**Analysing Cards**

Table of Contents

|  |  |  |
| --- | --- | --- |
| S. no. | Title | Page no. |
| 1 | Introduction | 2 |
| 2 | Background | 3 |
| 3 | Technical Details | 4 |
| 4 | Analysis Findings | 6 |
| 5 | Conclusion | 9 |

1. Introduction

The software performs data analysis on a pre-defined deck of cards. To simplify analysis, the values for Ace, Jack, Queen and King are assigned to 1,11,12,13 respectively and the suits Hearts, Club, Diamond, Spade are assigned values 0,1,2 ,3 respectively. If the number of suits needs to be increased, the new suits are assigned values starting from 4 to the number of suits added.

Fairness of the deck is proven by drawing a card from a shuffled deck and placing the card back into the deck. The process is repeated for user-defined number of trials – which by default is allocated to 1000 times.

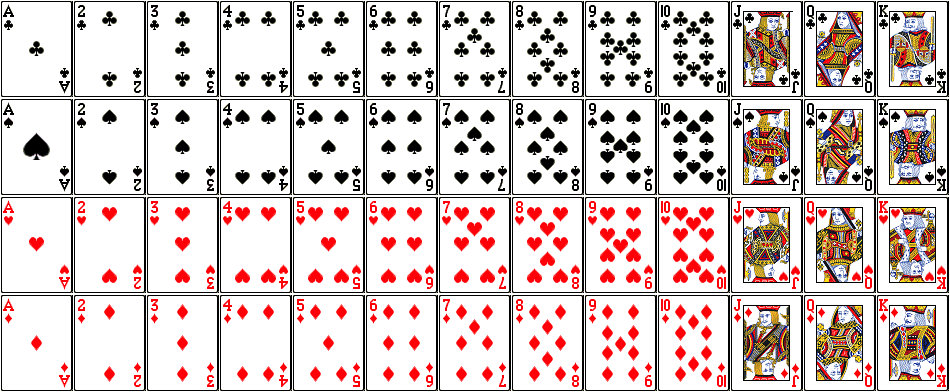
The software analyses the chance of landing the poker hand of Royal Flush for a deck of 52 cards containing 4 suits and 1 to 13 values. A hand in poker is a set of five cards selected from the top of the given deck. For a hand to be a Royal Flush hand, the hand must consist of values 10, Jack, Queen, King and Ace (10,11,12,13,1) and additionally, all the cards in the hands must be of the suit of Hearts (0). The probability of landing a Royal Flush is calculated user-defined number of times – by default allocated to 20 times. The average of all the captured probabilities is calculated to define probability of landing a Royal Flush hand.

The software is extended to perform the analysis of landing the Royal Flush hand for an increasing number of suits. The number of suits are increased from 1 suit to user-defined number of suits – by default allocated to 10.

The software also makes use of matplotlib and pandas libraries to provide comprehensive data analysis on the performed operations.

1. Background

The software makes use of Deck and Card classes to perform data analysis. Build-in methods of Deck class – drawCard() and placeCardTop() are used for drawing a card from given deck and returning the card to the given deck respectively. Built-in methods of Card class – getValue() and getSuit() are used to retrieve the value and suit of the card.

A deck of 52 cards can be depicted as follows:

Source: http://www.milefoot.com/math/discrete/counting/images/cards.png

A hand of royal flush is depicted below:



Source: https://upload.wikimedia.org/wikipedia/commons/d/d8/Royal\_Flush\_w.jpg

1. Technical Details
   1. Tasks.py
      1. **Variables used:**

no\_of\_attempts 🡪 number of trials used

list\_mean\_values 🡪list used to contain the mean face value of each trial

test\_deck 🡪deck object used calculate mean face value

face\_value\_sum 🡪 stores the summation value for all attempts in each trial

count\_attempts 🡪 number of attempts in each trial

test\_card 🡪stores the card object drawn from the deck object

test\_face 🡪 stores the face value of each card drawn

mean\_value 🡪 stores the mean face value for each trial

nbr\_of\_trials 🡪 stores the number of trials for iteration

condition 🡪 specifies whether the success condition is specified or not

cards 🡪 stores a hand/a set of 5 cards to check for success condition

test\_success 🡪 checks if a card satisfies the test success condition

list\_count\_attempts 🡪 list of attempts required to satisfy the success condition

counters 🡪 stores the list of lists of attempts required to perform the experiment

suit\_size 🡪 stores the size of suit in deck object for iteration purpose.

counter\_suit 🡪 stores the list of attempts required for success condition in each suit size.

* + 1. **Methods implemented:**

fairness(count = 1000, attempts = 20) 🡪 proves the fairness of the deck

chance\_of\_hand(deck = Deck(1,13,4), trials = 20) 🡪 calculates and returns the number of attempts required to draw a hand of Royal Flush

change\_in\_chance(size=10, trials=5) 🡪 calculates and returns the number of attempts required to draw a hand of Royal Flush for an increasing of suit size.

Main() 🡪 provides user-interface, error-handling and testing features to the software

* 1. Data\_Analysis.py
     1. **Variables used:**

df 🡪 specifies the dataframe for obtaining tabulated data and perform analysis.

x 🡪 specifies the x coordinates for graphical purposes

y 🡪 specifies the y coordinates for graphical purposes

face\_nos 🡪 specifies the number of face values for a deck of cards (1 to 13)

bins 🡪 specifies the range values for a histogram chart

sum 🡪 stores the summation value of all the calculated number of attempts.

mean\_list 🡪 stores the list of all calculated mean values

mean\_calc 🡪 stores the mean value calculated

* + 1. **Methods implemented:**

fairness\_df(list\_of\_mean\_values) 🡪 creates a data frame for all the mean face values calculated

fairness\_plot\_bar(list\_mean\_values) 🡪 plots a bar chart for all the mean face values calculated

fairness\_plot\_hist(list\_mean\_values) 🡪 plots a histogram chart for all the mean face values calculated

chance\_royal\_flush\_df(list\_attempts) 🡪 creates a data frame for all the attempt values calculated

royal\_flush\_plot\_bar(list\_attempts) 🡪 plots a bar chart for all the attempt values calculated

royal\_flush\_plot\_hist(list\_attempts) 🡪 plots a histogram chart for all the attempt values calculated

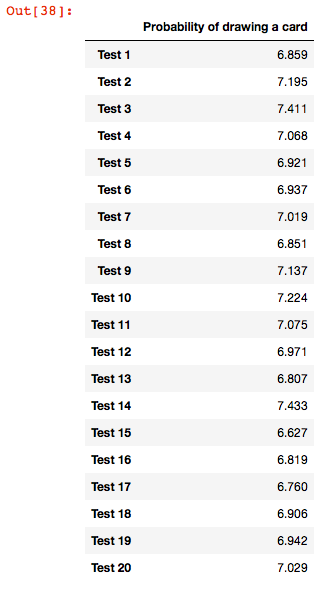
change\_in\_chance\_df(counters) 🡪 creates a data face for all the attempt values calculated for increasing suit size

change\_plot\_scatter(counters) 🡪 creates a scatter plot graph for all the mean attempt values calculated for increasing suit size

mean(counters) 🡪 calculates the mean value for all suit size for the provided data set.

Main() 🡪 provides user-interface, error-handling and testing features to the software

1. Data Analysis Findings

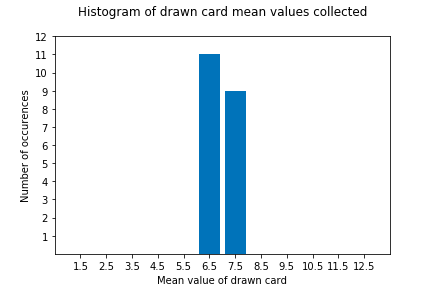
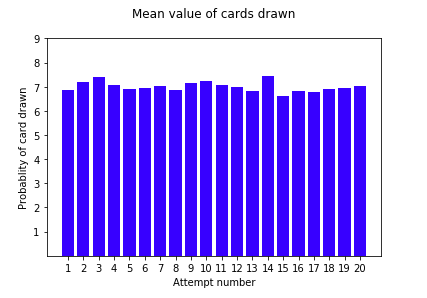


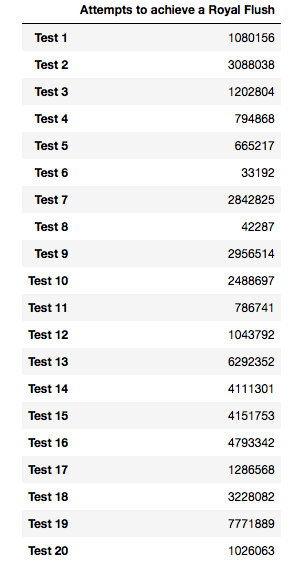
* 1. Proving Fairness:

A deck is considered a balanced deck if there is an equal chance of drawing a card. To prove that the deck used for forthcoming data analysis is indeed fair, a card is drawn from a shuffled (standard 52 card) deck user-defined number of times (by-default set to 1000 times). Each time a card drawn, the face value of card is stored and mean average of all attempts is calculated using the following formula:

The result of 20 tests consisting of 1000 attempts is tabulated and its data is analysed using bar chart and a histogram.

Each test provides a mean value in range of 6.5 to 7.5 for 1000 attempts of drawing a card from a shuffled deck.

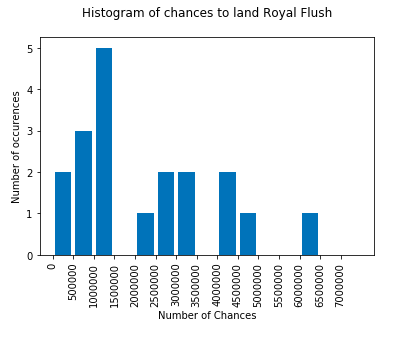
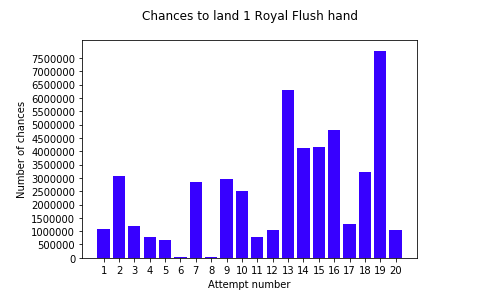


* 1. Chance of Hand – Royal Flush:

A hand of Royal Flush (1,10,11,12,13 of Hearts) is an extremely rare hand occurrence in the game of Poker. To find the probability of landing a Royal Flush, five cards are drawn from a shuffled deck of 52 cards (Face values: 1 to 13, Suits: 0 to 3) and success is calculated based on the drawn card’s face value and suit value. As the occurrence of Royal Flush is rare and often fluctuating, there is no user-defined number of attempts accepted and instead the attempts required to land one Royal Flush hand is calculated. Probability for each trial is calculated using the following formula:

The result of each test to achieve one success of drawing Royal Flush is tabulated and its data is analysed using bar chart and a histogram. Average probability for all set of trials is calculated using the following formula:

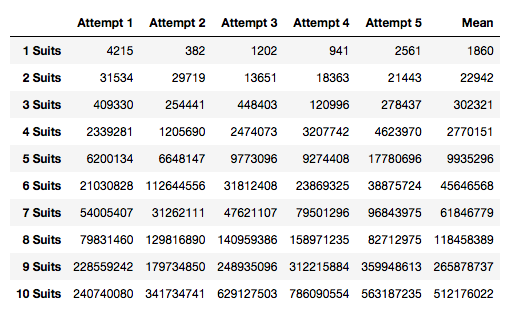
0.00000324904



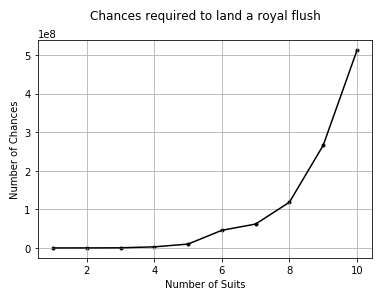
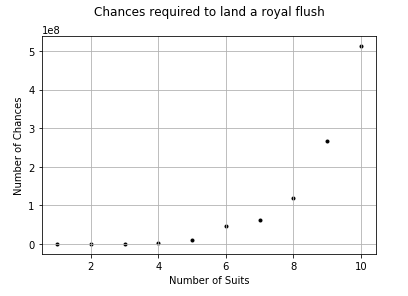
* 1. Change in Chance:

To find the probability of obtaining Royal Flush in a deck with increasing number of suits (from 1 suit to 10 suits), five cards are drawn from 10 shuffled decks each with increasing number of suits until a Royal Flush is obtained. The process is repeated user-defined number of times (by-default set to 5 times) and mean average for each suit is calculated using the following formula:

The following table consists of the five attempts for increasing number of suits from 1 suit to 10 suits:



The following scatter plots demonstrate the number of chances required for increasing number of suits:



1. Conclusion
   1. Proving Fairness:

From the analysis done in section 6.1, it can be concluded that the deck used for performing data analysis is indeed a fair deck. The mean value of the card drawn for 1000 attempts is a minimum of 6.627 and a maximum of 7.433 obtained from 20 independent attempts.

* 1. Chance of Hand – Royal Flush:

The calculated average of 0.00000324904 suggests that as expected, the probability of landing a Royal Flush hand is very rare. For 100000 attempts, a Royal Flush has a probability of occurring 3.2 number of times – for a standard deck of 52 cards. Out of the 20 independent attempts, the time required to obtain a Royal Flush was minimum in Test 6 – obtaining the hand after 33192 tries. The maximum time to achieve a Royal Flush hand was in Test 19 – Obtaining the hand after 7771889 tries.

* 1. Change in Chance:

As the number of suits increase, the number of chances/attempts required to land a hand of Royal Flush increases exponentially. As depicted in the scatter plot above, the number of attempts required for obtaining Royal Flush hand is least for deck with 1 suit and the highest for deck with 10 suits. 1e8 is a standard scientific notation which indicates that the y-coordinates for each suit be multiplied by 10^8.

Appendix A Sample Data:

1.1a

df = pd.DataFrame([6.859,7.195,7.411,7.068,6.921,6.937,7.019,6.851,7.137,7.224,7.075,6.971,6.807,7.433,6.627,6.819,6.76,6.906,6.942,7.029], index=['Test 1','Test 2','Test 3','Test 4','Test 5','Test 6','Test 7','Test 8','Test 9','Test 10','Test 11','Test 12','Test 13','Test 14','Test 15','Test 16','Test 17','Test 18','Test 19','Test 20'], columns=['Probability of drawing a card'])

df

1.1.b

import matplotlib.pyplot as plt

import numpy as np

x=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

y=[6.859,7.195,7.411,7.068,6.921,6.937,7.019,6.851,7.137,7.224,7.075,6.971,6.807,7.433,6.627,6.819,6.76,6.906,6.942,7.029]

plt.bar(x,y, color='b')

plt.xlabel('Attempt number')

plt.ylabel('Probablity of card drawn')

plt.xticks(np.arange(min(x), max(x)+1, 1.0))

plt.yticks(np.arange(1, max(y)+2, 1.0))

plt.title('Mean value of cards drawn\n')

plt.show()

1.1c

import matplotlib.pyplot as plt

y=[6.859,7.195,7.411,7.068,6.921,6.937,7.019,6.851,7.137,7.224,7.075,6.971,6.807,7.433,6.627,6.819,6.76,6.906,6.942,7.029]

attempt\_nos = [x+1 for x in range(13)]

plt.hist(y, attempt\_nos, histtype='bar', rwidth = 0.8)

plt.xlabel('Mean value of drawn card')

plt.ylabel('Number of occurences')

plt.xticks(np.arange(min(attempt\_nos)+0.5, max(attempt\_nos), 1))

plt.yticks(np.arange(1, max(y)+5, 1.0))

plt.title('Histogram of drawn card mean values collected\n')

plt.show()

1.2a

import pandas as pd

df = pd.DataFrame([1080156,3088038,1202804,794868,665217,33192,2842825,42287,2956514,2488697,786741,1043792,6292352,4111301,4151753,4793342,1286568,3228082,7771889,1026063], index=['Test 1','Test 2','Test 3','Test 4','Test 5','Test 6','Test 7','Test 8','Test 9','Test 10','Test 11','Test 12','Test 13','Test 14','Test 15','Test 16','Test 17','Test 18','Test 19','Test 20'], columns=['Attempts to achieve a Royal Flush'])

df

1.2b

import matplotlib.pyplot as plt

import numpy as np

x=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

y=[1080156,3088038,1202804,794868,665217,33192,2842825,42287,2956514,2488697,786741,1043792,6292352,4111301,4151753,4793342,1286568,3228082,7771889,1026063]

plt.bar(x,y, color='b')

plt.xlabel('Attempt number')

plt.ylabel('Number of chances')

plt.xticks(np.arange(min(x), max(x)+1, 1.0))

plt.yticks(np.arange(0, max(y), 500000))

plt.title('Chances to land 1 Royal Flush hand\n')

plt.show()

1.2c

import matplotlib.pyplot as plt

y=[1080156,3088038,1202804,794868,665217,33192,2842825,42287,2956514,2488697,786741,1043792,6292352,4111301,4151753,4793342,1286568,3228082,7771889,1026063]

bins=[0, 500000, 1000000, 1500000, 2000000, 2500000, 3000000, 3500000, 4000000, 4500000, 5000000, 5500000, 6000000, 6500000, 7000000, 7500000]

plt.hist(y, bins, histtype='bar', rwidth = 0.8)

plt.xlabel('Number of Chances')

plt.ylabel('Number of occurences')

plt.xticks(np.arange(min(bins), max(bins), 500000), rotation=90)

# plt.yticks(np.arange(1, max(y)+5, 500000))

# plt.xticks(rotation=90)

plt.title('Histogram of chances to land Royal Flush\n')

plt.show()

1.3a

import pandas as pd

df = pd.DataFrame([[4215, 382, 1202, 941, 2561], [31534, 29719, 13651, 18363, 21443], [409330, 254441, 448403, 120996, 278437], [2339281, 1205690, 2474073, 3207742, 4623970], [6200134, 6648147, 9773096, 9274408, 17780696], [21030828, 112644556, 31812408, 23869325, 38875724], [54005407, 31262111, 47621107, 79501296, 96843975], [79831460, 129816890, 140959386, 158971235, 82712975], [228559242, 179734850, 248935096, 312215884, 359948613], [240740080, 341734741, 629127503, 786090554, 563187235]], index=['1 Suits','2 Suits','3 Suits','4 Suits','5 Suits','6 Suits','7 Suits','8 Suits','9 Suits','10 Suits'], columns=['Attempt 1', 'Attempt 2','Attempt 3', 'Attempt 4', 'Attempt 5'])

df['Mean'] = [1860,22942,302321,2770151,9935296,45646568,61846779,118458389,265878737,512176022]

df

1.3b

import matplotlib.pyplot as plt

x=[1,2,3,4,5,6,7,8,9,10]

y=[1860,22942,302321,2770151,9935296,45646568,61846779,118458389,265878737,512176022]

plt.scatter(x,y,color='k',marker='.')

plt.plot(x,y,color='k')

plt.grid(True)

plt.xlabel('Number of Suits')

plt.ylabel('Number of Ate')

plt.title('Chances required to land a royal flush\n')

plt.show()s