

1 Exercise 1

1.1 1a

```
1 def temp_tester(normal_temp):  
2     def test_temp(temp):  
3         return abs(temp - normal_temp) <= 1  
4     return test_temp
```

Listing 1: temp tester function

1.2 1b

One misunderstanding about 1a's description is whether it includes an equal sign in the description within 1 degree of normal temp'

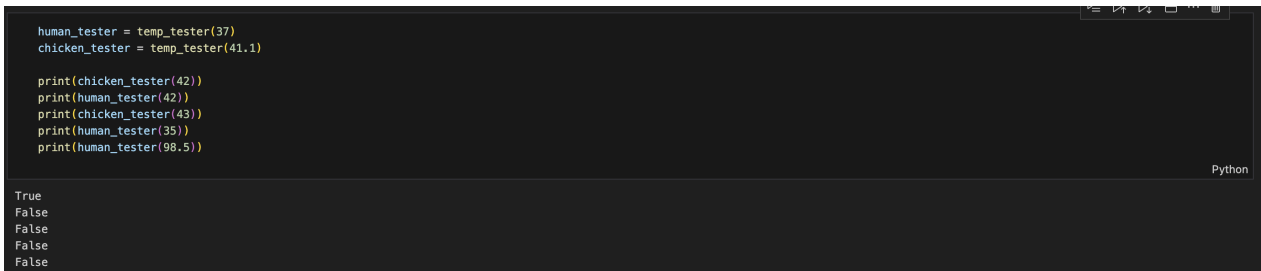
1.3 1c

Here is the code for testing the function above.

```
1 human_tester = temp_tester(37)  
2 chicken_tester = temp_tester(41.1)  
3  
4 print(chicken_tester(42))  
5 print(human_tester(42))  
6 print(chicken_tester(43))  
7 print(human_tester(35))  
8 print(human_tester(98.5))
```

Listing 2: temp tester testing code

And the output is shown in the figure 1.



```
human_tester = temp_tester(37)  
chicken_tester = temp_tester(41.1)  
  
print(chicken_tester(42))  
print(human_tester(42))  
print(chicken_tester(43))  
print(human_tester(35))  
print(human_tester(98.5))  
  
True  
False  
False  
False  
False  
False
```

Figure 1: 1c Output

2 Exercise 2

2.1 2a

The data we use in this exercise was retrieved from the great work by The New York Times[2].

2.2 2b

```
1 import matplotlib.pyplot as plt  
2 ##New case count  
3 def cal_new_cases(data, state_list):  
4     new_cases = {}  
5
```

```

6     for state in state_list:
7         state_list_data = data[data['state'] == state]
8         state_data = state_list_data.sort_values(by='date')
9         # Use diff() function instead of redundant calculation --inspired by ChatGPT
10        state_data['new_cases'] = state_data['cases'].diff().fillna(0)
11        new_cases[state] = state_data
12    return new_cases
13
14
15 def plot_new_cases(data, state_list):
16
17     new_cases = cal_new_cases(data, state_list)
18
19     plt.figure(figsize=(10, 6))
20
21     for state, state_data in new_cases.items():
22         plt.plot(state_data['date'], state_data['new_cases'], label=state)
23
24     plt.xlabel('Date', fontsize=12)
25     plt.ylabel('New Cases', fontsize=12)
26     plt.title('New COVID-19 Cases Over Time by State', fontsize=14)
27     plt.legend(title='State')
28     plt.xticks(rotation=45)
29
30     plt.show()

```

Listing 3: Visualization of New Cases

However, there is one limitation of this visualization. As shown in the figure 2, the lines mix up (overlap with each other) and we can hardly distinguish the lines based on their colors.

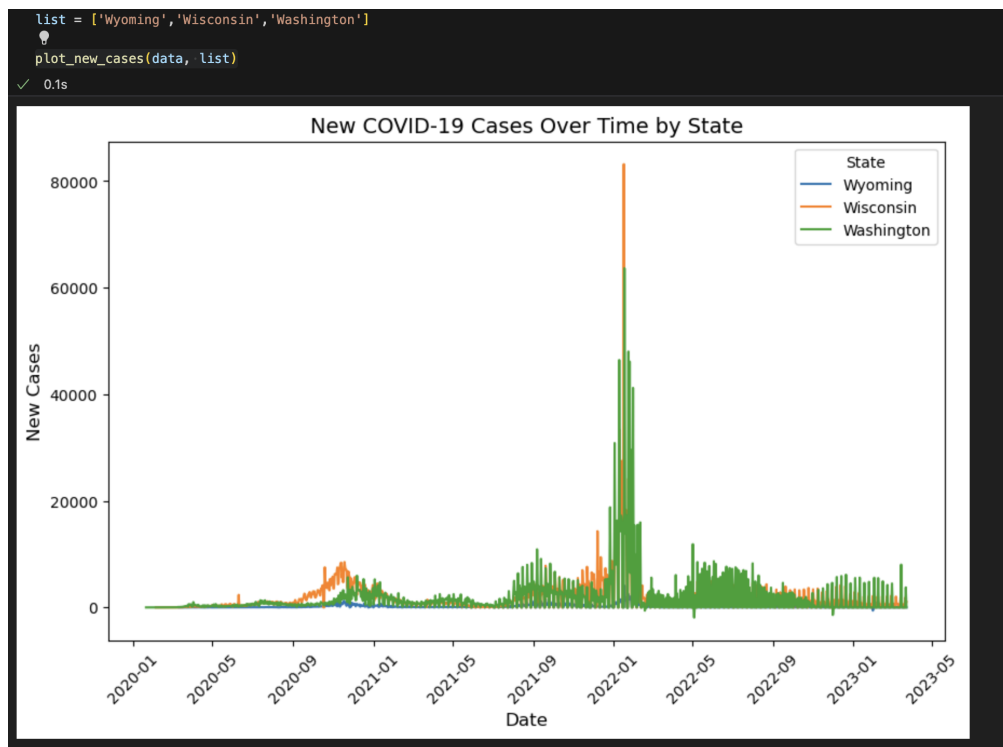


Figure 2: 2b Output

2.3 2c

```

1 def find_peak_cases_date(state):
2     state_data = data[data['state'] == state].copy()
3     state_data['new_cases'] = state_data['cases'].diff()
4     return state_data.loc[state_data['new_cases'].idxmax(), 'date']

```

Listing 4: Visualization of New Cases

In this function, I use the built-in functions `.loc` and `idxmax()` to locate the index of the peak, which has the largest new cases value. Given one state name, it can return the date of the peak cases. The image 3 is an example of using this function.

```
find_peak_cases_date("Washington")
✓ 0.0s
Timestamp('2022-01-18 00:00:00')
```

Figure 3: 2c Output

2.4 2d

```
1 def compare_peak(state1, state2):
2
3     state_list = [state1, state2]
4     new_cases_dict = cal_new_cases(data, state_list)
5
6     new_cases_1 = new_cases_dict[state1]['new_cases'].max()
7     new_cases_2 = new_cases_dict[state2]['new_cases'].max()
8
9     if new_cases_1 > new_cases_2:
10         print(f'The state with the higher peak is {state1} with {new_cases_1} new cases.')
11     else:
12         print(f'The state with the higher peak is {state2} with {new_cases_2} new cases.')
13     #Peak dates
14     date1 = find_peak_cases_date(state1)
15     date2 = find_peak_cases_date(state2)
16
17     if date1 < date2:
18         print(f'{state1} had its peak earlier on {date1.date()} than {state2} on {date2.date()}')
19     elif date1 > date2:
20         print(f'{state2} had its peak earlier on {date2.date()} than {state1} on {date1.date()}')
21     else:
22         print(f'{state2} had its peak on {date2.date()} the same date with {state1}.')
23     print(f'There are {abs((date1 - date2).days)} days between the peaks of {state1} and {state2}')
24
25     return abs((date1 - date2).days)
```

Listing 5: Compare peak cases

In this task, I created a function that compares the highest number of daily new cases between two states and then makes the report. The demonstration of using this function is in the figure 4.

```
days = compare_peak('Washington', 'New York')
✓ 0.0s
Python
The state with the higher peak is New York with 90132.0 new cases.
New York had its peak earlier on 2022-01-08 than Washington on 2022-01-18.
There are 10 days between the peaks of Washington and New York
```

Figure 4: 2d Output

2.5 2e

I ran the function *plot new cases* and had this output shown in figure 5 We can easily find that there is an outlier in this data. According to the definition of the variable **new cases**, it is a cumulative feature that its value should be greater or equal to zero. In this case, the peak that goes below 0 might be an outlier. It might be a typo when creating the dataset.

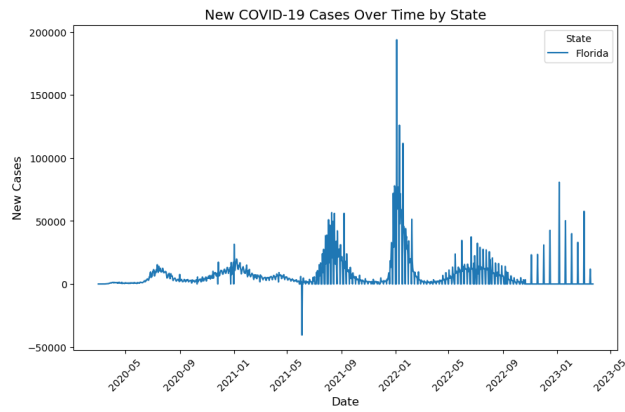


Figure 5: 2e Output

3 Exercise 3

3.1 3a

As shown in 6, the dataset has 152361 rows and 4 columns, namely *name*, *age*, *weight*, *eyecolor*.

	name	age	weight	eyecolor
0	Edna Phelps	88.895690	67.122450	brown
1	Cara Yasso	9.274597	29.251244	brown
2	Gail Rave	18.345613	55.347903	brown
3	Richard Adams	16.367545	70.352184	brown
4	Krista Slater	49.971604	70.563859	brown
...
152356	John Fowler	23.930833	71.532569	blue
152357	Diana Shuffler	21.884819	67.936753	brown
152358	Kevin Cuningham	87.705907	60.074646	brown
152359	James Libengood	21.727666	81.774985	brown
152360	Cathleen Ballance	10.062236	34.327767	brown

152361 rows x 4 columns

Figure 6: 3a Output

3.2 3b

```

1 import matplotlib.pyplot as plt
2
3 # Compute the age statistics
4 def compute_age_statistics(data):
5     age_mean = data['age'].mean()
6     age_std = data['age'].std()
7     age_min = data['age'].min()
8     age_max = data['age'].max()
9
10    print(f"Mean: {age_mean}")
11    print(f"Standard Deviation: {age_std}")
12    print(f"Minimum Age: {age_min}")
13    print(f"Maximum Age: {age_max}")
14
15    return age_mean, age_std, age_min, age_max
16
17 # Plot histogram
18 def plot_age_histogram(data, bins=10):
19     plt.figure(figsize=(8, 6))
20     plt.hist(data['age'], bins=bins, edgecolor='black')
21     plt.title('Age Distribution')
22     plt.xlabel('Age')
23     plt.ylabel('Frequency')
24     plt.grid(True)
25     plt.show()

```

```

26
27 compute_age_statistics(data)
28 plot_age_histogram(data, bins=25)

```

Listing 6: Analyze Age Distribution

The printed results are shown in figure 7. I chose 50 bins which means I separated the plot into 50 intervals. This can clearly indicate the pattern of this distribution. I observed that the distribution is approximately uniform for ages between 0 and 64, and the frequency drops afterward meaning that the old population is less in this dataset.

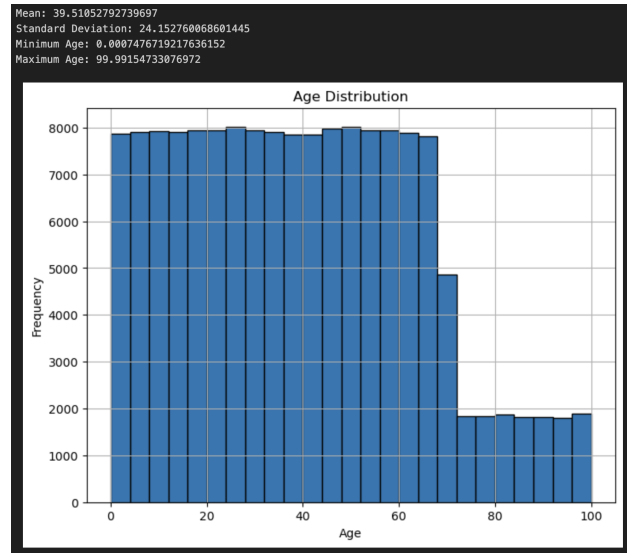


Figure 7: 3b Output

3.3 3c

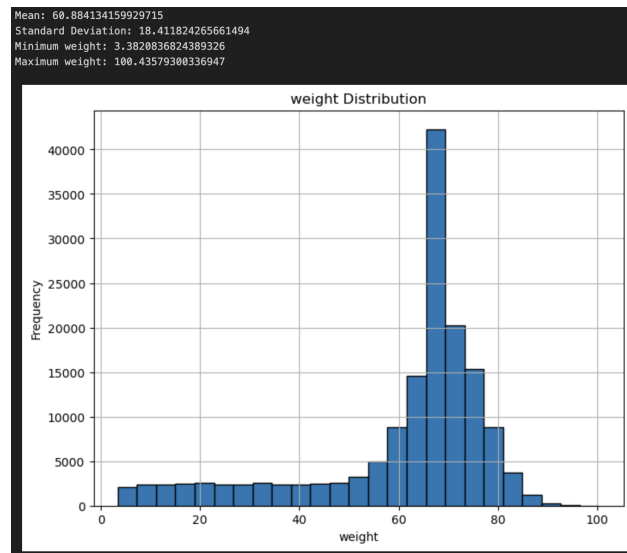


Figure 8: 3c Output

The figure 8 shows the pattern of weight distribution. Different from the age distribution, we can find that it similarly shows a long-tailed normal distribution. The peak shows the most frequent weight among the population.

3.4 3d

Then we create a scatter plot to visualize a correlation between weight and age. Generally speaking, from this figure, we can tell that people's weight and age have a linear correlation before the age of twenty. Furthermore,

we find an outlier in this figure. To filter this outlier, we designed a function to locate this outlier based on the range of its age and weight.



Figure 9: 3d Scatter plot

```
1 def find_outlier_person(data):
2     # Filter the data based on age and weight range
3     result = data[(data['age'] > 40) & (data['weight'] < 25)]
4
5     if not result.empty:
6         print(result[['name', 'age', 'weight']])
7     return result['name']
```

Listing 7: Identify outlier

Figure 10 shows the result of the outlier.

```
find_outlier_person(data)
✓ 0.0s

      name  age  weight
537  Anthony Freeman  41.3    21.7

537  Anthony Freeman
Name: name, dtype: object
```

Figure 10: 3d Outlier identification

4 Exercise 4

4.1 4a

```
1 import random
2
3 def generate_population(n, d):
4     population = [True] * d + [False] * (n - d)
5     random.shuffle(population)
6     return population
```

Listing 8: Population Generation

4.2 4b

```
1 def randomize_protocal(population,s):
2     sample = random.sample(population,s)
3     responses = []
4     for i in sample:
5         flip = random.random()
6         if flip>=0.5:
7             flip = random.random()
```

```

8         responses.append(flip >= 0.5)
9     else:
10         responses.append(i)
11     return responses

```

Listing 9: Sample and Protocol Simulation

4.3 4c

For this question, there are two ways to think of it. One is frequency-based, and the other is Bayesian-based logic. After discussing this with Professor Robert, the first one might be what is expected of this question.

If thinking of this question with the view of a frequentist, we regard the proportion in the samples as the proportion in the real population. Also, based on the rule of designed simulation protocol, we should adjust/correct the proportion of the Drug takers. We have two assumptions. The first assumption is that for each flipping, the probability of each side is 0.5. The second assumption is that no matter the size of the sample if the number of drug users is less than 1/4 of the population, we set the number of drug users after adjustment/corrections as 0. That is, if the proportion of drug users in the sample is smaller than the possibility of liars who said they are drug users, we regard this proportion of drug users as negligible in the entire population.

```

1 def estimate_drug_users(n, d, s):
2     """
3     Parameters:
4     - n: total population size.
5     - d: true number of drug users.
6     - s: sample size.
7     """
8     population = generate_population(n, d)
9     responses = randomize_protocal(population, s)
10
11     true_responses_count = sum(responses)
12     adjusted_true_count = (true_responses_count - 0.25 * s) / 0.5
13     if adjusted_true_count < 0:
14         adjusted_true_count = 0
15     proportion_true_responses = adjusted_true_count / s
16     estimated_drug_users = proportion_true_responses * n
17
18     return round(estimated_drug_users)

```

Listing 10: Estimation function

We also provide the other Bayesian-based idea. The Bayesian interpretation of probability can be viewed as an extension of propositional logic, allowing for reasoning with hypotheses—propositions whose truth or falsehood is uncertain [1]. In this case, we are using Bayesian inference to estimate the proportion of drug users in a population. The likelihood is influenced by the randomized response protocol, and the Beta distribution is used as the prior for the proportion of drug users. The Beta Distribution requires people to have a prior belief so that one can adjust the prior parameters based on the observations of the sampling. The code snippet is also provided. *Please note that the answers to the upcoming questions are all based on my first default idea.*

```

1 from scipy.stats import beta
2
3 def estimate_drug_users_bayesian(total_population, true_drug_users, sample_size):
4     population = generate_population(total_population, true_drug_users)
5     responses = randomize_protocal(population, sample_size)
6     true_count = sum(responses)##Number of True in responses
7     ## Adjust the count regarding the possibility of randomized answer, and do
8     standardization
9     adjusted_true_count = (true_count - 0.25 * sample_size) / 0.5
10    alpha_prior=true_count
11    beta_prior=sample_size-true_count
12    #Beta Distribution, recalculate the posterior based on the likelihood estimation
13    #We assume the likelihood is the observation in our trial
14    #alpha_posterior = alpha_prior + max(adjusted_true_count, 0)
15    #beta_posterior = beta_prior + sample_size - max(adjusted_true_count, 0)
16    alpha_posterior = round(alpha_prior * adjusted_true_count)
17    beta_posterior = sample_size - alpha_posterior
18    # Posterior distribution
19    #posterior_dist = beta(alpha_posterior, beta_posterior)
20    #estimated_proportion = posterior_dist.mean()
21    # Update posterior based on the adjusted true count
22    alpha_posterior = alpha_prior + max(adjusted_true_count, 0)
23    beta_posterior = beta_prior + (sample_size - max(adjusted_true_count, 0))

```

```

23
24 # Posterior distribution
25 posterior_dist = beta(alpha_posterior, beta_posterior)
26 estimated_proportion = posterior_dist.mean()
27
28 # Estimated number of drug users
29 estimated_drug_users = estimated_proportion * total_population
30
31 return round(estimated_drug_users)

```

Listing 11: Estimation function 2 the Bayesian method

4.4 4d

This code snippet shows the implementation of the functions provided above. Since we created the population and the sampling protocol randomly, the results of this implementation each time will change as well.

```

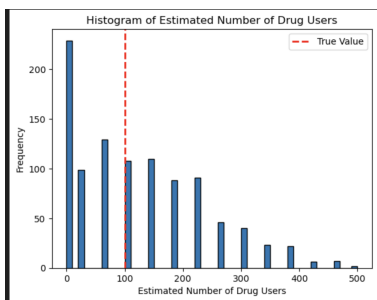
1 total_population = 1000
2 true_drug_users = 100
3 sample_size = 50
4
5 estimated_drug_users = estimate_drug_users(total_population, true_drug_users, sample_size)
6 estimated_drug_users

```

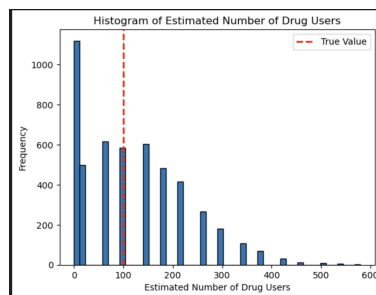
Listing 12: Run a simulation

4.5 4e

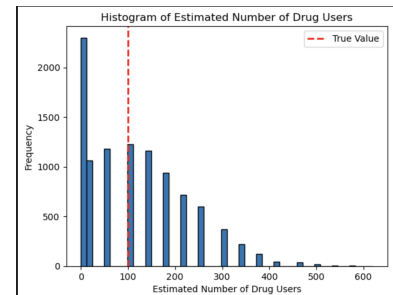
We set the number of bins in the histogram to 50 and changed the number of experiments. The figure shows the results. As we can observe when the number of experiments increases, the histogram shows a pattern of normal distribution and the mean value approaches the true number (100) of drug users in the population. The figure 11 shows the mean and variance of the simulation results (the number of simulations = 10000).



(a) Number of experiments = 1000



(b) Number of experiments = 5000



(c) Number of experiments = 10000

Figure 11: Distribution histogram outcome of multiple simulations.

The mean value = 116.466 and the variance is 11654.270844, shown in 12.

```

from scipy.stats import norm
import numpy as np
mean_estimate = np.mean(estimates)
variance_estimate = np.var(estimates)
print(mean_estimate, variance_estimate)
✓ 0.0s
116.466 11654.270844

```

Figure 12: Mean and Variance of simulation

4.6 4f

This code snippet shows how we explore the relationship between sample size and standard deviation of possible predictions.

```

1 n = 1000 # Total population
2 d1 = 100
3 d2 = 500
4 s_range = range(10, 1001, 10) # Sample sizes with margin = 10

```



```

5
6 mean_estimates_1, std_estimates_1 = simulate_estimates(n, d1, s_range)
7 mean_estimates_2, std_estimates_2 = simulate_estimates(n, d2, s_range)

```

Listing 13: Run a simulation

The figure 13 visualizes the result of the simulations with the x-axis as the number of sample sizes. We find that as the sample size increases, the mean of the estimation value approaches the real number of drug users.

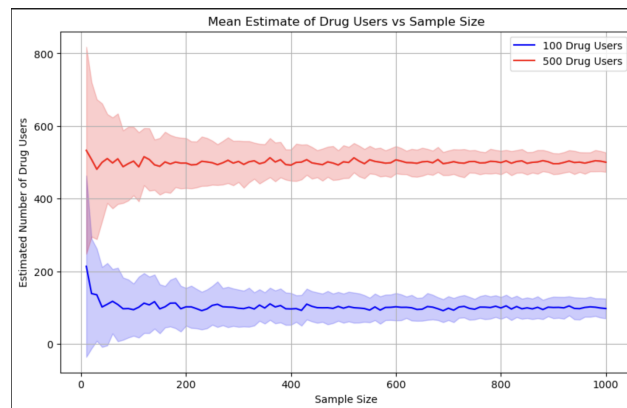


Figure 13: 4f visualization

References

- [1] Theodore Hailperin. 1996. *Sentential probability logic: Origins, development, current status, and technical applications*. Lehigh University Press.
- [2] The New York Times. 2021. Coronavirus (Covid-19) Data in the United States. <https://github.com/nytimes/covid-19-data>. Retrieved [2024-9-05].