Problem 1. Binomial Sim of Card Drawing:

In [2]:
```
#just one experiment, consisting of 100 trials

#import stuff:
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd

#start trial:
n_trials = 100

#sim card pulling
card_pull = rnd.choice(['Red Card', 'Black Card'], size = n_trials)
red_count = np.sum(card_pull == 'Red Card')
black_count = np.sum(card_pull == 'Black Card')
print(f'Red Cards: {red_count}, Black Card: {black_count}')
```

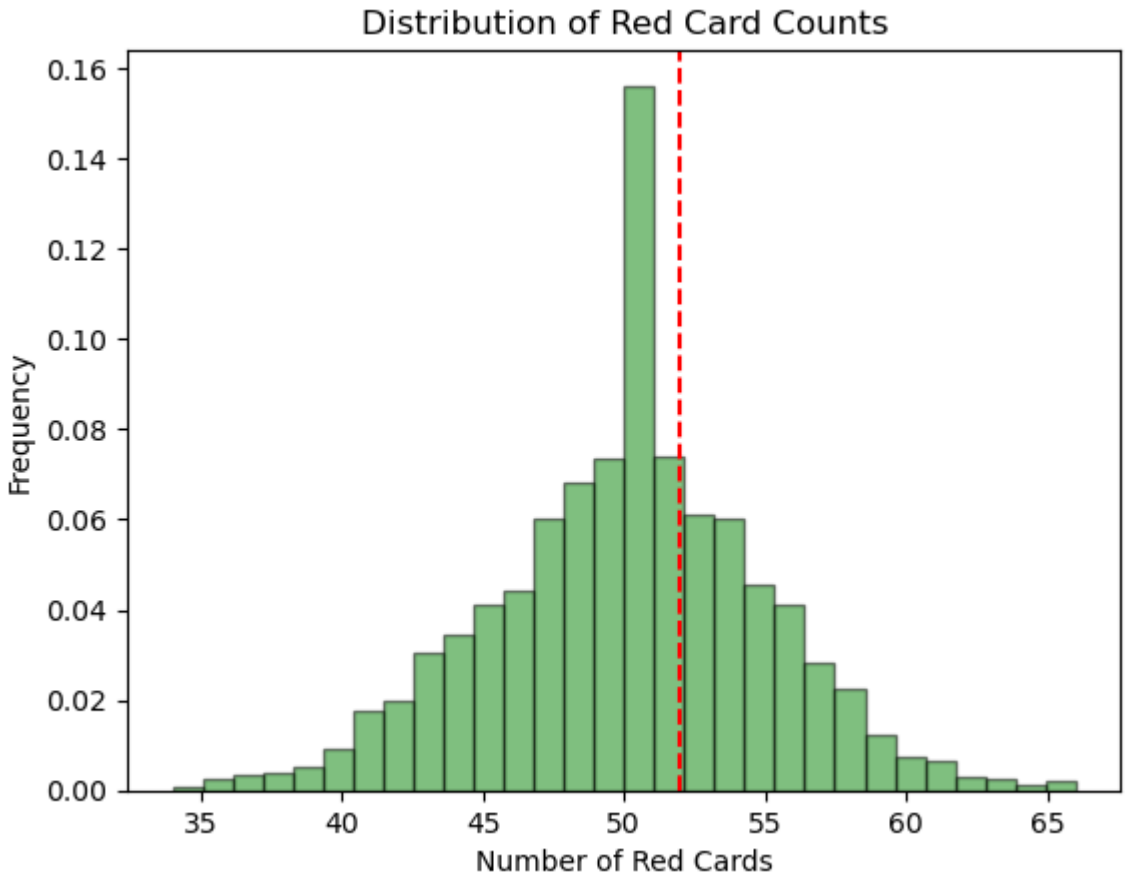
```
Red Cards: 48, Black Card: 52
```

Based on this, there is a 52% chance of pulling a red card! Slightly above the expected 50%. In my opinion, this wouldn't raise any *red* flags.

Problem 2: Binomial vs. Gaussian Approx.

In [7]:
```
#now make a simulation consisting of 2000 experiments (each with 100 trials)

n_trials = 100
n_experiments = 2000
n_red = np.zeros(n_experiments)


for i in range(n_experiments):
    card_pull = rnd.choice(['Red Card', 'Black Card'], size = n_trials)
    red_count = np.sum(card_pull == 'Red Card')
    n_red[i] = red_count
```

In [8]:
```
# Plot the distribution of heads counts
plt.hist(n_red, bins = 30,
         density = True,
         color = 'green',
         edgecolor = 'black',
         alpha = 0.5)
plt.xlabel('Number of Red Cards')
plt.ylabel('Frequency')
plt.title('Distribution of Red Card Counts')

# let's add a vertical line at our original result (52 red cards)
plt.axvline(x=52, color='r', linestyle='--')

plt.show()
```



In [12]:
```
#mean/std of simulated distribution
```

In [32]:
```
print(f'The simulated distributions\'s mean is : {np.mean(n_red):.2f}')
```

```
The simulated distributions's mean is : 50.08
```

In [33]:
```
print(f'The simulated distributions\'s standard deviation is : {np.std(n_red):
```

```
The simulated distributions's standard deviation is : 4.93
```

In [ ]:
```
#mean/std using Gaussian approx:
```

- mean = $n*p$ (where *n* is sample size and *p* is the probability of a red card)
- standard error = $\sqrt{n*p*(1-p)}$

In [34]:
```
print("The mean using the Gaussian formula is", 100 * .5)
```

```
The mean using the Gaussian formula is 50.0
```

In [36]:
```
print("The standard deviation using the Gaussian formula is",(100*.5*.5)**(1/2
```

```
The standard deviation using the Gaussian formula is 5.0
```

The Gaussian formula mean/std is extremely close to the data simulated 2000 times. This makes sense because as the sample size increases, the mean/std get closer to the true value.

Problem 3: Simulating an Election in a Fictional Country:

In [3]:
```
#setting the parameters woo hoo:
voter_turnout=np.array([.27,.45,.68,.52])
voter_count=np.array([10000,14000,9000,6000])
candidate_A_polls=np.array([.65,.45,.53,.42])
num_sims=2000

A_results = np.zeros(num_sims)
```

calculation:

since the winner is based just on popular vote the voting process will look like this:

state 1 voting: votes for candidate A in state 1= voter count *voter turnout* candidate A polls

then we'll calculate **state 1 voting+ state 2 voting+ state 3 voting+ state 4 voting** to find the total candidate A votes

lastly, we'll divide the total candidate A votes by the total vote across all 4 states and if that number is higher than 50%, that means candidate A won!

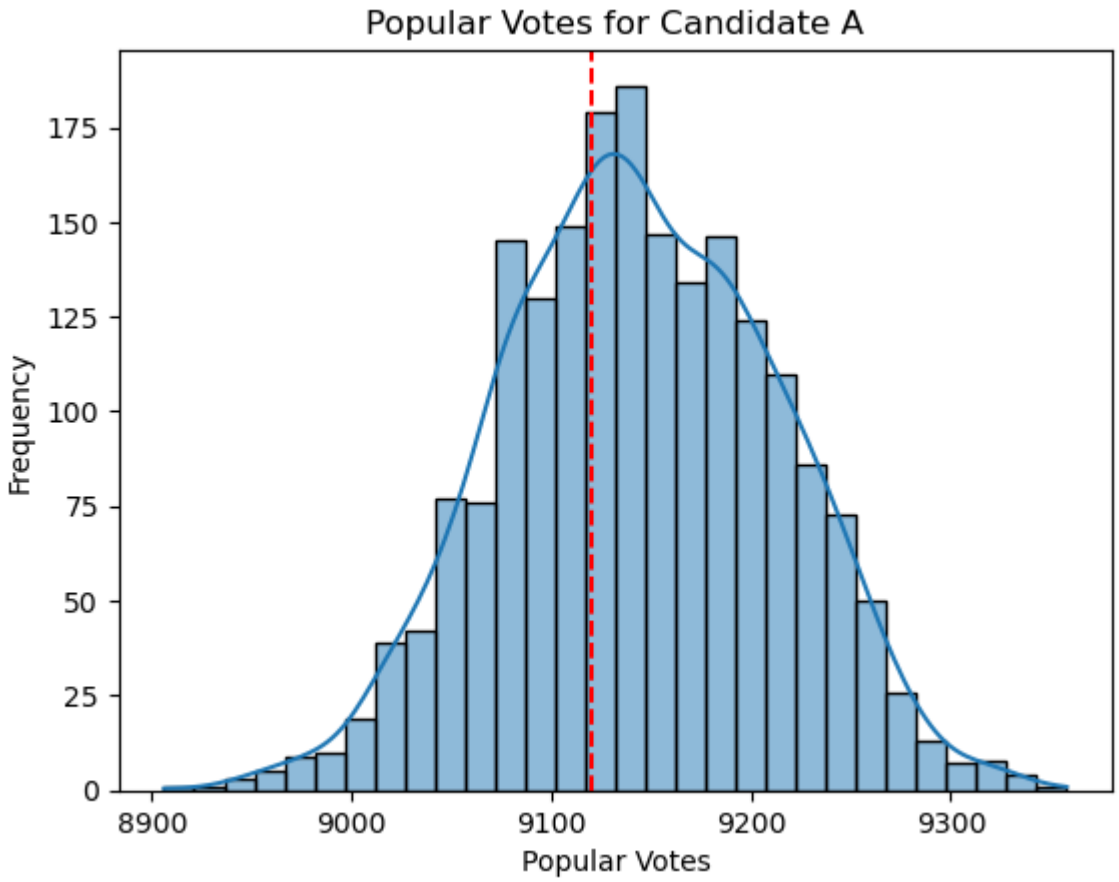
In [4]:
```
for i in range(num_sims):
    popular_votes_A = 0


    for voters, turnout, poll in zip(voter_count, voter_turnout, candidate_A_p
        #calc how many people actually voted
        people_who_voted=voters*turnout
        votes_A = np.random.binomial(people_who_voted, poll) # Simulate the po
        popular_votes_A += votes_A # Add the popular votes for this state to t

    #store results
    A_results[i] = popular_votes_A
```

In [5]:
```
np.mean(A_results)
```

Out[5]:
```
9143.4865
```

In [7]:
```
import seaborn as sns
sns.histplot(A_results, kde=True, bins=30)
plt.title('Popular Votes for Candidate A')
plt.xlabel('Popular Votes')
plt.ylabel('Frequency')
plt.axvline(x=9120, color='r', linestyle='--')
```

Out[7]:
```
<matplotlib.lines.Line2D at 0x177380350>
```



If all my code is correct, the graph seems to indicate that *slightly more than half of the time* Candidate A wins. It is not a super clear cut win or lose, unfortunately.

Problem 4: Effects of Sample Size on the Distribution of Sums

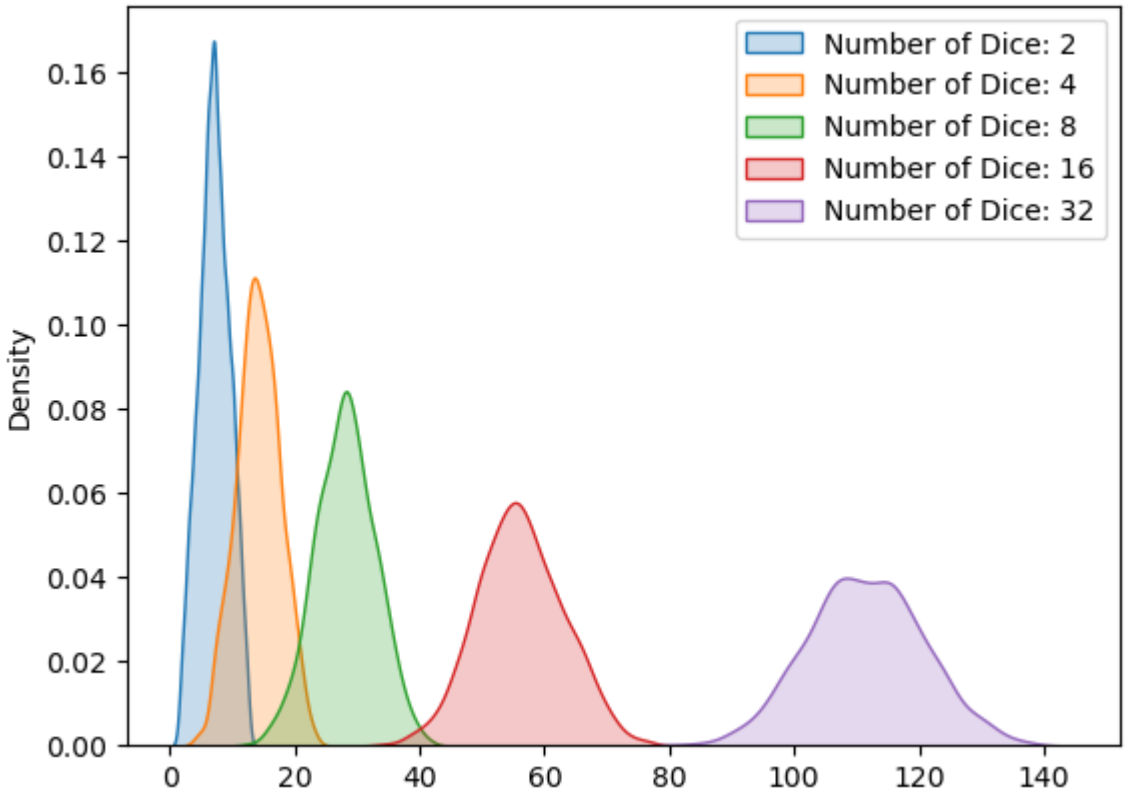
In [ ]:
```
n_dice = 10
n_trials = 2000
dice_sums = np.zeros(n_dice*6 + 1) # Initialize the histogram


# Simulate rolling the dice
for i in range(n_trials) :
    dice_rolls = np.random.randint(1, 7, size=n_dice)
    # Sum the rolls
    dice_sum = np.sum(dice_rolls)

    #store the sum into the array
    dice_sums[dice_sum]

# Plot the results
plt.bar(np.arange(2, 61), dice_sums[2:])
plt.xlabel('Dice Sum')
plt.ylabel('Frequency')
plt.title('Dice Sum Results')
plt.show()
```

In [147…
```
def dice_dist(n_dice, trials):
```

```
    dice_sums = []

    for i in range(trials):

        dice_rolls = np.random.randint(1, 7, size=n_dice)

        # Sum the rolls
        dice_sum = np.sum(dice_rolls)

        #store the sum into the array
        dice_sums.append(dice_sum)

    sns.kdeplot(dice_sums, label=f'Number of Dice: {n_dice}', alpha=0.25, fill=
```

In [148…

```
dice = np.array([2,4,8,16,32])

for i in dice:
    dice_dist(i, 2000)
    plt.legend()
```



As the number of dice increase, the sum of the dice also increases. The sampling distribution also seems to flatten as the number of dice increases.