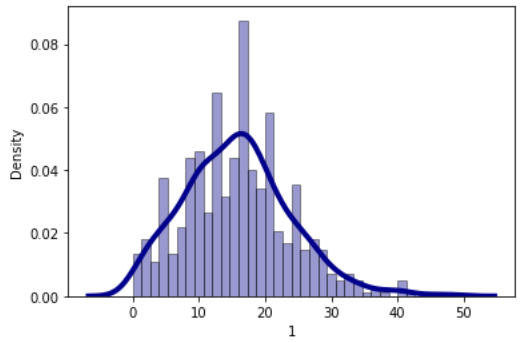
Note

The distribution of the card selections below is for my understanding. I will need to standardise the card selection figures in the data preparation phase

```
sns.distplot(all_data[1], hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

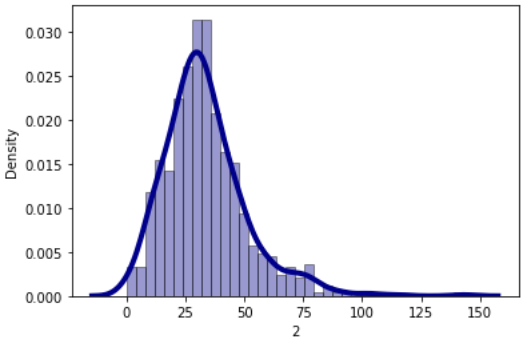
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4885d69a90>
```




```
sns.distplot(all_data[2], hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

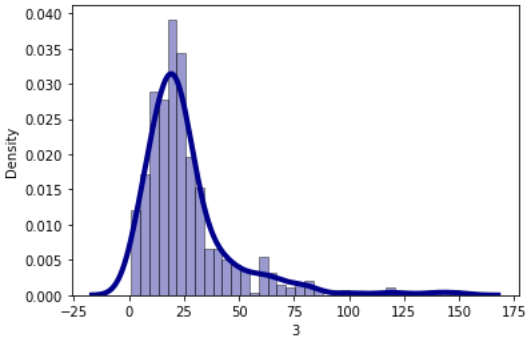
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f487bc018d0>
```




```
sns.distplot(all_data[3], hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

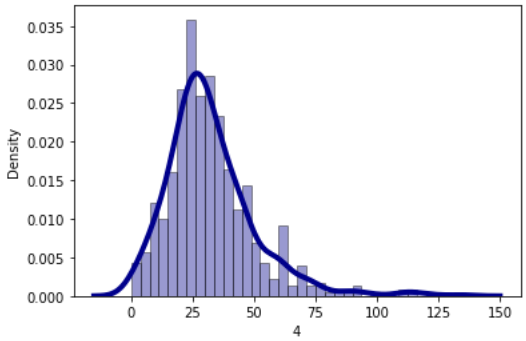
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f487bb7e210>
```




```
sns.distplot(all_data[4], hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f487b67ea90>
```

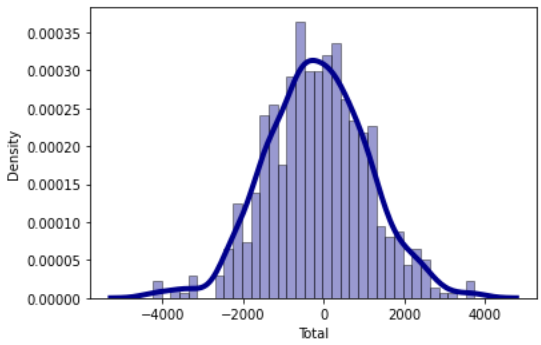


Distribution of totals

```
sns.distplot(all_data['Total'], hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f48775798d0>
```




```
all_data["Total"].describe()
```

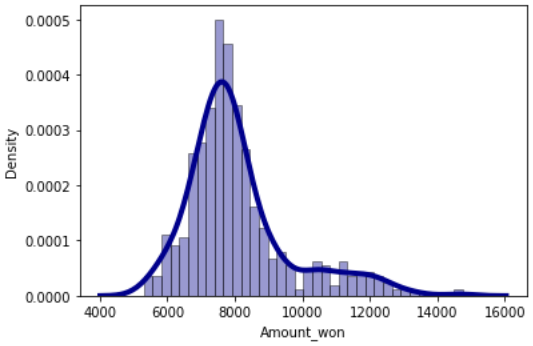
```
count     617.000000
mean     -156.831442
std      1251.585443
min     -4250.000000
25%     -1000.000000
50%      -170.000000
75%       650.000000
max      3750.000000
Name: Total, dtype: float64
```

From looking at these figures it might be interesting to view these figures in the clustering section along with the study they were apart of

```
sns.distplot(all_data['Amount_won'], hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

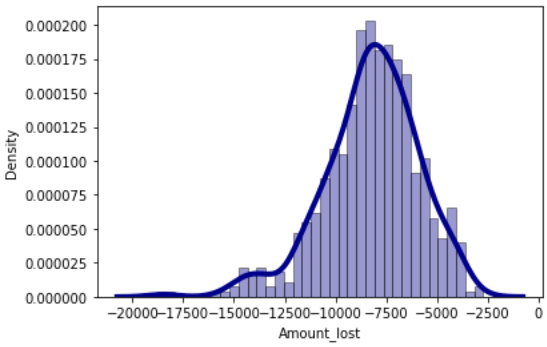
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f48773c2c50>
```




```
sns.distplot(all_data['Amount_lost'], hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f48772d2b50>
```



From the distribution plot the data for the total won/lost seems to follow a normal distribution

```
all_data["Total"].describe()
```

```
count     617.000000
mean     -156.831442
std      1251.585443
min     -4250.000000
25%     -1000.000000
50%      -170.000000
75%       650.000000
max      3750.000000
Name: Total, dtype: float64
```

```
all_data.isna().sum().sum()
```

```
0
```

Data Analysis comments

During the data analysis process I was expecting to see some concrete evidence from the studies. Considering all the participants were all deemed to be “healthy” you would assume that they would soon realise the most advantageous were decks 3 and 4. Although it is obvious from the analysis above that this was not the case. It is clear that participants found deck 1 to be the worst deck due to the fact that it was selected the least, it was selected a total of 14.67% of the time. The next least chosen deck was deck no 3. This is surprising due to the fact that it is one of the supposed “good” decks. In my opinion the most surprising aspect of the data analysis was deck two being selected the most of all the decks.

From my initial data we saw that of all the decks picked, deck no 2 was picked the most. It might be interesting to do some clustering on deck no2 for totals compared to the other decks. Along with deck 2, I would also like to perform cluster analysis on the other decks to see if there are any similarities.

The final step is to export the cleaned data set for data preparation

```
all_data.to_csv('data/all_data.csv')
```

Data preparation for clustering

For clustering using the k-means method we will need to follow a few steps:

- **Standardise the data:** The process of converting an actual range of values into a standard range of values
- **Find a Similarity Measure:**
- **Interpret Results:**

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
```

Importing cleaned data

```
all_data = pd.read_csv('data/all_data.csv')
all_data=all_data.drop(["Unnamed: 0"], axis=1) # Drop Unnamed: 0 column
all_data.head()

df = pd.read_csv('data/all_data.csv')
df=df.drop(["Unnamed: 0"], axis=1) # Drop Unnamed: 0 column
```

Changing categorical data entries can be done by One-hot encoding.

Label encoding is a method during data preparation for converting categorical data variables so they can be provided to machine learning algorithims to improve predictions

LabelEncoder() is a data manipulation function used to convert categorical data into indicator variables

> **Note**
>
> Machine learning models require all input and output variables to be numeric

It is impossible to do k-means clustering on a categorical variable

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
df["Study_Type"] = labelencoder.fit_transform(df["Study_Type"])
df.head()
```

	Total	Study_Type	No_participants	Amount_won	Amount_lost	1	2	3	4
0	1150	0	95	5800	-4650	12.0	9.0	3.0	71.0
1	-675	0	95	7250	-7925	24.0	26.0	12.0	33.0
2	-750	0	95	7100	-7850	12.0	35.0	10.0	38.0
3	-525	0	95	7000	-7525	11.0	34.0	12.0	38.0
4	100	0	95	6450	-6350	10.0	24.0	15.0	46.0

```
df["Study_Type"].value_counts()
```

```
1    162
8    153
6     70
5     57
7     41
3     40
9     35
4     25
2     19
0     15
Name: Study_Type, dtype: int64
```

I opted for LabelEncoder as opposed to One-Hot Encoder to reduce the number of demensions being used in the data set.

This will be important for clustering due to the fact that k-means clustering can suffer from the curse of dimensionality

```
all_data.head()
```

	Total	Study_Type	No_participants	Amount_won	Amount_lost	1	2	3	4
0	1150	Fridberg	95	5800	-4650	12.0	9.0	3.0	71.0
1	-675	Fridberg	95	7250	-7925	24.0	26.0	12.0	33.0
2	-750	Fridberg	95	7100	-7850	12.0	35.0	10.0	38.0
3	-525	Fridberg	95	7000	-7525	11.0	34.0	12.0	38.0
4	100	Fridberg	95	6450	-6350	10.0	24.0	15.0	46.0

Standardising data

As for K-means, often it is not sufficient to normalize only mean. One normalizes data equalizing variance along different features as K-means is sensitive to variance in data, and features with larger variance have more emphasis on result. So for K-means, I would recommend using StandardScaler for data preprocessing.

Don't forget also that k-means results are sensitive to the order of observations, and it is worth to run algorithm several times, shuffling data in between, averaging resulting clusters and running final evaluations with those averaged clusters centers as starting points.

Standardising data Test

```
scaler = preprocessing.StandardScaler()
segmentation_std = scaler.fit_transform(df)
```

```
segmentation_std
```

```
array([[ 1.04498799, -1.60707299, -0.69883592, ..., -1.38690362,
        -1.0891972 ,  2.13175282],
       [-0.41434565, -1.60707299, -0.69883592, ..., -0.42018166,
        -0.66885443,  0.03874291],
       [-0.47431826, -1.60707299, -0.69883592, ...,  0.09161232,
        -0.76226394,  0.31413895],
       ...,
       [ 1.28487845,  0.74499351,  2.29926737, ..., -0.81824364,
         0.49876436,  2.40714887],
       [ 1.08496973,  0.74499351,  2.29926737, ..., -0.19271767,
         0.82569762,  1.03016866],
       [-1.31393487,  0.74499351,  2.29926737, ...,  4.01536616,
        -0.94908294, -0.18157392]])
```

The standardised data is now stored in an array. I will convert it back to a pandas dataframe,

```
df_standard = pd.DataFrame(segmentation_std, columns=['Total', 'Study_Type',
'No_participants', 'Amount_won', 'Amount_lost', '1', '2', '3', '4'])
df_standard
```

	Total	Study_Type	No_participants	Amount_won	Amount_lost	1	
0	1.044988	-1.607073	-0.698836	-1.471413	1.525683	-0.477115	-1.386
1	-0.414346	-1.607073	-0.698836	-0.538613	0.135451	1.024186	-0.420
2	-0.474318	-1.607073	-0.698836	-0.635110	0.167288	-0.477115	0.091
3	-0.294400	-1.607073	-0.698836	-0.699441	0.305250	-0.602224	0.034
4	0.205371	-1.607073	-0.698836	-1.053262	0.804036	-0.727332	-0.533
...	...	...	...	...	...	...	
612	0.365298	0.744994	2.299267	2.613607	-1.530705	1.024186	2.025
613	1.844623	0.744994	2.299267	0.780173	0.464437	-1.352875	-0.135
614	1.284878	0.744994	2.299267	0.812338	0.146063	0.273535	-0.818
615	1.084970	0.744994	2.299267	1.391318	-0.342110	1.149295	-0.192
616	-1.313935	0.744994	2.299267	3.321249	-2.889100	-0.602224	4.015

617 rows × 9 columns

Exporting Data to CSV file

```
df_standard.to_csv('data/normalise.csv')
```

K-Means Clustering

From my first analysis of the data in data exploration section, I was interested to see how differnet card selections and different studies related.

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
%matplotlib inline
```

Read in normalised data

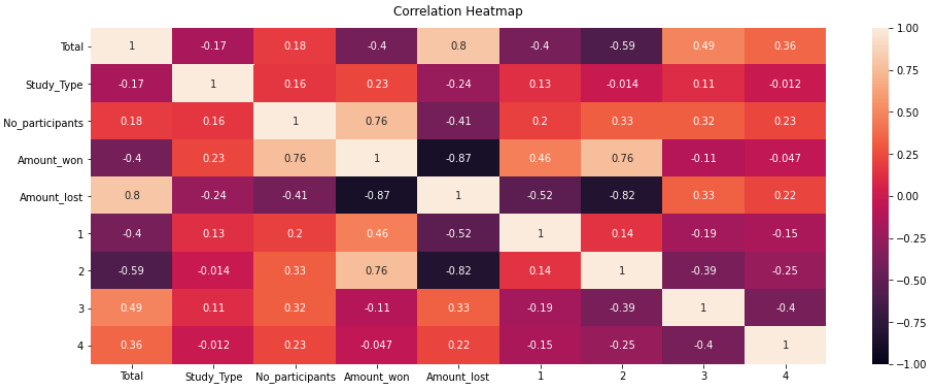
```
df = pd.read_csv('data/normalise.csv')
df=df.drop(["Unnamed: 0"], axis=1) # Drop Unnamed: 0 column
df.head()
```

	Total	Study_Type	No_participants	Amount_won	Amount_lost	1	
0	1.044988	-1.607073	-0.698836	-1.471413	1.525683	-0.477115	-1.38690
1	-0.414346	-1.607073	-0.698836	-0.538613	0.135451	1.024186	-0.42018
2	-0.474318	-1.607073	-0.698836	-0.635110	0.167288	-0.477115	0.09161
3	-0.294400	-1.607073	-0.698836	-0.699441	0.305250	-0.602224	0.03474
4	0.205371	-1.607073	-0.698836	-1.053262	0.804036	-0.727332	-0.53391

Note

This correlation matrix is used to find highly correlated variables

```
plt.figure(figsize=(16, 6))
heatmap = sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True)
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':12}, pad=12);
```



PCA

To avoid the curse of dimensionality. The curse of dimensionality is the problem caused by the exponential increase in volume associated with adding extra dimensions to Euclidean space. To avoid this we can perform Principal Component analysis (PCA) to our data frame. This will leave use with an array of components.

PCA is believed to imporve the performance for cluster analysis

By reducing the number of features we are increasing the performance of our algorithm

For the purpose of this assignment I will select 2 features to be used for clustering

```
pca = PCA(2)
df_tester = pca.fit_transform(df)
```

```
df_tester
```

```
array([[-2.98565138, -0.90534442],
       [-0.2845081 , -1.42223416],
       [-0.54699829, -1.50399143],
       ...,
       [-0.20959764,  3.05643034],
       [ 0.99635261,  3.05873358],
       [ 5.87050839,  1.25408754]])
```

The newly optained PCA scores will be incorporated in the the K-means algorithm

The next step of the clustering process is to determine a value for K. This can be done by:

- **The Elbow Method**
- **Silhouette score**

These two methods will be used to find the best value for K to use for clustering

Methods for finding K Value

```
# Elbow method and silhouette method
```

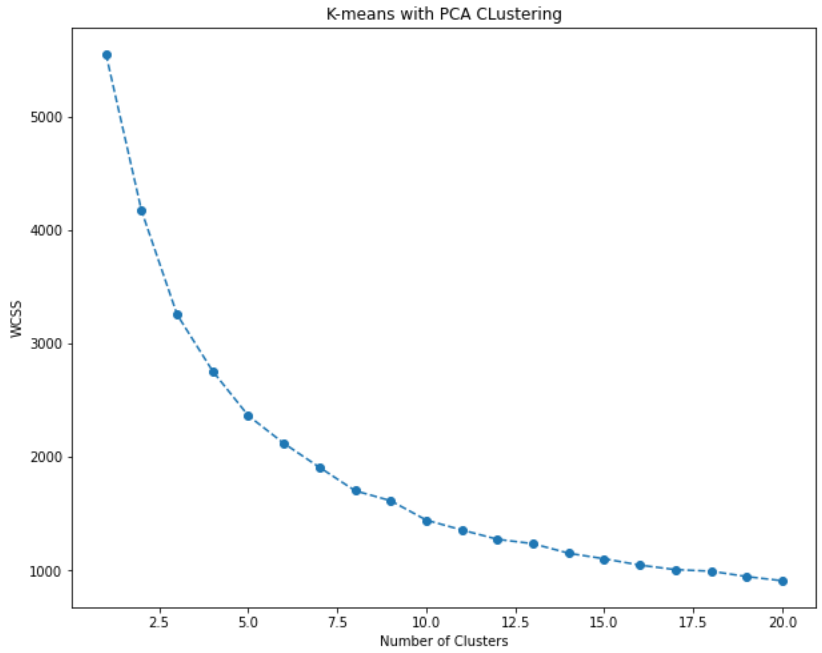
When the distortions are plotted and the plot looks like an arm then the "elbow"(the point of inflection on the curve) is the best value of k.

The formula for the Elbow method can be seen here:
$$SumSquaredErrors = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

The formula for the Silhouette Score method can be seen here:
$$s_i = \frac{b_i - a_i}{max(b_i, a_i)}$$

```
wcss=[]
for i in range(1,21):
    k_means_pca=KMeans(n_clusters=i, init="k-means++", random_state=42)
    k_means_pca.fit(df)
    wcss.append(k_means_pca.inertia_)
```

```
plt.figure(figsize=(10,8))
plt.plot(range(1,21), wcss, marker="o", linestyle = "--")
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.title("K-means with PCA CLustering")
plt.show()
```




```
for n in range(2, 21):
    km = KMeans(n_clusters=n)
    km.fit_predict(df)
    score = silhouette_score(df, km.labels_, metric='euclidean')
    print('N = ' + str(n) + ' Silhouette Score: %.3f' % score)
```

```
N = 2 Silhouette Score: 0.231
```

```
N = 3 Silhouette Score: 0.263
```

```
N = 4 Silhouette Score: 0.267
N = 5 Silhouette Score: 0.283
```

```
N = 6 Silhouette Score: 0.284
```

```
N = 7 Silhouette Score: 0.293
```

```
N = 8 Silhouette Score: 0.303
```

```
N = 9 Silhouette Score: 0.310
```

```
N = 10 Silhouette Score: 0.287
```

```
N = 11 Silhouette Score: 0.281
```

```
N = 12 Silhouette Score: 0.243
```

```
N = 13 Silhouette Score: 0.256
```

```
N = 14 Silhouette Score: 0.235
```

```
N = 15 Silhouette Score: 0.249
```

```
N = 16 Silhouette Score: 0.234
```

```
N = 17 Silhouette Score: 0.241
```

```
N = 18 Silhouette Score: 0.252
```

```
N = 19 Silhouette Score: 0.231
```

```
N = 20 Silhouette Score: 0.226
```

From looking at the Elbow method, paired with the silhouette score we come to a conclusion about the number of clusters to run.

From the various methods above the optimal value for K to us used for clustering will be 4

Implementation of the K-means Clustering Process

```
k_means_pca = KMeans(n_clusters=4, init="k-means++", random_state=42)
```

```
k_means_pca.fit(df_tester)
```

```
KMeans(n_clusters=4, random_state=42)
```

```
df_segm_pca_kmeans = pd.concat([df.reset_index(drop=True), pd.DataFrame(df_tester)],
axis=1)
df_segm_pca_kmeans.columns.values[-2:] = ["Component 1", "Component 2"]

df_segm_pca_kmeans["Segment K-means PCA"] = k_means_pca.labels_
```

```
df_segm_pca_kmeans.head()
```

	Total	Study_Type	No_participants	Amount_won	Amount_lost	1	2
0	1.044988	-1.607073	-0.698836	-1.471413	1.525683	-0.477115	-1.38690
1	-0.414346	-1.607073	-0.698836	-0.538613	0.135451	1.024186	-0.42018
2	-0.474318	-1.607073	-0.698836	-0.635110	0.167288	-0.477115	0.09161
3	-0.294400	-1.607073	-0.698836	-0.699441	0.305250	-0.602224	0.03474
4	0.205371	-1.607073	-0.698836	-1.053262	0.804036	-0.727332	-0.53391

```
df_segm_pca_kmeans["Segment K-means PCA"].value_counts()
```

```
0    308
1    208
2     61
3     40
Name: Segment K-means PCA, dtype: int64
```

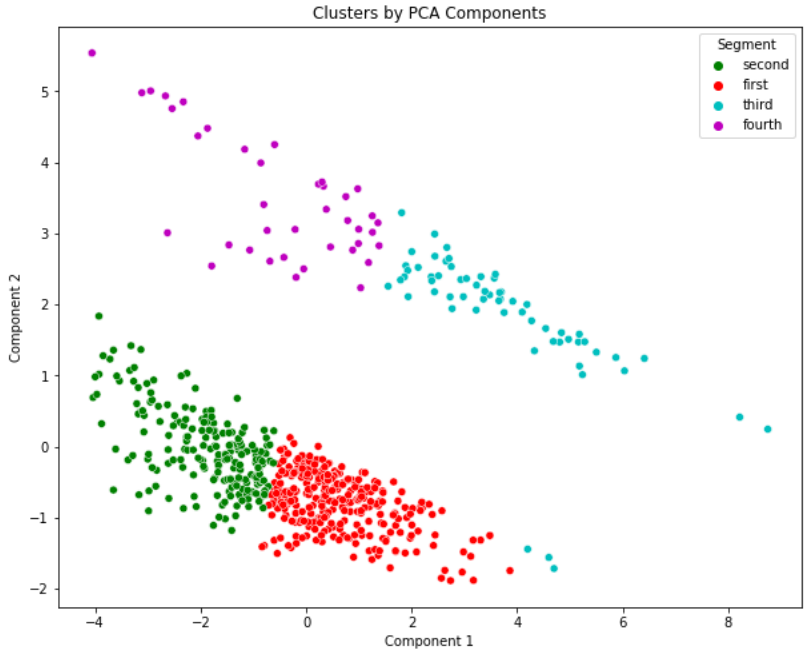
```
df_segm_pca_kmeans["Segment"] = df_segm_pca_kmeans["Segment K-means
PCA"].map({0:"first", 1:"second", 2:"third", 3:"fourth"})
```

```
df_segm_pca_kmeans["Segment"].value_counts()
```

```
first     308
second    208
third      61
fourth     40
Name: Segment, dtype: int64
```

```
x_axis = df_segm_pca_kmeans["Component 1"]
y_axis = df_segm_pca_kmeans["Component 2"]
plt.figure(figsize = (10, 8))
sns.scatterplot(x_axis, y_axis, hue = df_segm_pca_kmeans["Segment"], palette=["g", "r",
"c", "m"])
plt.title("Clusters by PCA Components")
plt.show()
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



We can now observe the seperate clusters

With the Clusters seperated we are able to visualise almost all the entire data set

Before showing cluster analysis, change study types back to their original names

```
df_original = pd.read_csv('data/all_data.csv')
df_original=df_original.drop(["Unnamed: 0"], axis=1) # Drop Unnamed: 0 column
df_original.head(1)
```

	Total	Study_Type	No_participants	Amount_won	Amount_lost	1	2	3	4
0	1150	Fridberg	95	5800	-4650	12.0	9.0	3.0	71.0

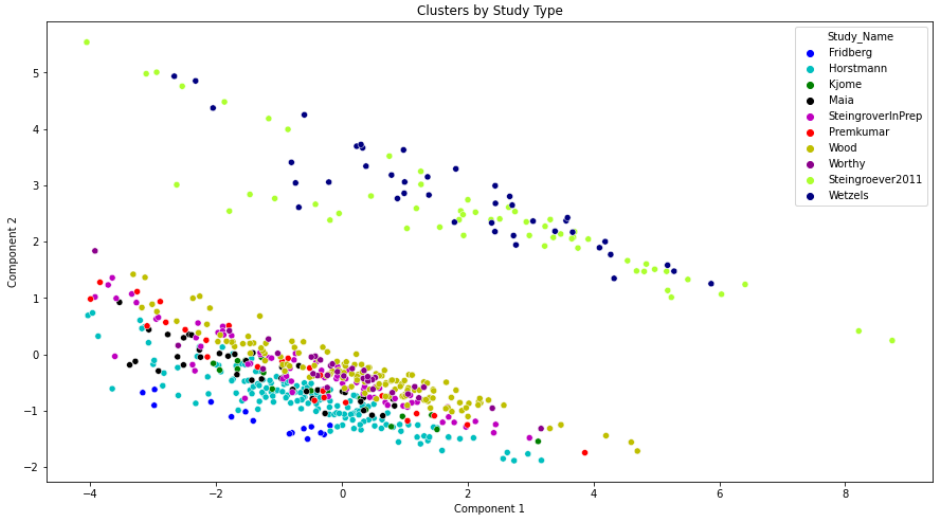
```
df_segm_pca_kmeans["Study_Name"] = df_original["Study_Type"]
df_segm_pca_kmeans["Study_Name"]
```

```
0      Fridberg
1      Fridberg
2      Fridberg
3      Fridberg
4      Fridberg
         ...
612     Wetzels
613     Wetzels
614     Wetzels
615     Wetzels
616     Wetzels
Name: Study_Name, Length: 617, dtype: object
```

Below is a cluster graph for the studies involved in these experiments

```
x_axis = df_segm_pca_kmeans["Component 1"]
y_axis = df_segm_pca_kmeans["Component 2"]
plt.figure(figsize = (15, 8))
sns.scatterplot(x_axis, y_axis, hue = df_segm_pca_kmeans["Study_Name"], palette=["b",
"c", "g", "k", "m", "r", "y", "darkmagenta", "greenyellow", "navy"])
plt.title("Clusters by Study Type")
plt.show()
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



The first observation to note is that the Steingroever2011 study and Wetzels study are out on their own in the scatter plot.

They are both part of segments 3 and 4. Both of these studies contained 150 participants.

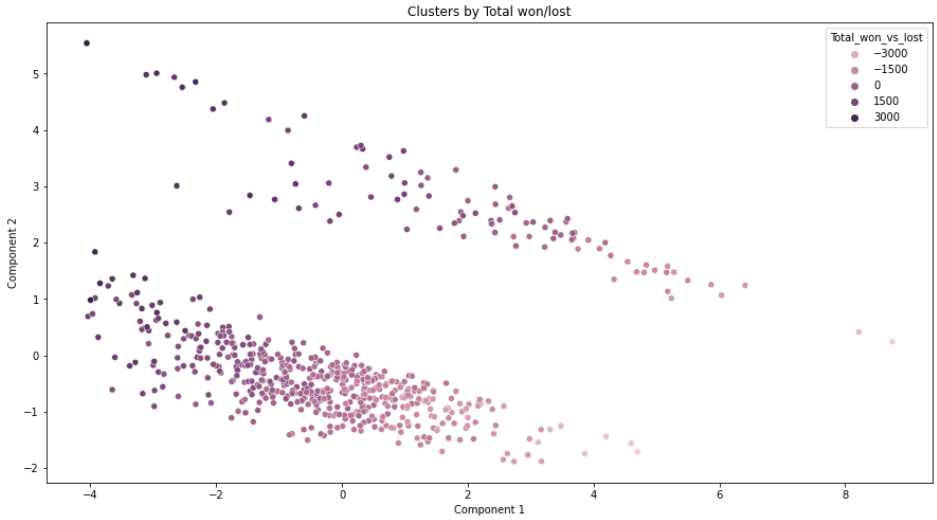
From looking at the way the Studies have clustered it is clear to see that they all follow the same pattern

Cluster analysis for Total won/lost

```
df_segm_pca_kmeans["Total_won_vs_lost"] = df_original["Total"]
```

```
x_axis = df_segm_pca_kmeans["Component 1"]
y_axis = df_segm_pca_kmeans["Component 2"]
plt.figure(figsize = (15, 8))
sns.scatterplot(x_axis, y_axis, hue = df_segm_pca_kmeans["Total_won_vs_lost"])
plt.title("Clusters by Total won/lost")
plt.show()
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



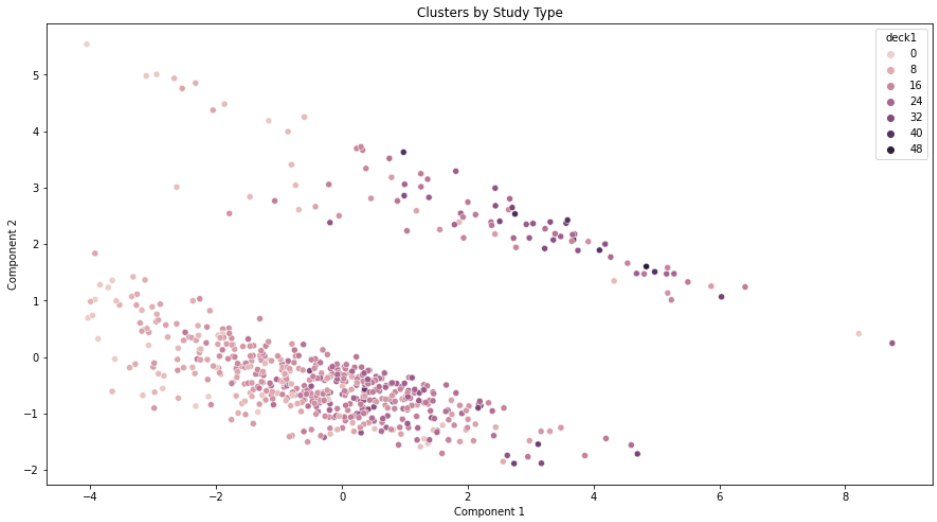
From analysing this scatter plot it is clear to see that there is a reduction in total won by the participant as component 1 gets bigger

Cluster analysis for deck selection

```
df_segm_pca_kmeans["deck1"] = df_original["1"]
df_segm_pca_kmeans["deck2"] = df_original["2"]
df_segm_pca_kmeans["deck3"] = df_original["3"]
df_segm_pca_kmeans["deck4"] = df_original["4"]
```

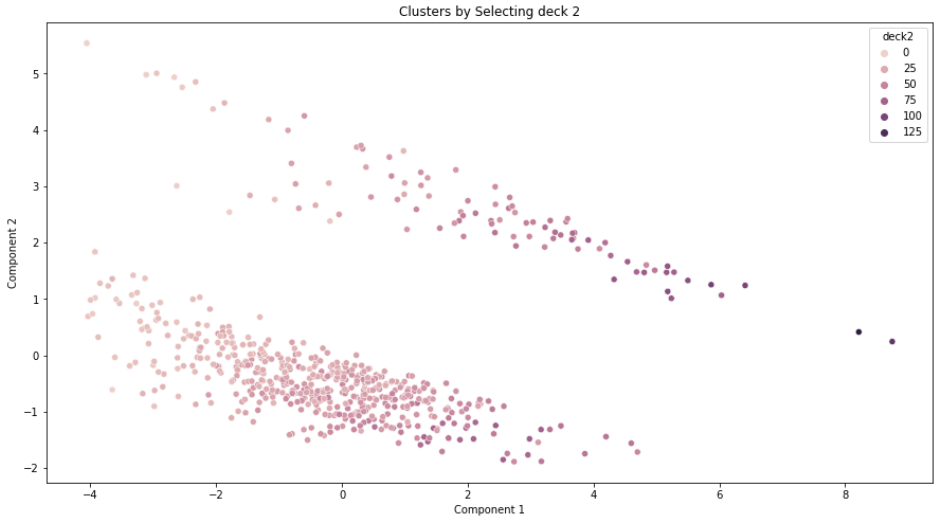
```
x_axis = df_segm_pca_kmeans["Component 1"]
y_axis = df_segm_pca_kmeans["Component 2"]
plt.figure(figsize = (15, 8))
sns.scatterplot(x_axis, y_axis, hue = df_segm_pca_kmeans["deck1"])
plt.title("Clusters by Study Type")
plt.show()
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



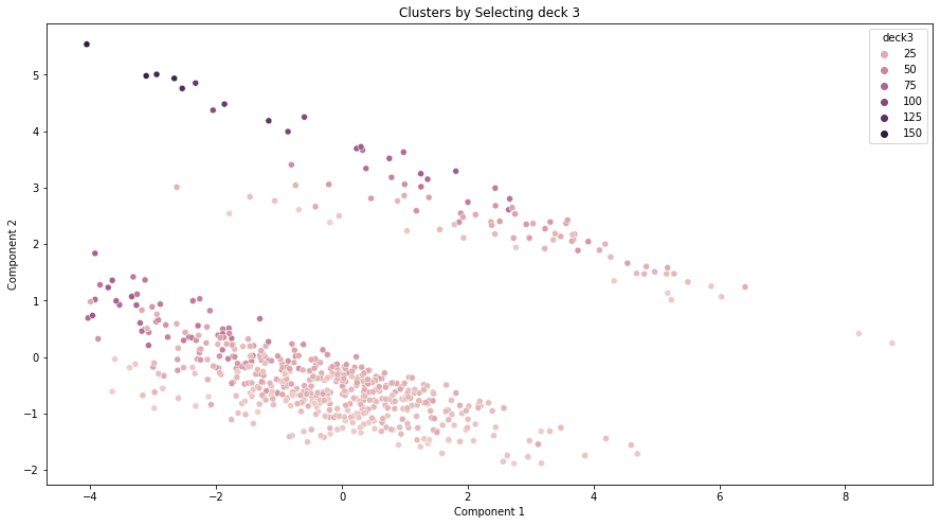

```
x_axis = df_segm_pca_kmeans["Component 1"]
y_axis = df_segm_pca_kmeans["Component 2"]
plt.figure(figsize = (15, 8))
sns.scatterplot(x_axis, y_axis, hue = df_segm_pca_kmeans["deck2"])
plt.title("Clusters by Selecting deck 2")
plt.show()
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



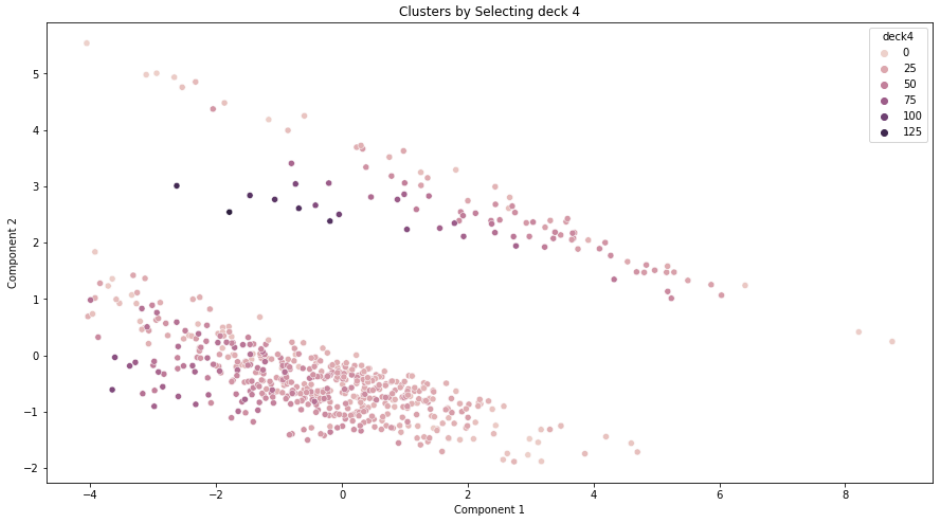

```
x_axis = df_segm_pca_kmeans["Component 1"]
y_axis = df_segm_pca_kmeans["Component 2"]
plt.figure(figsize = (15, 8))
sns.scatterplot(x_axis, y_axis, hue = df_segm_pca_kmeans["deck3"])
plt.title("Clusters by Selecting deck 3")
plt.show()
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```




```
x_axis = df_segm_pca_kmeans["Component 1"]
y_axis = df_segm_pca_kmeans["Component 2"]
plt.figure(figsize = (15, 8))
sns.scatterplot(x_axis, y_axis, hue = df_segm_pca_kmeans["deck4"])
plt.title("Clusters by Selecting deck 4")
plt.show()
```

```
/home/michael/anaconda3/lib/python3.7/site-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12,
the only valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



This is a cluster analysis for deck 4. This is the supposed best deck so it is intersting to see that the most successful instances for each study don't pick deck four in comparison to deck 3

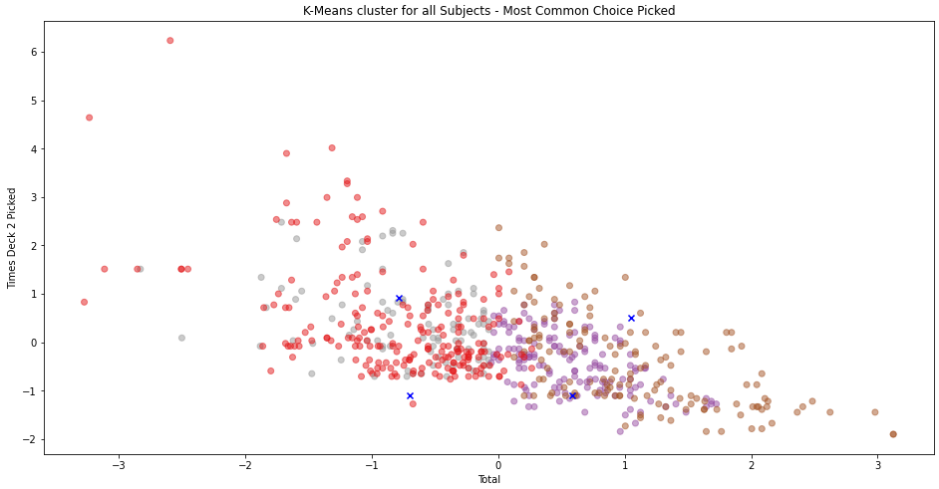
The study most likely to pick deck 4 the most is Steingrover 2011 based on the cluster analysis above.

This is an example of a cluster made without using the Principal Component analysis.

It is evident that the groups are all mixed together, therefore making it harder to determine any proper findings from the clusters.

```
kmeans_margin_standard = KMeans(n_clusters=4).fit(df[["Total", "Study_Type"]])
centroids_betas_standard = kmeans_margin_standard.cluster_centers_
```

```
plt.figure(figsize=(16,8))
plt.scatter(df['Total'], df['2'], c= kmeans_margin_standard.labels_, cmap = "Set1",
alpha=0.5)
plt.scatter(centroids_betas_standard[:, 0], centroids_betas_standard[:, 1], c='blue',
marker='x')
plt.title('K-Means cluster for all Subjects - Most Common Choice Picked')
plt.xlabel('Total')
plt.ylabel('Times Deck 2 Picked')
plt.show()
```



Put total values into bins for examination approx 10 bins

Conclusions

After running my **Principal component analysis** it made the clustering phase a lot easier.

It allowed me to interpret results easier

From analysing all the cluster graphs above it is clear to see that there are some findings to take away.

When taking into account, totals with the chosen decks, it is expected that decks 3 and 4 would be selected the most for the participants who made the most monet and this is evident in the cluster maps above.

The deck selection and the totals all seem to follow the same trend, with the exception of a few outliers.

In the data exploration section we saw that deck 2 was the deck that was picked the most out of all the decks. This was very surprising as it was considered to be one of the supposed “bad” decks. From comparing the Total graphs to the times deck 2 was selected it is clear to see that participants who selected this deck the most were clearly punished resulting in total losses overall

Conclusions

As part of this assignment the task was to perform K-means clustering, an unsupervised machine learning method on the data provided as part of the Iowa gambling task. The research as part of this assignment allowed to find various features about the data during the data exploration phase. As part of the clustering phase of this assignment

Data Analysis conclusions

The first conclusion that stuck out was the fact that deck 2 was the most selected deck out of all the decks. Deck 2 was considered to be a “bad” deck and reward less compared to decks 3 and 4. It was for this reason I decided to do my cluster analysis on deck selections and totals. The findings of the cluster analysis provided more in-depth answers for this.

K-means Clustering conclusions

From looking at the cluster analysis in the clustering section of this paper, the main take away is that all studies seem to have similar trends. For example when you look at the cluster plots, the studies that make the most money tend to have the same selection in the cards chosen. For example, the plot for deck 1 shows that a lot of the participants who picked this deck a lot also lost money.

The cluster anaylsis provided does show that participants who selected the good decks, decks 3 and 4, were far more profitable in comparison to the partcipants who selected the “bad” decks the most often (decks 1 and 2). This was not obvious in the initail data exploration section but it became obvious from running the k-means algorithm on the data.

Future work

The data that is used as part of this experiment can be very beneficial. It could be used as training data for “helathy” participants for similar projects in the future. It would be interesting to see how a predictive model works for a psychological experiment like this.

By Michael Mc Cahill