CS Internal Assesment: Turkish Poker

Introduction

This is my CS Internal Assesment project. The format of the project requires that I have a client who asks for a certain deliverable which I provide by using the things I learned in my CS class. My client wanted two things: two tables showing the chances of winning for different hands in Turkish poker and simple program that will tell him how much to bet given a certain hand, a pot size, and the stage of the game. To complete the assignment, I did four things:

1. Wrote a Python program that can simulate Turkish Poker games and record the results in a file.
2. Analyzed the results of the simualted hands using the Pandas data analysis library.
3. Used the output of the analysis to prepare a nice looking table for the client.
4. Used the output of the analysis to write a simple program that suggests how mcuh to bet.

Turkish Poker Explained

Turkish poker is a 5 Card Draw poker variant played with a short deck of 32 cards. Each player receives five cards and up to four can be exchanged.

Players and Cards

Although between 2 and 5 players can play the game, for the purposes of this brief, we will only look at the 4 player variety. In the four player variety, there are thirty two cards, A-K-Q-J-10-9-8-7 of each suit. The Ace is normally the highest card, but can be used as a low card to make a straight (or straight flush). When used as a low card the Ace ranks immediately below the lowest card in the pack - for example with four players the Ace ranks immediately below the 7 and 10-9-8-7-A is a straight. The rank of the suits, from high to low, is: Hearts, Diamonds , Spades, and Clubs.

Names and Ranks of Hands

Note that there are four important differences from international poker:

1. A flush beats a full house (flushes are more difficult to make with a shortened deck).
2. Three of a kind beats a straight (straights are easier make with a shortened deck).
3. Aces are considered consecutive with the lowest card of the deck when making a low (minor) straight.
4. In other cases, suits are used to break ties between otherwise equal hands.

Here is a list of the hand types in order from highest to lowest and notes on how to compare them when two or more players have the same type of hand.

1. Straight Flush: five consecutive cards of a suit. For example ♠K – ♠Q – ♠J – ♠10 – ♠9. An Ace can be used instead of 6 making a Ace Low Straight Flush such as ♠10 – ♠9-♠8 – ♠7-♠A. When comparing straight flushes, first compare the rank or the highest card. In Ace Low Straight Flushes, the highest card is the 10, not the Ace. If the ranks are the same, the suit of the highest card determines the winner.
2. Four of a Kind: four equal ranked cards, with any fifth card, for example ♥K - ♠K - ♦K - ♣K - ♠9. When comparing these, the higher set of four is better.
3. Flush: any five cards of the same suit, such as ♣A – ♣Q – ♣10 – ♣9 – ♣7. When comparing two flushes, compare the rank of the highest card, then the second highest and so on down; if all five cards are equal the suit decides. For example ♣A – ♣Q – ♣J – ♣8 – ♣7 is better than ♥A - ♥Q - ♥10 - ♥9 - ♥8 because the Jack is higher than the 10.
4. Full House: three cards of one rank plus two cards of another rank, such as ♠8 – ♦8 – ♣8 – ♥J – ♣J. When comparing these, the higher three of a kind is better.
5. Three of a kind: three cards of the same rank plus two unequal cards, such as ♥9 – ♦9 – ♣9 – ♠J – ♦Q. When comparing these, the higher three of a kind is better.
6. Straight: five consecutive cards of mixed suits, such as ♣A - ♦K - ♠Q - ♠J - ♥10 or ♦Q – ♦J – ♠10 - ♣9 - ♥8. Ace can be used instead of a 6 making an Ace Low Straight, in which case the 10 is considered the highest card of the straighht. The straight with the higher top card is better. If the high cards are equal, the suit of the 10 decides the winner (every straight has a 10).
7. Two Pairs: two cards of one rank, two cards of another rank, and one other card - for example ♣A – ♠A – ♥J – ♦J - ♣8. First compare the higher pair, then the lower pair, then the odd card. If the rank of all three groups are equal, the suit of the odd card determines the winner.
8. Pair: two cards of the same rank and three odd cards. For example ♥Q – ♦Q – ♣K – ♠7 - ♦8. Compare the rank of the pairs, then the rank of the odd cards, beginning with the highest. Only if all five cards are equal, the suit of the highest odd card determines the winner.
9. High Card: five cards that do not form any of the above combinations - for example ♠A – ♥J – ♣10 – ♠8 - ♥7. Compare the rank of the highest card, then the second highest and so on. If all ranks are equal, the suit of the highest card decides.

Deal and First Betting Round

The first dealer is chosen at random and the turn to deal passes to the left after each hand. Before each deal all players must ante an equal amount to the pot. The cards are shuffled, cut, and dealt clockwise one at a time until each player has five cards. At this point there is a round of betting, in which the minimum requirement to open is a pair of Kings. Players who have less than this must pass, and players are allowed to pass even if they meet the opening requirement. Once a player has opened, other players can call , raise or re-raise even without a pair of kings. Players who do not wish to match the latest bet can fold. They throw in their cards face down, take no further part in the hand, and cannot win the pot. Note that if the opener folds, he or she may be required to show enough cards to prove that the opening was legal -

1. The Simulator: simulatehands.py

The simulatehands.py program simulates Turkish poker games. A game consists of a deal, draw and showdown. The program doesn't do any betting simulation but simply notes the value of each hand in the game and the winner. More specifically in each game:

- Cards are dealt to 4 four players.
- Each hand is evaluated to find its rank. These are initial hands.
- If there are no hands that can open the game then the game is discontinued.
- If there is an opening hand each player discards her own unwanted cards and draws new cards. These are final hands.
- Final hands are evaluated and the highest hand is declared the winner.
- The initial and final values of hands are recorded for analysis. If simulatehands.py is run with no arguments, it simulates a hand and prints each it in human readeable form to the screen. If it is run with the -n argument, it simulates number of hands indicated by the value that comes after n. If it is run with the -f argument, it does not print to the screen but to a file named by the string that comes after -t. To make the analysis, I ran the program with: `python simulatehands.py –n 10000000 –f 40M` This produced a 2.34 GB file named 40M.csv The source code of simulatehands.py and its ouput for a single gamme is below

In [ ]:
```
# %load simulatehands.py
#!/usr/bin/python

"""
This is a script for creating simulated Turkish poker hands.
It takes two arguments:
- the number of hands to be simulated (defaults to 1)
- the name of the output file. If none given, results are written to the
stdout in pretty format.

If a file name is given, the program produces the file
name.csv. This is a csv file containing all details for all simulated hands

simulatehands.py 2000 test
will simulate 2000 hands and produce test.csv.

simulatehands.py
will simulate one hand and write the results to stdout
"""
# TODO: User interface

import argparse
import collections
import itertools
import random
import operator

parser = argparse.ArgumentParser()
parser.add_argument('-n', '--number', nargs='?', default=1, type=int,
                    help="number of hands to be simulated")
parser.add_argument('-f', '--file', nargs='?', default="", type=str,
                    help="base file name for output files")
args = parser.parse_args()

games = int(args.number)
file_name = args.file + ".csv"

# Unicode symbols for Clubs, Spades, Diamonds, Hearts
L_SUIT_LIST = (u"\u2663", u"\u2660", u"\u2666", u"\u2665")
# Ace is also at the beginning to display the hand correctly when we use the
# Ace as a six in straights
L_NUMERAL_LIST = ("Ace", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace")
D_NUMERAL_LIST = ("A", "7", "8", "9", "T", "J", "Q", "K", "A")
D_SUIT_LIST = ("C", "S", "D", "H")
SUIT_LIST = (0, 1, 2, 3)  # Clubs, Spades, Diamonds, Hearts
NUMERAL_LIST = (1, 2, 3, 4, 5, 6, 7, 8)
OPENER_LIMIT = 7  # Minimum value to open a game, corresponds to a pair of K's
CSV_HEADERS = "Game,Initial_Hand_1,Initial_Hand_2,Initial_Hand_3," \
              "Initial_Value,Final_Hand_1,Final_Hand_2,Final_Hand_3," \
              "Final_Value,Winner\n"


class Card:
    def __init__(self, numeral, suit):
        self.numeral = numeral
        self.suit = suit
        self.sort_order = 0
        self.card = self.numeral, self.suit, self.sort_order

    def __repr__(self):
        return D_NUMERAL_LIST[self.numeral] + L_SUIT_LIST[self.suit]


class Deck(set):
    def __init__(self):
        self.cards = set()
        self.discards = set()
        for numeral, suit in itertools.product(NUMERAL_LIST, SUIT_LIST):
            self.cards.add(Card(numeral, suit))
```

In [2]:
```
#!/usr/bin/python

"""
This is a script for creating simulated Turkish poker hands.
It takes two arguments:
- the number of hands to be simulated (defaults to 1)
- the name of the output file. If none given, results are written to the
stdout in pretty format.

If a file name is given, the program produces the file
name.csv. This is a csv file containing all details for all simulated hands

simulatehands.py 2000 test
will simulate 2000 hands and produce test.csv.

simulatehands.py
will simulate one hand and write the results to stdout
"""

import argparse
import collections
import itertools
import random
import operator

parser = argparse.ArgumentParser()
parser.add_argument('-n', '--number', nargs='?', default=1, type=int,
                    help="number of hands to be simulated")
parser.add_argument('-f', '--file', nargs='?', default="", type=str,
                    help="base file name for output files")
args = parser.parse_args()

games = int(args.number)
file_name = args.file + ".csv"

# Unicode symbols for Clubs, Spades, Diamonds, Hearts
L_SUIT_LIST = (u"\u2663", u"\u2660", u"\u2666", u"\u2665")
# Ace is also at the beginning to display the hand correctly when we use the
# Ace as a six in straights
L_NUMERAL_LIST = ("Ace", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace")
D_NUMERAL_LIST = ("A", "7", "8", "9", "T", "J", "Q", "K", "A")
D_SUIT_LIST = ("C", "S", "D", "H")
SUIT_LIST = (0, 1, 2, 3)  # Clubs, Spades, Diamonds, Hearts
NUMERAL_LIST = (1, 2, 3, 4, 5, 6, 7, 8)
OPENER_LIMIT = 7  # Minimum value to open a game, corresponds to a pair of K's
CSV_HEADERS = "Game,Initial_Hand_1,Initial_Hand_2,Initial_Hand_3," \
              "Initial_Value,Final_Hand_1,Final_Hand_2,Final_Hand_3," \
              "Final_Value,Winner\n"


class Card:
    def __init__(self, numeral, suit):
        self.numeral = numeral
        self.suit = suit
        self.sort_order = 0
        self.card = self.numeral, self.suit, self.sort_order

    def __repr__(self):
        return D_NUMERAL_LIST[self.numeral] + L_SUIT_LIST[self.suit]


class Deck(set):
    def __init__(self):
        self.cards = set()
        self.discards = set()
        for numeral, suit in itertools.product(NUMERAL_LIST, SUIT_LIST):
            self.cards.add(Card(numeral, suit))

    def get_cards(self, n=1):
```

In [3]:
```
%run simulatehands.py
```

```
--------------------------------------------------------------------------------
--
Game 1
K♦J♥T♦9♠8♠ High Card (Full open ended straight) (0.754322) D: K♦ -> J♥T♦9♠8♠7♣
Straight Jack high (90.543212) Winner!
A♥A♣K♠T♠7♠ A pair of Aces (8.741100) D: K♠T♠7♠ -> A♥A♣Q♠8♣7♦ A pair of Aces (8
.621100)
J♦J♠A♠Q♦8♦ A pair of Jacks (5.862100) D: A♠Q♦8♦ -> J♦J♠K♥9♥8♥ A pair of Jacks
(5.732300)
Q♥Q♣T♣T♥J♣ Two Pairs: Queens over 10s (64.500000) D: J♣ -> Q♥Q♣T♣T♥7♥ Two Pair
s: Queens over 10s (64.130000)
```

2. The Analysis

The file 40M.csv has the following fields for each hand (each game has four hands):

Game: The number of the simualated game.

Initial_Hand_1: Rank of the initial hand.

Initial_Hand_2: The second level of evaluation for the initial hand.

Initial_Hand_3: The third level of evaluation for the initial hand.

Initial_Value: A numeric value for the initial hand.

Final_Hand_1: Rank of the final hand.

Final_Hand_2: The second level of evaluation for the final hand.

Final_Hand_3: The third level of evaluation for the final hand.

Final_Value: A numeric value for the final hand.

Winner: 1 is that hand won the game, 0 if it has not.

We analyze the data with pandas.

In [4]:
```
import pandas as pd
```

The data is in string format, to get nicely sorted hands, we will turn these into categorical data.

Level 1 evaluations are just cards names. We do not distinguish between high card hands.

Level 2 data is different between the initial and final hands. In initial hands, if we have a hand other than a high card, we use the secondary level to indicate the strength of the following cards, if we only have a high card, we use the second level to indicate the potential of the high card (which indicated in the simulated which cards we kept and which cards we discarded)

Level 3 data is the same for both final and initial hands, it indicated the stregth of the thrid part of the hand. Such as the suite of a flsuh or the numeral value of the smaller pair of a two pairs.

In [5]:
```
level1 = ['High Card',
       'Pair',
       'Two Pairs',
       'Straight',
       'Trips',
       'Full House',
       'Flush',
       'Four of a kind',
       'Straight Flush']
initlevel2 =['(Almost flush)',
         '(Full open ended straight)',
         '(Keep the highest)',
         '(Long shot flush)',
         '(Gut shot straight)',
         '(Full open ended long straight)',
         '(Hail Mary flush)',
         '(Half gut shot long straight)',
         '(Full gut shot long straight)',
         '(Keep the highest non A or K)',
         '(Two high cards)',
         'A', 'K', 'Q', 'J', 'T', '9', '8', '7']
finallevel2 = ['A', 'K', 'Q', 'J', 'T', '9', '8', '7']
level3 = ['A', 'K', 'Q', 'J', 'T', '9', '8', '7', 'H', 'D', 'S', 'C']
```

We read the game data of 40 Million hands (of 10 Million games).

In [6]:
```
hands = pd.read_csv("40M.csv")
```

We change the string values to categorical values to save memory space and to sort the data better.

In [7]:
```
hands["Initial_Hand_1"] = pd.Categorical(hands["Initial_Hand_1"], level1)
hands["Final_Hand_1"] = pd.Categorical(hands["Final_Hand_1"], level1)
hands["Initial_Hand_2"] = pd.Categorical(hands["Initial_Hand_2"], initlevel2)
hands["Final_Hand_2"] = pd.Categorical(hands["Final_Hand_2"], finallevel2)
hands["Initial_Hand_3"] = pd.Categorical(hands["Initial_Hand_3"], level3)
hands["Final_Hand_3"] = pd.Categorical(hands["Final_Hand_3"], level3)
```

This is what the data looks like.

In [8]:
```
hands.head(8)
```

Out[8]:

	Game	Initial_Hand_1	Initial_Hand_2	Initial_Hand_3	Initial_Value	Final_Hand_1	Final_Hand_2
0	1	Pair	J	9	5.8631	Trips	J
1	1	Two Pairs	Q	9	63.4200	Two Pairs	Q
2	1	Two Pairs	8	7	21.8300	Full House	8
3	1	Pair	T	9	4.8710	Pair	T
4	2	Pair	J	9	5.8433	Trips	J
5	2	Two Pairs	9	8	32.1200	Two Pairs	9
6	2	Pair	T	9	4.7523	Pair	T
7	2	Two Pairs	Q	7	61.8100	Two Pairs	Q

Let us first look at the winning chances of different initial hands considering only the first level of evaluation. We do that by grouping the data by Initial_Hand_1 and adding two kinds of values: the count of winners (which is equal to the hands played) and the sum of winners (which is equal to the number of times that hand won). We then add a new column `perc` by dividing the number of wins by the number of games. Since we do not need the number of games and wins, we discard them by using `df.drop()`.

In [9]:
```
init1 = pd.DataFrame(hands.groupby(["Initial_Hand_1"]).agg({"Winner": [sum, "co
unt"]}))
init1.columns = init1.columns.droplevel()
init1["perc"] = init1["sum"]/init1["count"]
init1.drop(labels=['sum', 'count'], axis=1, inplace=True)
init1
```

Out[9]:

	perc
Initial_Hand_1	
High Card	0.092293
Pair	0.214392
Two Pairs	0.331776
Straight	0.486066
Trips	0.671508
Full House	0.859842
Flush	0.961437
Four of a kind	0.978157
Straight Flush	0.999382

The init2 dataframe is the first of the tables that is necessary for the assignment. We calculate the final winning chance of each initial hand at level 2. The operations are identical to what we have done for level one, the only difference is that we group by both by Initial_Hand_1 and by Initial_Hand_2. We do not take level 3 into consideration because it adds very little value while increasing the size of the table significantly.

In [11]:
```
init2 = pd.DataFrame(hands.groupby(["Initial_Hand_1", "Initial_Hand_2"]).agg({"
Winner": [sum, "count"]}))
init2.columns = init2.columns.droplevel()
init2["perc"] = init2["sum"]/init2["count"]
init2.drop(labels=['sum', 'count'], axis=1, inplace=True)
init2
```

Out[11]:

		perc
Initial_Hand_1	**Initial_Hand_2**	
High Card	**(Almost flush)**	0.159149
	(Full open ended straight)	0.129922
	(Keep the highest)	0.100633
	(Long shot flush)	0.073221
	(Gut shot straight)	0.070580
	(Full open ended long straight)	0.069428
	(Hail Mary flush)	0.084042
	(Half gut shot long straight)	0.064773
	(Full gut shot long straight)	0.067729
	(Keep the highest non A or K)	0.087710
	(Two high cards)	0.089987
Pair	**A**	0.346424
	K	0.287446
	Q	0.197637
	J	0.181470
	T	0.169362
	9	0.161286
	8	0.153031
	7	0.145947
Two Pairs	**A**	0.455678
	K	0.358807
	Q	0.302494
	J	0.263786
	T	0.241006
	9	0.222451
	8	0.210962
Straight	**A**	0.529140
	K	0.508345
	Q	0.488637
	J	0.464027
...	**...**	...
Trips	**Q**	0.731045
	J	0.682919
	T	0.641544
	9	0.602398
	8	0.567948
	7	0.533806
Full House	**A**	0.943999

This table provides two very valuable insights:

1. You have a better chance of winning a hand if you draw to a pair of 7s instead of drawing to a open ended straight. Most players would do otherwise.
2. You have a better chance of winning just by keeping your high card and drawing four cards compared to most long shot straight and flush draws--except an Almost Flush (a hand with four cards of the same suit) and an Open Ended Straight (a hand that has four consecutive cards and that can be turned into a straight by getting a consecutive card to either end of the four consecutive cards. Most players would prefer to draw one card to go for close ended or gut shot straights or two cards to make three cards of the same suit into a flush.

We may use this data in the future in a different python program so we turn it into a nice dictionary keyed by tuples and with values corresponding to chances of winning.

In [20]:
```
temp_init2_dict = init2.to_dict('split')
init2_data = temp_init2_dict['data']
init2_index = temp_init2_dict['index']
init2dict = dict(zip(init2_index, init2_data))
init2dict
```

Out[20]:
```
{('Flush', 'A'): [0.96810849304889],
 ('Flush', 'K'): [0.9494690589527646],
 ('Flush', 'Q'): [0.9479544277576385],
 ('Four of a kind', '7'): [0.9603776283793448],
 ('Four of a kind', '8'): [0.9636761487964989],
 ('Four of a kind', '9'): [0.969778174780408],
 ('Four of a kind', 'A'): [0.9988075719183187],
 ('Four of a kind', 'J'): [0.9805738194859533],
 ('Four of a kind', 'K'): [0.994155464640561],
 ('Four of a kind', 'Q'): [0.9856430406528638],
 ('Four of a kind', 'T'): [0.9731563421828908],
 ('Full House', '7'): [0.7830505959740838],
 ('Full House', '8'): [0.7986367439342488],
 ('Full House', '9'): [0.8245424395037465],
 ('Full House', 'A'): [0.9439994138621599],
 ('Full House', 'J'): [0.8705113720096281],
 ('Full House', 'K'): [0.9164529397532059],
 ('Full House', 'Q'): [0.8937535477183544],
 ('Full House', 'T'): [0.8460277458344547],
 ('High Card', '(Almost flush)'): [0.15914949932299596],
 ('High Card', '(Full gut shot long straight)'): [0.06772883038882094],
 ('High Card', '(Full open ended long straight)'): [0.06942786298098999],
 ('High Card', '(Full open ended straight)'): [0.12992202373571227],
 ('High Card', '(Gut shot straight)'): [0.07057956325439971],
 ('High Card', '(Hail Mary flush)'): [0.08404163018856073],
 ('High Card', '(Half gut shot long straight)'): [0.06477291079727959],
 ('High Card', '(Keep the highest non A or K)'): [0.08771020502188435],
 ('High Card', '(Keep the highest)'): [0.10063349929647479],
 ('High Card', '(Long shot flush)'): [0.07322100267209083],
 ('High Card', '(Two high cards)'): [0.08998725774674052],
 ('Pair', '7'): [0.14594735803319678],
 ('Pair', '8'): [0.15303089282756335],
 ('Pair', '9'): [0.16128568936611865],
 ('Pair', 'A'): [0.34642431277726426],
 ('Pair', 'J'): [0.18147002682001595],
 ('Pair', 'K'): [0.2874463549581572],
 ('Pair', 'Q'): [0.19763692097130903],
 ('Pair', 'T'): [0.1693619950110472],
 ('Straight', 'A'): [0.5291399926304751],
 ('Straight', 'J'): [0.46402737839698677],
 ('Straight', 'K'): [0.5083448504392665],
 ('Straight', 'Q'): [0.4886366397498927],
 ('Straight', 'T'): [0.44028860005568626],
 ('Straight Flush', 'A'): [1.0],
 ('Straight Flush', 'J'): [1.0],
 ('Straight Flush', 'K'): [1.0],
 ('Straight Flush', 'Q'): [0.9989764585465711],
 ('Straight Flush', 'T'): [0.9979591836734694],
 ('Trips', '7'): [0.5338057819882011],
 ('Trips', '8'): [0.5679477032756008],
 ('Trips', '9'): [0.6023982135756019],
 ('Trips', 'A'): [0.8336218857823455],
 ('Trips', 'J'): [0.682918593014189],
 ('Trips', 'K'): [0.7793895214164527],
 ('Trips', 'Q'): [0.7310447440676147],
 ('Trips', 'T'): [0.6415444229172433],
 ('Two Pairs', '8'): [0.21096224116930573],
 ('Two Pairs', '9'): [0.22245052889715902],
 ('Two Pairs', 'A'): [0.45567752435892545],
 ('Two Pairs', 'J'): [0.2637864957643146],
 ('Two Pairs', 'K'): [0.3588068030594223],
 ('Two Pairs', 'Q'): [0.30249424189116053],
 ('Two Pairs', 'T'): [0.24100557983492857]}
```

We do the the same thing we did for initial hands but now for final hands to see what is the winning percentage of a final hand. Note that since we recorded games where somebody opened the game (implying that they had a minimum of two Kings) high card hands cannot win the game.

In [26]:
```
final1 = pd.DataFrame(hands.groupby(["Final_Hand_1"]).agg({"Winner": [sum, "cou
nt"]}))
final1.columns = final1.columns.droplevel()
final1["perc"] = final1["sum"]/final1["count"]
final1.drop(labels=['sum', 'count'], axis=1, inplace=True)
final1
```

Out[26]:

	perc
Final_Hand_1	
High Card	NaN
Pair	0.026366
Two Pairs	0.223479
Straight	0.473760
Trips	0.608133
Full House	0.859365
Flush	0.961627
Four of a kind	0.978156
Straight Flush	0.999252

The final2 dataframe is the second of the tables that is necessary for the assignment. We calculate winning chance of each final hand at level 2.

In [15]:
```
final2 = pd.DataFrame(hands.groupby(["Final_Hand_1", "Final_Hand_2"]).agg({"Win
ner": [sum, "count"]}))
final2.columns = final2.columns.droplevel()
final2["perc"] = final2["sum"]/final2["count"]
final2.drop(labels=['sum', 'count'], axis=1, inplace=True)
final2
```

Out[15]:

		perc
Final_Hand_1	**Final_Hand_2**	
Pair	**A**	0.109663
	K	0.065742
	Q	0.000000
	J	0.000000
	T	0.000000
	9	0.000000
	8	0.000000
	7	0.000000
Two Pairs	**A**	0.366150
	K	0.250758
	Q	0.176835
	J	0.138478
	T	0.116046
	9	0.099331
	8	0.090169
Straight	**A**	0.517879
	K	0.494309
	Q	0.473327
	J	0.449032
	T	0.428342
Trips	**A**	0.807344
	K	0.744751
	Q	0.652523
	J	0.598518
	T	0.551485
	9	0.512627
	8	0.476820
	7	0.442720
Full House	**A**	0.944715
	K	0.919402
	Q	0.891276
	J	0.867084
	T	0.841388
	9	0.820942
	8	0.799602
	7	0.779584
Flush	**A**	0.965961
	K	0.954002

We turn this data into a dictionary as well for later use.

In [21]:
```
temp_final2_dict = final2.to_dict('split')
final2_data = d['data']
final2_index = d['index']
final2dict = dict(zip(final2_index, final2_data))
```

We use the pickle format to save the two dictionaries we created for the initial and final hands' chances of winnig at level 2. If we want to use them later we can read them using:

```
with open('final2.pickle.pickle', 'rb') as handle:
    final2dict = pickle.load(handle)
```

In [22]:
```
import pickle
with open('final2.pickle', 'wb') as handle:
    pickle.dump(final2dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('init2.pickle', 'wb') as handle:
    pickle.dump(init2dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

We also save the related dataframes as CSV files to keep them accessible.

In [23]:
```
init2.to_csv("Win_Odds_Init_Hand_2.csv")
final2.to_csv("Win_Odds_Final_Hand_2.csv")
```

3. The Tables

We finally write all these table to an MS Excel file to make them more presentable for the final output.

In [27]:
```
all_tables = [init1, init2, final1, final2]
all_sheet_names = ["Win_Odds_Init_Hand_1",
                   "Win_Odds_Init_Hand_2",
                   "Win_Odds_Final_Hand_1",
                   "Win_Odds_Final_Hand_2"]
```

In [28]:
```
writer = pd.ExcelWriter('PokerOdds.xlsx')
for i, oddstable in enumerate (all_tables):
    oddstable.to_excel(writer, all_sheet_names[i])
writer.save()
```