Data Analysis

[1]:
```
import numpy as np
from typing import List, Tuple
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
import matplotlib as mpl
mpl.rcParams['figure.dpi']= 120
import os
import seaborn as sns
sns.set(style="whitegrid")

img_dir = 'images'
```

[2]:
```
data_path = '/home/tjy/repos/red-bag-data/all-csv/'

data_files = []
filenames = os.listdir(data_path)
for f in filenames:
    if '.csv' in f:
        data_files.append(os.path.join(data_path, f))

dfs: List[pd.DataFrame] = [pd.read_csv(f) for f in data_files]
dfs[0]
```

[2]:

	order	value
0	0	1.23
1	1	6.29
2	2	12.43
3	3	4.13
4	4	1.10
5	5	0.81
6	6	4.73
7	7	10.97
8	8	12.90
9	9	11.41

[3]:
```
data_df = []

for i in range(len(dfs)):
    df = dfs[i].values
    df = np.hstack([df, np.ones((10, 1)) * i])
    data_df.append(df)

data_df = np.vstack(data_df)
data_df = pd.DataFrame(data_df, columns=['order', 'money', 'trial'])
data_df.to_csv('data_df.csv')
```

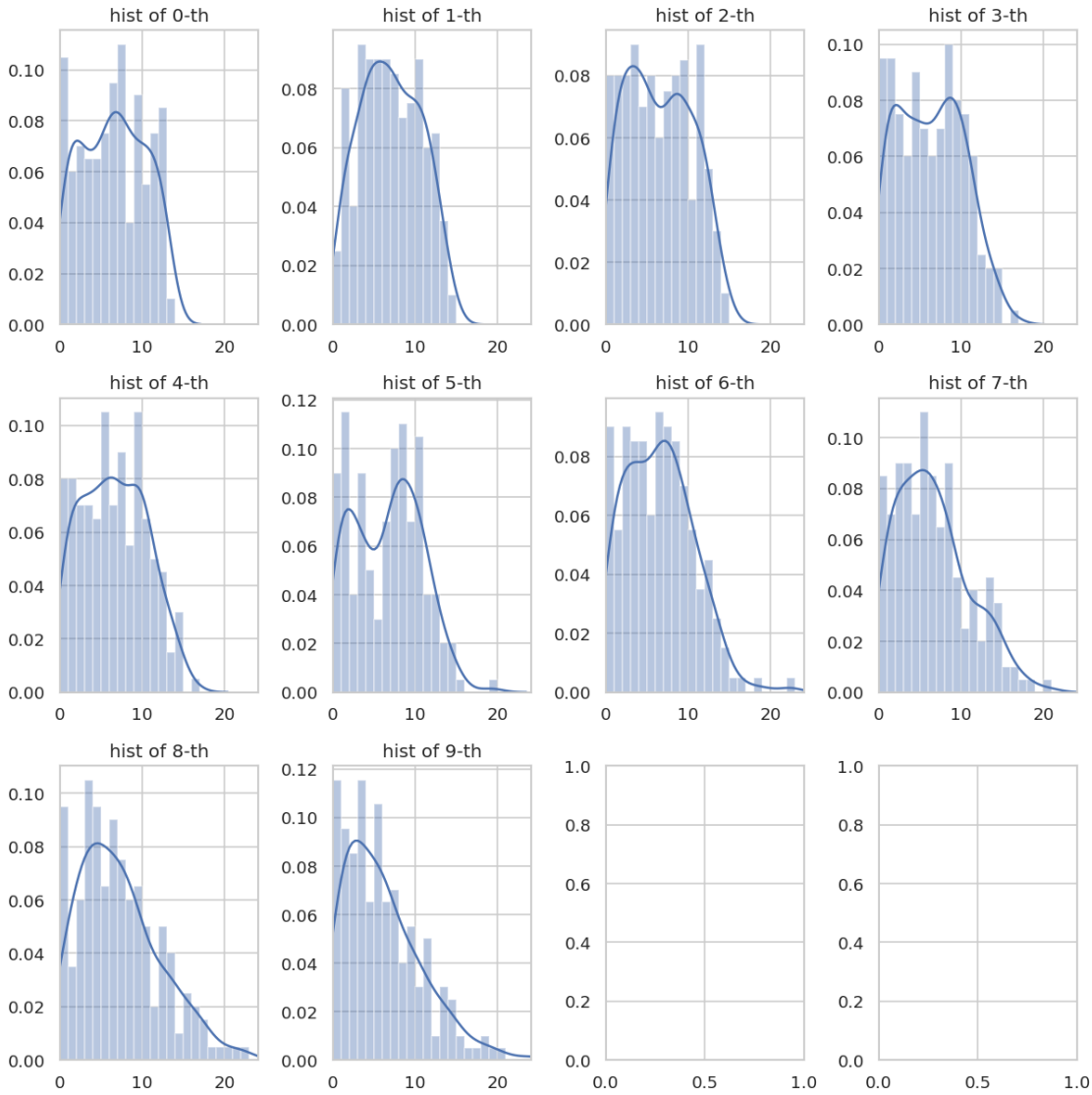
[4]:
```
n_trials = len(dfs)
data_dict = {'order': [i for i in range(10)]}

for i in range(n_trials):
    data_dict['trial_{}'.format(i)] = dfs[i]['value'].tolist()
data = pd.DataFrame(data_dict)
data.set_index('order', inplace=True)
np.savetxt('trials.csv', data.values)
# data
```

1 histogram

[5]:
```
def plot_hist_for_players(data1, bin_size: float = 1.0):
    fig, axs = plt.subplots(3, 4)
    fig.set_size_inches(10.24, 10.24)
    axs = axs.flat
    xlim = np.max(data1.money)
    bins = np.arange(0.0, xlim + 0.1, step=bin_size)
    for i in range(10):
        _data1 = data1[data1.order == i].money.values
        sns.distplot(_data1, bins=bins, label="true data", ax=axs[i])
        axs[i].set_title('hist of {}-th'.format(i))
        axs[i].set_xlim([0, xlim])
    # axs[0].legend()
    fig.tight_layout()
    plt.savefig(os.path.join(img_dir, "distribution-true.png"))
    plt.show()

plot_hist_for_players(data_df)
```



2 Data profile

[14]:
```
pd.DataFrame(data.T).describe()
```

[14]:

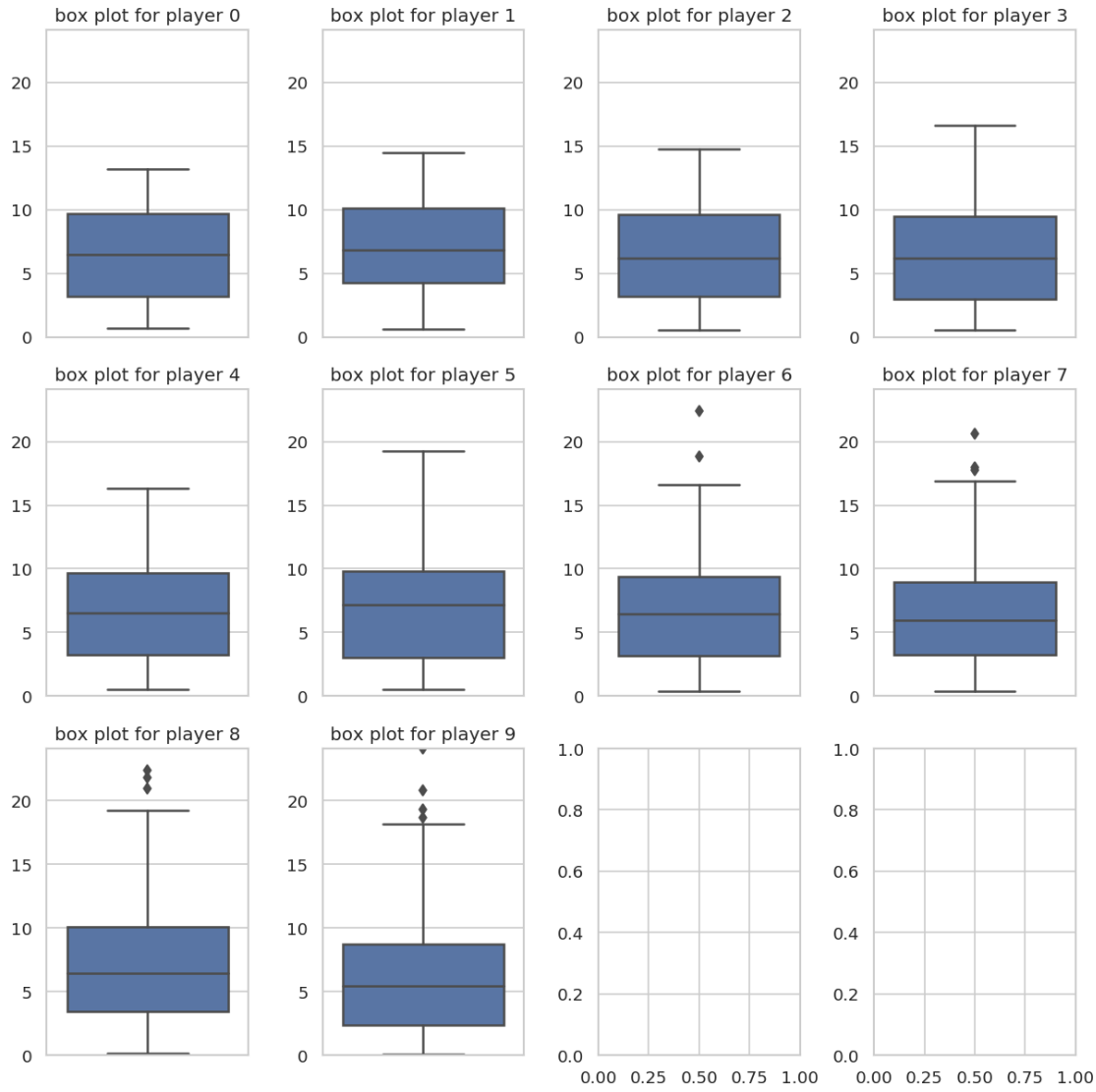
order	0	1	2	3	4	5
count	200.000000	200.000000	200.000000	200.000000	200.0000	200.000000
mean	6.524450	7.014650	6.517550	6.357850	6.5826	6.618050
std	3.823247	3.621468	3.894147	3.932249	3.9002	4.075096
min	0.660000	0.620000	0.550000	0.540000	0.5200	0.500000
25%	3.212500	4.232500	3.210000	2.942500	3.2550	3.047500
50%	6.510000	6.860000	6.175000	6.220000	6.5250	7.170000
75%	9.662500	10.137500	9.615000	9.442500	9.6875	9.802500

```
max    13.170000  14.490000  14.760000  16.630000  16.3300   19.270000
```

```
order           6           7           8           9
count  200.000000  200.000000  200.000000  200.000000
mean     6.516000    6.564750    7.223750    6.080350
std      4.089039    4.345045    4.875636    4.690813
min      0.420000    0.360000    0.190000    0.050000
25%      3.187500    3.275000    3.470000    2.350000
50%      6.465000    5.935000    6.460000    5.455000
75%      9.367500    8.975000   10.062500    8.742500
max     22.400000   20.580000   22.370000   24.100000
```

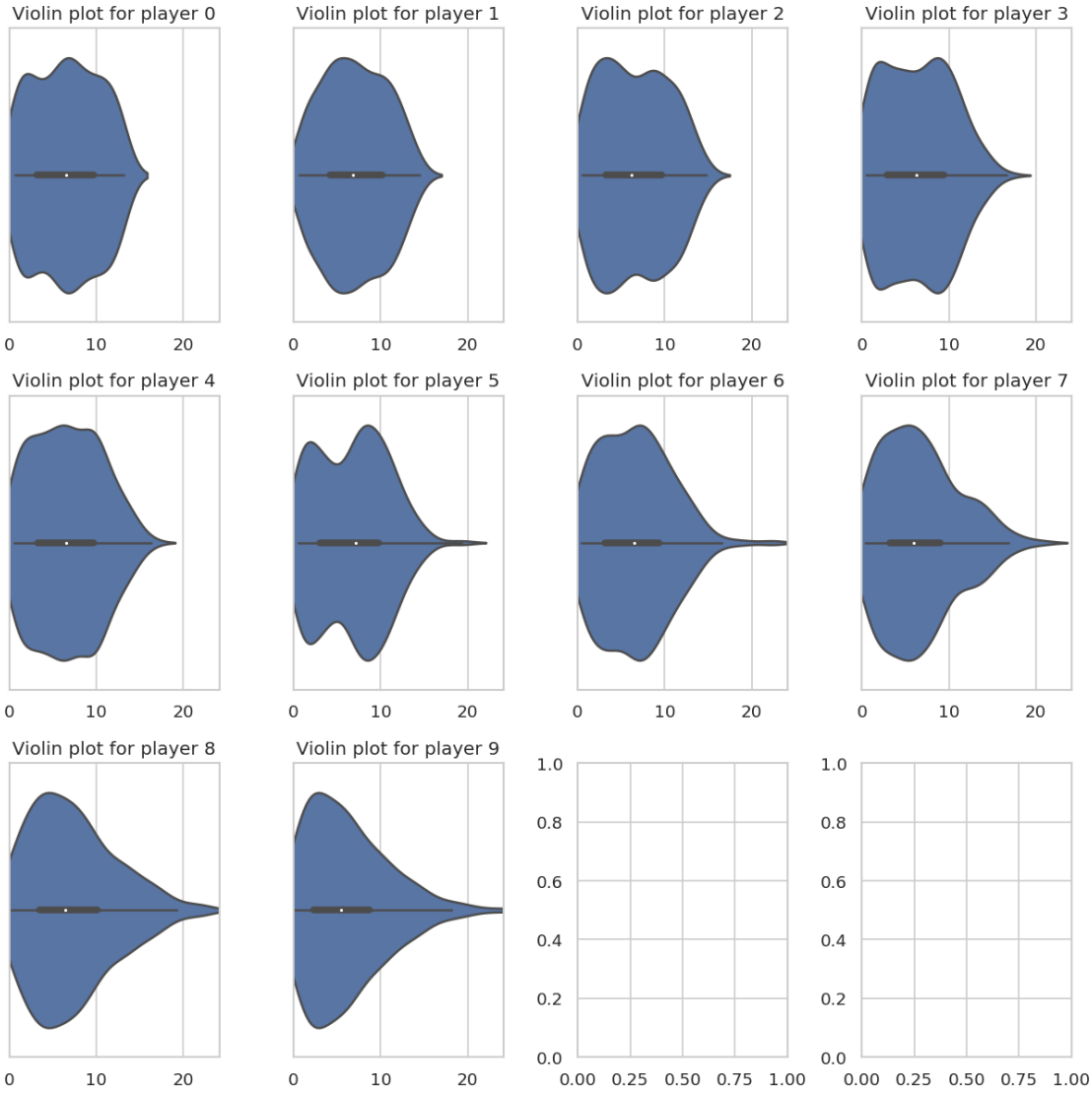
[7]:
```
def boxplot(data: pd.DataFrame, plot_name):
    fig, axs = plt.subplots(3, 4)
    fig.set_size_inches(10.24, 10.24)
    axs = axs.flat
    ylim = np.max(data.values)
    for i in range(10):
        sns.boxplot(data.values[i,], ax=axs[i], orient='v')
        title = 'box plot for player {}'.format(i)
        axs[i].set_title(title)
        axs[i].set_ylim([0, ylim])
    fig.tight_layout()
    plt.savefig(os.path.join(img_dir, 'box-plot-{}.png'.format(plot_name)))
    plt.show()

boxplot(data, 'true-data')
```



[8]:
```
def violinplot(data: pd.DataFrame, plot_name):
    fig, axs = plt.subplots(3, 4)
    fig.set_size_inches(10.24, 10.24)
    axs = axs.flat
    lim = np.max(data.values)
    for i in range(10):
        sns.violinplot(data.values[i,], ax=axs[i])
        title = 'Violin plot for player {}'.format(i)
        axs[i].set_title(title)
        axs[i].set_xlim([0, lim])
    fig.tight_layout()
    plt.savefig(os.path.join(img_dir, 'violin-plot-{}.png'.format(plot_name)))
    plt.show()
```

```
violinplot(data, 'true-data')
```



3 Luckiest Players & Least Lucky Players

[9]:
```
def plot_lucky_n_unlucky_players(data_lucky: pd.DataFrame, data_unlucky: pd.
↪DataFrame, bar_width: float = 0.35):
    labels = [str(i) for i in range(10)]
    lucky = data_lucky.values.flatten().tolist()
    unlucky = data_unlucky.values.flatten().tolist()
    fig, ax = plt.subplots()
    ax.bar(labels, lucky, width=bar_width, label='Lucky')
```

```
    ax.bar(labels, unlucky, width=bar_width, bottom=lucky,
           label='Unlucky')
    ax.set_ylabel('Frequency')
    ax.set_xlabel('Player order')
    ax.set_title('Count of luckiest and unluckiest players')
    ax.legend()
    plt.show()

# plot_lucky_n_unlucky_players(luckiest_player, unluckiest_player)
```

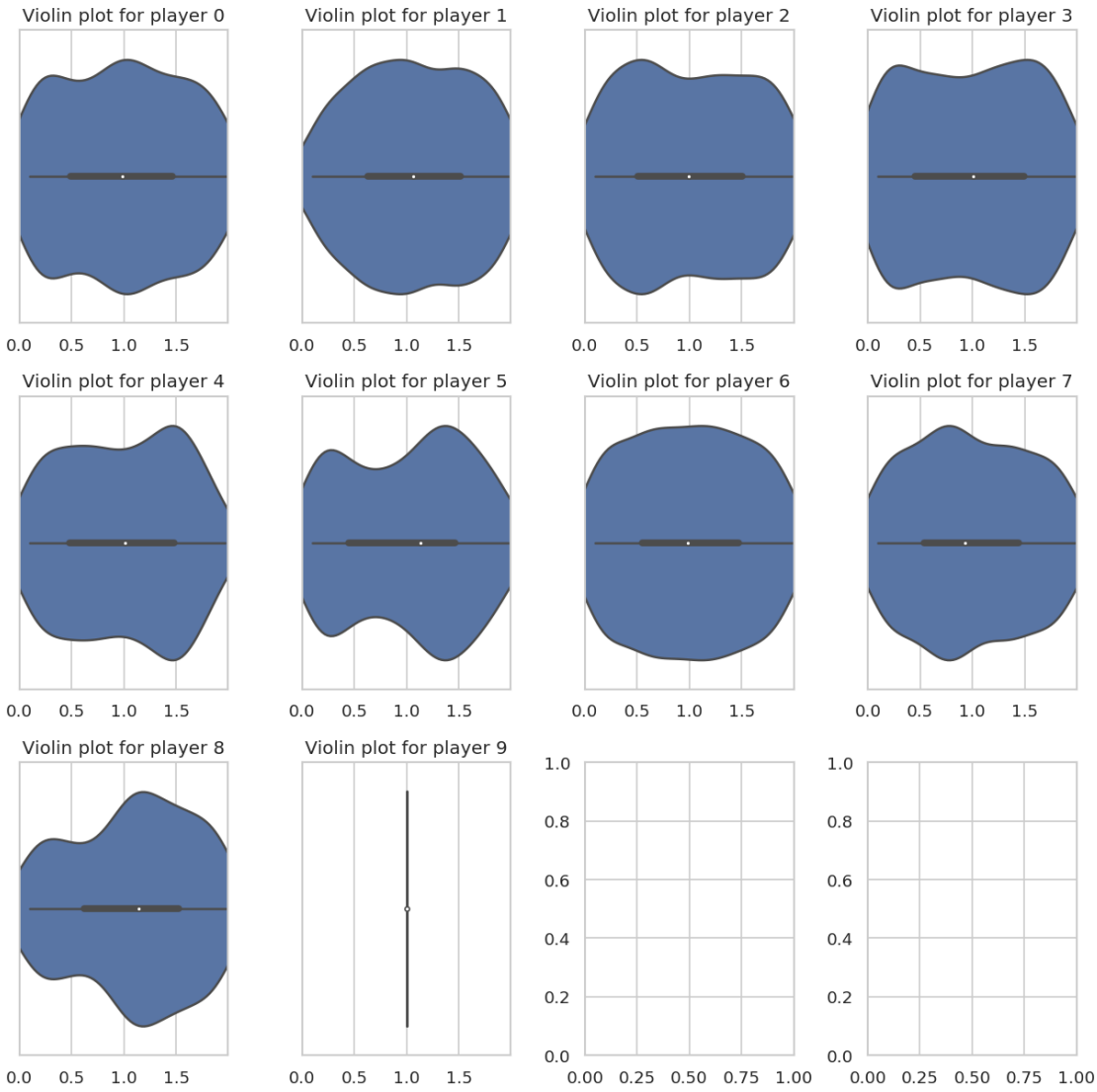
[15]:
```
def remain_average(data: pd.DataFrame, money=66.0, n_trials=200):
    data = data.values
    data_cumsum = np.cumsum(data, axis=0)
    data_cumsum = np.vstack([np.zeros((1, n_trials)), data_cumsum[:-1]])
    remaining = money - data_cumsum
    remain_n_players = np.arange(1, 11)[::-1].reshape(10, 1)
    remain_n_players = np.repeat(remain_n_players, n_trials, axis=1)
    remaining /= remain_n_players # no need for the last player
    k = data / remaining
    return k

k = remain_average(data)
k_df = pd.DataFrame(k[:-1].flatten())
k_df.describe()
```

[15]:

	0
count	1800.000000
mean	1.008518
std	0.567789
min	0.097968
25%	0.524374
50%	1.032778
75%	1.486377
max	1.997090

[11]:
```
k_data = pd.DataFrame(k)
violinplot(k_data, 'k')
```

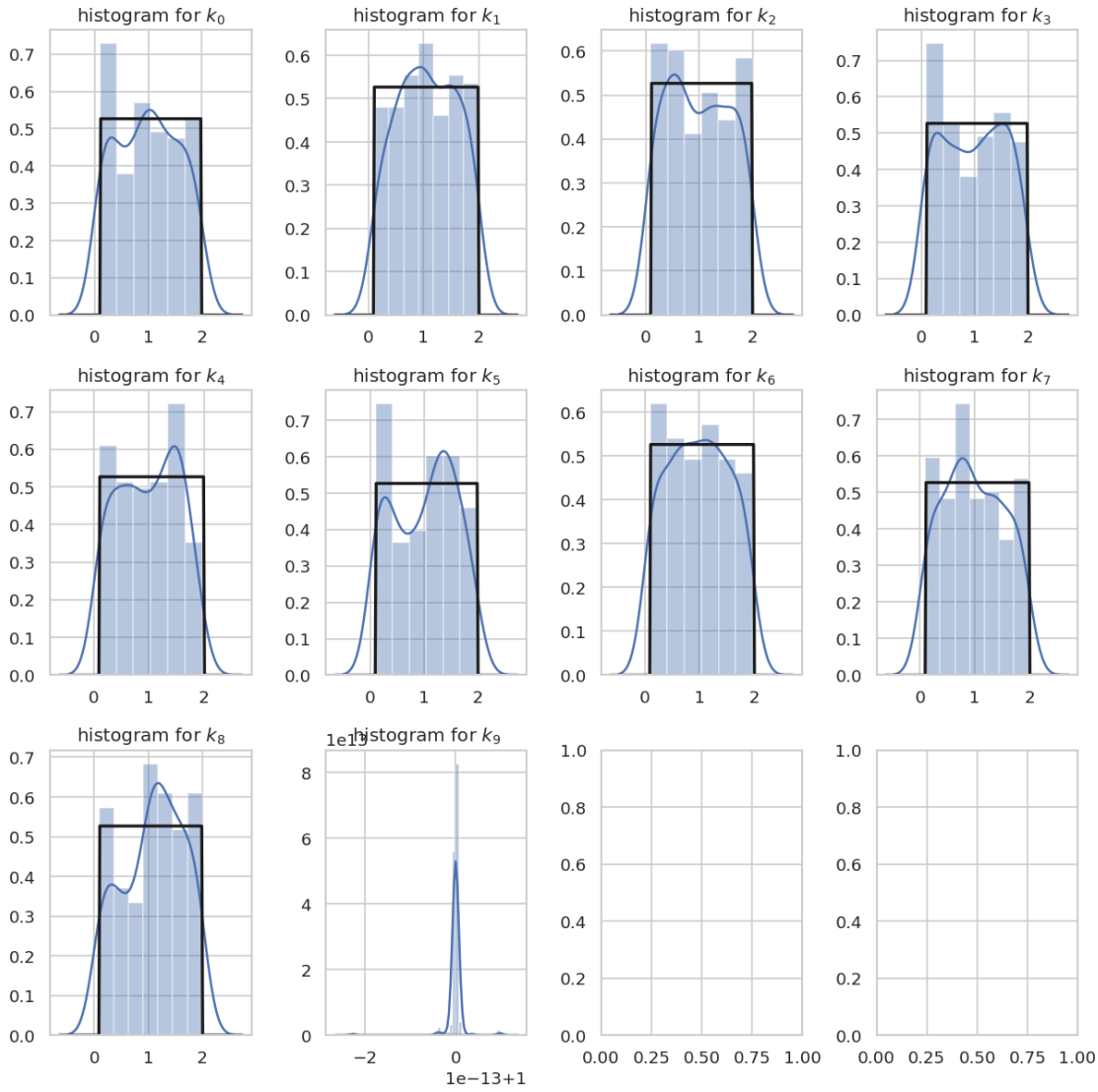


[12]:
```
from scipy.stats import uniform

def plot_dists_fit(data: np.ndarray, dist=uniform):
    fig, axs = plt.subplots(3, 4)
    fig.set_size_inches(10.24, 10.24)
    axs = axs.flat
    d = dist.fit(data.flatten())
    for i in range(10):
        sns.distplot(data[i,], ax=axs[i])
        title = 'histogram for $k_{}$'.format(i)
        axs[i].set_title(title)
        # plot the PDF
        xmin, xmax = axs[i].get_xlim()
```

```
            x = np.linspace(xmin, xmax, data.shape[1])
            p = dist.pdf(x, *d)
            axs[i].plot(x, p, 'k', linewidth=2)
        fig.tight_layout()
        plt.savefig(os.path.join(img_dir, 'k-histogram.png'))
        plt.show()
        return d

plot_dists_fit(k)
```



[12]: (0.09796806966618292, 1.899122282554338)

[12]:

sim_competition.py

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

sns.set(style="whitegrid")

__all__ = ['RedBag']


def trunc(values, decs=0):
    return np.trunc(values * 10 ** decs) / (10 ** decs)


class RedBag:
    def __init__(self, n_bags: int, money: float):
        self.n_remain = self.n_bags = n_bags
        self.money_remain = self.money = money

    def get_money(self):
        """
        Get money from this red bag
        :return: how much money of a new red bag
        """
        assert self.money_remain >= 0
        if self.n_remain == 1:
            money = self.money_remain
            self.money_remain = 0
            self.n_remain = 0
        else:
            min_ = 0.01
            max_ = self.money_remain / self.n_remain
            money = np.random.uniform(0, 2.0, 1)[0] * max_
            money = max(min_, money)
            money = trunc(money, decs=2)
            self.n_remain -= 1
            self.money_remain -= money
        return money


def sim_trial(n_bags: int, money: float):
    rb = RedBag(n_bags, money)
    trials = [rb.get_money() for _ in range(n_bags)]
    return trials


def sim_player_money(n_players=10, n_trials=10000, money=66.0):
    np.random.seed(1024)
    data = np.asarray([sim_trial(n_players, money) for _ in range(n_trials)]).T
    ranks = get_ranks(data)
    player_money = np.zeros(n_players)
    lucky = np.argmax(data, axis=0)
    u, c = np.unique(lucky, return_counts=True)

    # get remaining player money
    player_money[u] -= money * c
    player_money += np.sum(data, axis=1)

    # get number of luckiest for each player
    n_lucky = np.empty(n_players)
    n_lucky[u] = c
    return player_money, ranks, n_lucky


def get_ranks(array: np.ndarray):
    idx = array.argsort(axis=0)
    ranks = np.empty_like(idx)
    for i in range(ranks.shape[1]):
        ranks[idx[:, i], i] = np.arange(array.shape[0])
    return ranks
```

```
def normalize(data: np.ndarray, low=-1.0, high=1.0) -> np.ndarray:
    return low + (high - low) * (data - np.min(data)) / np.ptp(data)


def sim_trial1(n_players: int, money: float, trial_i: int):
    """
    :return: [order, money, trial]
    """
    rb = RedBag(n_players, money)
    money = np.asarray([rb.get_money() for _ in range(n_players)]).reshape(n_players, 1)
    trial = np.ones((n_players, 1), dtype=int) * trial_i
    order = np.arange(n_players, dtype=int).reshape(n_players, 1)
    return np.hstack([order, money, trial])


def sim_trials(n_trials=200, n_players=10, money=66.0):
    np.random.seed(1024)
    data = np.vstack([sim_trial1(n_players, money, i) for i in range(n_trials)])
    data = pd.DataFrame(data, columns=['order', 'money', 'trial'])
    data = data.astype({'order': int, 'money': float, 'trial': int})
    return data


if __name__ == '__main__':
    n_trials = 10000
    money = 66.0
    data1 = [sim_trials(n_trials, np, money) for np in range(3, 25)]
    data = [sim_player_money(np) for np in range(3, 25)]
    n = len(data)
    bar_width = 0.38
    for i in range(n):
        fig, ax = plt.subplots()

        data_, _, _ = data[i]
        n_players = data_.size
        labels = np.asarray(list(range(n_players)))
        # normalize
        data_ = normalize(data_)
        plt.bar(labels - bar_width / 2, data_, label='Remaining money', color='#a6cee3', width=bar_width)

        # find the number of luckiest
        data1_ = data1[i]
        n_p = len(data1_.order.unique())
        idx = data1_.groupby(['trial'])['money'].transform(max)
        idx = idx == data1_['money']
        lucky = data1_[idx]
        n_lucky = lucky.groupby(['order']).order.count()
        # normalize
        n_lucky = normalize(n_lucky.values, low=0, high=1.0)
        plt.bar(labels + bar_width / 2, n_lucky, label='Number of luckiest', color="#edd1cb", width=bar_width)

        ax.set_xlabel('Player order')
        ax.set_ylabel('Normalized value')
        ax.set_title('Red bag competition for {} players'.format(n_players))
        ax.set_xticks(labels)
        ax.set_xticklabels([str(i) for i in labels])
        ax.tick_params(axis='x', which='major', labelsize=10)
        plt.legend()
        fig.tight_layout()
        plt.savefig('competition-{}-players.png'.format(n_players))
        plt.close(fig)
```

Simulation

[2]:
```
import numpy as np
import pandas as pd
import os
from wechat_red_bag_simulation.sim_competition import RedBag
from matplotlib import pyplot as plt
%matplotlib inline
import matplotlib as mpl
mpl.rcParams['figure.dpi']= 120

import seaborn as sns
sns.set(style="whitegrid")

img_dir = 'images'
```

1 Algorithm for generating money

[3]:
```
def sim_trial(n_players: int, money: float, trial_i: int):
    """
    :return: [order, money, trial]
    """
    rb = RedBag(n_players, money)
    money = np.asarray([rb.get_money() for _ in range(n_players)]).
,→reshape(n_players, 1)
    trial = np.ones((n_players, 1), dtype=int) * trial_i
    order = np.arange(n_players, dtype=int).reshape(n_players, 1)
    return np.hstack([order, money, trial])


def sim_trials(n_trials=200, n_players=10, money=66.0):
    np.random.seed(1024)
    data = np.vstack([sim_trial(n_players, money, i) for i in range(n_trials)])
    data = pd.DataFrame(data, columns=['order', 'money', 'trial'])
    data = data.astype({'order': int, 'money': float, 'trial': int})
    return data

data = sim_trials()
```

```
data
```

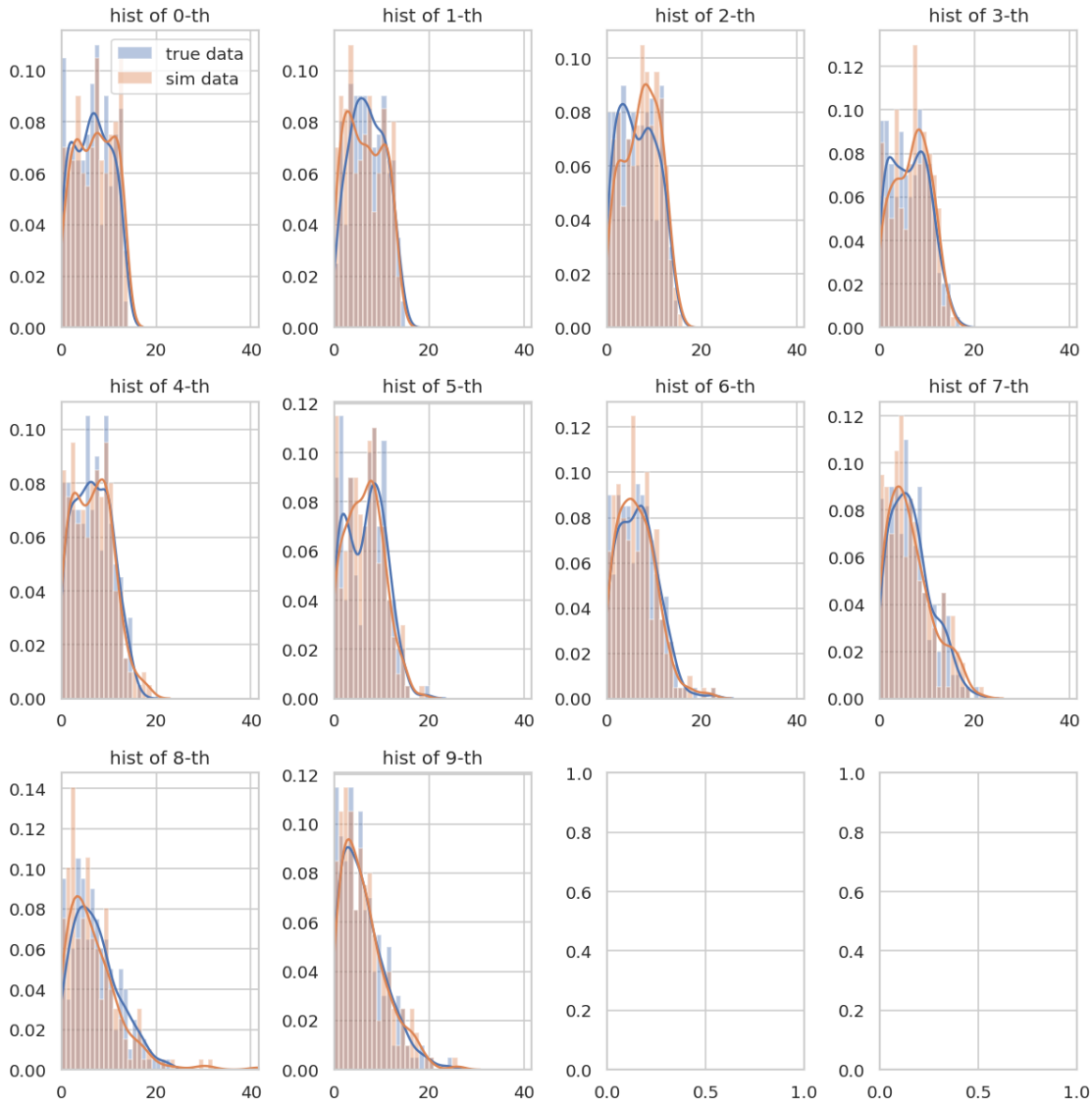
[3]:

	order	money	trial
0	0	8.54	0
1	1	12.72	0
2	2	5.80	0
3	3	7.32	0
4	4	6.31	0
...	...	...	...
1995	5	8.19	199
1996	6	3.84	199
1997	7	7.12	199
1998	8	9.17	199
1999	9	0.30	199

[2000 rows x 3 columns]

[4]:
```
def plot_hist_for_players(data1, data2, bin_size: float = 1.0):
    fig, axs = plt.subplots(3, 4)
    fig.set_size_inches(10.24, 10.24)
    axs = axs.flat
    xlim = max(np.max(data1.money), np.max(data2.money))
    bins = np.arange(0.0, xlim + 0.1, step=bin_size)
    for i in range(10):
        _data1 = data1[data1.order == i].money.values
        _data2 = data2[data2.order == i].money.values
        sns.distplot(_data1, bins=bins, label="true data", ax=axs[i])
        sns.distplot(_data2, bins=bins, label="sim data", ax=axs[i])
        axs[i].set_title('hist of {}-th'.format(i))
        axs[i].set_xlim([0, xlim])
    axs[0].legend()
    fig.tight_layout()
    plt.savefig(os.path.join(img_dir, "distribution-true-and-sim.png"))
    plt.show()

data_true = pd.read_csv('data_df.csv', index_col=0)
plot_hist_for_players(data_true, data)
```



H_0: Our algorithm of generating money create the same distribution as the one that the experiment data is sampled from.

So we reject H_0 if $p < 1\%$, where p is obtained by calculating Kolmogorov-Smirnov statistic on 2 samples data.

See kstest.py

2 Luckiest & Unluckiest Players

[5]:
```
data_df = pd.read_csv('data_df.csv', index_col=0)
data_df.order = data_df.order.astype(int)
data_df.trial = data_df.trial.astype(int)
data_df.describe()
```

[5]:

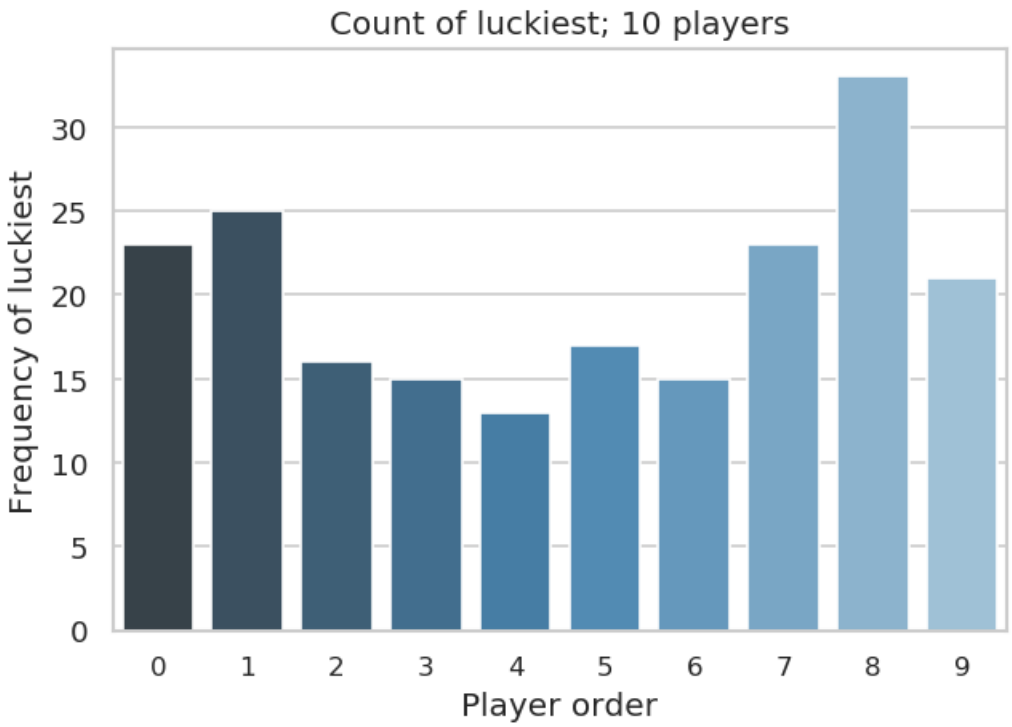
	order	money	trial
count	2000.000	2000.000000	2000.000000
mean	4.500	6.600000	99.500000
std	2.873	4.143568	57.748744
min	0.000	0.050000	0.000000
25%	2.000	3.210000	49.750000
50%	4.500	6.360000	99.500000
75%	7.000	9.550000	149.250000
max	9.000	24.100000	199.000000

[9]:
```
def plot_lucky(d: pd.DataFrame, plot_prefix=''):
    n_p = len(d.order.unique())
    idx = d.groupby(['trial'])['money'].transform(max)
    idx = idx == d['money']
    lucky = d[idx]
    n_lucky = lucky.groupby(['order']).order.count()
    lucky = pd.DataFrame({'order': n_lucky.index, 'n_lucky': n_lucky.values})

    # display(lucky)
    ratio = lucky.n_lucky.iloc[-1] / lucky.n_lucky.iloc[0]

    fig, ax = plt.subplots()
    sns.barplot(x='order', y='n_lucky', data=lucky,
                label='Lucky', ax=ax, palette="Blues_d")
    ax.set_ylabel('Frequency of luckiest')
    ax.set_xlabel('Player order')
    ax.set_title('Count of luckiest; {} players'.format(n_p))
    # We change the fontsize of ticks label
    ax.tick_params(axis='x', which='major', labelsize=10)
    # ax.legend()
    plt.savefig(os.path.join(img_dir, plot_prefix + 'lucky-{}-players.png'.
↪format(n_p)))
    plt.show()
    return ratio

plot_lucky(data_df, 'true-data-')
```



[9]: 0.9130434782608695

[10]:
```
n_trials = 10000
money = 66.0
ratios = []
for n_players in range(3, 25):
    data = sim_trials(n_trials, n_players, money)
    if n_players >= 8:
        ratios.append(plot_lucky(data))
print(ratios)
```

kstest.py

```
import numpy as np
# import pandas as pd
from scipy import stats
from wechat_red_bag_simulation.sim_competition import RedBag


def sim_trial(n_bags: int, money: float):
    rb = RedBag(n_bags, money)
    trials = [rb.get_money() for _ in range(n_bags)]
    return trials


if __name__ == '__main__':
    data_true = np.loadtxt('../trials.csv')
    n_trials = 200
    n_players = 10
    money = 66.0
    np.random.seed(1024)
    data = np.asarray([sim_trial(n_players, money) for i in range(n_trials)]).T
    # data_df = pd.DataFrame(data.T)
    # with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    #     print(data_df.describe())
    ks_stats = [stats.ks_2samp(data_true[i][data_true[i] != 0],
                               data[i][data[i] != 0])
                for i in range(data_true.shape[0])]
    ks_stats = np.asarray(ks_stats)

    md_str = """
    |order| p |reject?|
    |-----|---|-------|
    |0    |{0}|  {10}|
    |1    |{1}|  {11}|
    |2    |{2}|  {12}|
    |3    |{3}|  {13}|
    |4    |{4}|  {14}|
    |5    |{5}|  {15}|
    |6    |{6}|  {16}|
    |7    |{7}|  {17}|
    |8    |{8}|  {18}|
    |9    |{9}|  {19}|
    """

    ps = (ks_stats[:, 1])
    md_str = md_str.format(*ps, *(ps < 0.05))
    # Markdown(md_str)
    print(md_str)
```