BIS 420 PROGRAMMING FOR DATA SCIENCE

PRAJAKTA POHARE

CHAPTER 18 EXERCISE 18.6

ILLINOIS STATE UNIVERSITY

The following are the possible hands in poker, in increasing order of value (and decreasing order of probability):

pair: two cards with the same rank

two pair: two pairs of cards with the same rank

three of a kind: three cards with the same rank

straight: five cards with ranks in sequence (aces can be high or low, so Ace-2-3-4-5 is a straight and so is 10-Jack-Queen-King-Ace, but Queen-King-Ace-2-3 is not.)

flush: five cards with the same suit

full house: three cards with one rank, two cards with another

four of a kind: four cards with the same rank

straight flush: five cards in sequence (as defined above) and with the same suit

The goal of these exercises is to estimate the probability of drawing these various hands.

1. Download the following files from http: // thinkpython. com/ code :

Card.py : A complete version of the Card, Deck and Hand classes in this chapter.

PokerHand.py : An incomplete implementation of a class that represents a poker hand, and some code that tests it.

2. If you run PokerHand.py, it deals seven 7-card poker hands and checks to see if any of them contains a flush. Read this code carefully before you go on.

3. Add methods to PokerHand.py named has_pair, has_twopair, etc. that return True or False according to whether or not the hand meets the relevant criteria. Your code should work correctly for “hands” that contain any number of cards (although 5 and 7 are the most common sizes).

4. Write a method named classify that figures out the highest-value classification for a hand and sets the label attribute accordingly. For example, a 7-card hand might contain a flush and a pair; it should be labeled “flush”.178 Chapter 18. Inheritance

5. When you are convinced that your classification methods are working, the next step is to estimate the probabilities of the various hands. Write a function in PokerHand.py that shuffles a deck of cards, divides it into hands, classifies the hands, and counts the number of times various classifications appear.

6. Print a table of the classifications and their probabilities. Run your program with larger and larger numbers of hands until the output values converge to a reasonable degree of accuracy. Compare your results to the values at http: // en. wikipedia. org/ wiki/ Hand_ rankings .

```
from __future__ import print_function, division

from Card import Hand, Deck

class Hist(dict):



    def __init__(self, seq=[]):

        for x in seq:
            self.count(x)

    def count(self, x, f=1):
        self[x] = self.get(x, 0) + f
        if self[x] == 0:
            del self[x]

class PokerHand(Hand):

    all_labels = ['straightflush', 'fourkind', 'fullhouse', 'flush',
                  'straight', 'threekind', 'twopair', 'pair', 'highcard']

    def make_histograms(self):

        self.suits = Hist()
        self.ranks = Hist()

        for c in self.cards:
            self.suits.count(c.suit)
            self.ranks.count(c.rank)

        self.sets = list(self.ranks.values())
        self.sets.sort(reverse=True)

    def has_highcard(self):
        return len(self.cards)

    def check_sets(self, *t):

        for need, have in zip(t, self.sets):
```

```
        if need > have:
            return False
    return True

def has_pair(self):
    return self.check_sets(2)

def has_twopair(self):
    return self.check_sets(2, 2)

def has_threekind(self):
    return self.check_sets(3)

def has_fourkind(self):


    return self.check_sets(4)

def has_fullhouse(self):


    return self.check_sets(3, 2)

def has_flush(self):


    for val in self.suits.values():
        if val >= 5:
            return True
    return False
```

```
def has_straight(self):

    ranks = self.ranks.copy()
    ranks[14] = ranks.get(1, 0)

    return self.in_a_row(ranks, 5)

def in_a_row(self, ranks, n=5):

    count = 0
    for i in range(1, 15):
        if ranks.get(i, 0):
            count += 1
            if count == n:
                return True
        else:
            count = 0
    return False

def has_straightflush(self):

    s = set()
    for c in self.cards:
        s.add((c.rank, c.suit))
        if c.rank == 1:
            s.add((14, c.suit))
    for suit in range(4):
        count = 0
        for rank in range(1, 15):
            if (rank, suit) in s:
                count += 1
                if count == 5:
                    return True
            else:
                count = 0
    return False

def has_straightflush(self):

    d = {}
    for c in self.cards:
        d.setdefault(c.suit, PokerHand()).add_card(c)

    for hand in d.values():
        if len(hand.cards) < 5:
            continue
        hand.make_histograms()
        if hand.has_straight():
            return True
    return False

def classify(self):
        self.make_histograms()

        self.labels = []
        for label in PokerHand.all_labels:
            f = getattr(self, 'has_' + label)
            if f():
                self.labels.append(label)

class PokerDeck(Deck):

    def deal_hands(self, num_cards=5, num_hands=10):

        hands = []
        for i in range(num_hands):
            hand = PokerHand()
            self.move_cards(hand, num_cards)
            hand.classify()
            hands.append(hand)
        return hands

def main():

    lhist = Hist()

    n = 10000
    for i in range(n):
        if i % 1000 == 0:
            print(i)

        deck = PokerDeck()
        deck.shuffle()

        hands = deck.deal_hands(7, 7)
        for hand in hands:
            for label in hand.labels:
                lhist.count(label)

    # print the results
    total = 7.0 * n
    print(total, 'hands dealt:')

    for label in PokerHand.all_labels:
        freq = lhist.get(label, 0)
        if freq == 0:
            continue
        p = total / freq
        print('%s happens one time in %.2f' % (label, p))


if __name__ == '__main__':
    main()
```

```python
from __future__ import print_function, division

from Card import Hand, Deck


class Hist(dict):


    def __init__(self, seq=[]):

        for x in seq:
            self.count(x)

    def count(self, x, f=1):

        self[x] = self.get(x, 0) + f
        if self[x] == 0:
            del self[x]


class PokerHand(Hand):

    all_labels = ['straightflush', 'fourkind', 'fullhouse', 'flush',
                  'straight', 'threekind', 'twopair', 'pair', 'highcard']

    def make_histograms(self):

        self.suits = Hist()
        self.ranks = Hist()

        for c in self.cards:
            self.suits.count(c.suit)
            self.ranks.count(c.rank)

        self.sets = list(self.ranks.values())
        self.sets.sort(reverse=True)

    def has_highcard(self):
        return len(self.cards)

    def check_sets(self, *t):

        for need, have in zip(t, self.sets):
            if need > have:
                return False
        return True

    def has_pair(self):
        return self.check_sets(2)

    def has_twopair(self):
        return self.check_sets(2, 2)

    def has_threekind(self):
        return self.check_sets(3)

    def has_fourkind(self):

        return self.check_sets(4)

    def has_fullhouse(self):

        return self.check_sets(3, 2)

    def has_flush(self):

        for val in self.suits.values():
            if val >= 5:
                return True
        return False

    def has_straight(self):

        ranks = self.ranks.copy()
        ranks[14] = ranks.get(1, 0)

        return self.in_a_row(ranks, 5)

    def in_a_row(self, ranks, n=5):

        count = 0
        for i in range(1, 15):
            if ranks.get(i, 0):
                count += 1
                if count == n:
                    return True
            else:
                count = 0
        return False

    def has_straightflush(self):

        s = set()
        for c in self.cards:
            s.add((c.rank, c.suit))
            if c.rank == 1:
                s.add((14, c.suit))

        for suit in range(4):
            count = 0
            for rank in range(1, 15):
                if (rank, suit) in s:
                    count += 1
                    if count == 5:
                        return True
                else:
                    count = 0
        return False

    def has_straightflush(self):

        d = {}
        for c in self.cards:
            d.setdefault(c.suit, PokerHand()).add_card(c)

        for hand in d.values():
            if len(hand.cards) < 5:
                continue
            hand.make_histograms()
            if hand.has_straight():
                return True
        return False

    def classify(self):

        self.make_histograms()

        self.labels = []
        for label in PokerHand.all_labels:
            f = getattr(self, 'has_' + label)
            if f():
                self.labels.append(label)


class PokerDeck(Deck):

    def deal_hands(self, num_cards=5, num_hands=10):

        hands = []
        for i in range(num_hands):
            hand = PokerHand()
            self.move_cards(hand, num_cards)
            hand.classify()
            hands.append(hand)
        return hands


def main():

    lhist = Hist()

    n = 10000
    for i in range(n):
        if i % 1000 == 0:
            print(i)

        deck = PokerDeck()
        deck.shuffle()

        hands = deck.deal_hands(7, 7)
        for hand in hands:
            for label in hand.labels:
                lhist.count(label)

    # print the results
    total = 7.0 * n
    print(total, 'hands dealt:')

    for label in PokerHand.all_labels:
        freq = lhist.get(label, 0)
        if freq == 0:
            continue
        p = total / freq
        print('%s happens one time in %.2f' % (label, p))


if __name__ == '__main__':
    main()
```