

```
In [18]: # Initialize Otter
import otter
grader = otter.Notebook("HW5.ipynb")
```

Homework 5: Inference and Hypothesis Testing (50 points)

Name: Wonjae Oh:wonjae

Names and Onyens of fellow students you discussed Homework 5 problems and ideas:

- Students: Kallie, Layla, Nuhamin

We encourage discussing ideas and brainstorming with your peers, but the final text, code, and comments in this homework assignment MUST be 100% written by you as mentioned in syllabus.

Problem 1. Getting started on the project proposal (4 points)

This question is here to make sure you don't all work on your project proposal a day before. :)))

- You must pick a dataset as a group. There's no penalty for changing it down the road, but you might as well pick one now!

From R,

- Variable 1: psavert_df\$open
- Source: tq_get("^GSPC", from = "1985-01-01")
- Frequency: Daily
- Variable 2: psavert_df\$Personal_Savings_Rate
- Source: data("economics")
- Frequency: Monthly
- Variable 3: unemploy_df\$Unemployment
- Source: data("economics")
- Frequency: Monthly
- Reason to include Variable 1: My Epstein-Zin based "spending on financial asset" optimization model requires wealth at each period and the previous period's wealth as endogenous variables, along with a row-wise feature vector at each period. psavert_df\$open acts as the "spot price" of the S&P index at each period t and is an important column-wise feature vector because how expensive a financial asset is is definitely highly correlated with whether or not to purchase it at period t.
- Reason to include Variable 2: psavert_df\$Personal_Savings_Rate would be a column-wise feature vector that is a proxy for endogenous wealth at period t. It is necessary even after period 0 because people usually earn income from work and decide how much to invest, which further influences available wealth at each period t.
- Reason to include Variable 3: unemploy_df\$Unemployment is a column-wise feature vector that is kind of unrelated to W_t directly but still influences W_{t+1}. Our goal is to show how.

- Conduct individual data exploration: calculate **summary statistics** and create **at least two** basic visualizations of your dataset.

Try to create visualizations that are different from those created by your teammates.

```
In [10]: # To load dataset from a different folder, change the variable fn appropriately.
import pandas as pd
fn = 'sp500_df'
data = pd.read_csv(f'~/110-F25/project/S_and_P/{fn}.csv')
data.head()
```

Out[10]:

	symbol	date	open	high	low	close	volume	SP500_Close
0	^GSPC	1985-01-02	167.199997	167.199997	165.190002	165.369995	67820000.0	165.369995
1	^GSPC	1985-01-03	165.369995	166.110001	164.380005	164.570007	88880000.0	164.570007
2	^GSPC	1985-01-04	164.550003	164.550003	163.360001	163.679993	77480000.0	163.679993
3	^GSPC	1985-01-07	163.679993	164.710007	163.679993	164.240005	86190000.0	164.240005
4	^GSPC	1985-01-08	164.240005	164.589996	163.910004	163.990005	92110000.0	163.990005

```
In [12]: # Write your code here

# 0) Imports & style
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_context("notebook")

# 1) Load data
# Assumes four CSVs exist. If any is missing or names differ, we adapt below.
sp500_df = pd.read_csv('~/110-F25/project/S_and_P/sp500_df.csv')
try:
    SP500_simple_returns = pd.read_csv('~/110-F25/project/S_and_P/SP500_simple_returns.csv')
except FileNotFoundError:
    SP500_simple_returns = None

psavert_df = pd.read_csv('~/110-F25/project/S_and_P/psavert_df.csv')
unemploy_df = pd.read_csv('~/110-F25/project/S_and_P/unemploy_df.csv')

# 2) Date parsing + monthly key creation
# We standardize on a month-begin key called 'date_som' (start-of-month).
def _to_datetime(df):
    # Grab the first column that looks like date
    date_cols = [c for c in df.columns if 'date' in c.lower()]
    if not date_cols:
        raise ValueError("No date-like column found.")
    c = date_cols[0]
    df[c] = pd.to_datetime(df[c])
    return df, c

sp500_df, sp_date = _to_datetime(sp500_df)
psavert_df, ps_date = _to_datetime(psavert_df)
unemploy_df, un_date = _to_datetime(unemploy_df)

# create month-begin key (date_som) in each
sp500_df['date_som'] = sp500_df[sp_date].values.astype('datetime64[M]') # month start
psavert_df['date_som'] = psavert_df[ps_date].values.astype('datetime64[M]')
unemploy_df['date_som'] = unemploy_df[un_date].values.astype('datetime64[M]')

# 3) Ensure we have monthly S&P 500 returns (Value in percent or decimal).
# If SP500_simple_returns not provided, compute from sp500_df (daily).
# We'll use last trading day close each month, then pct_change.
if SP500_simple_returns is None:
    # pick the close column
    close_col = 'SP500_Close' if 'SP500_Close' in sp500_df.columns else 'close'
    if close_col not in sp500_df.columns:
        raise ValueError("Could not find a closing-price column (SP500_Close/close).")

    # Last observation per month:
    monthly_price = (
        sp500_df.sort_values(sp_date)
        .groupby('date_som', as_index=False)
        .tail(1)[[close_col]]
        .sort_values('date_som')
        .reset_index(drop=True)
    )
```

```

monthly_price['SP500_Returns'] = monthly_price[close_col].pct_change() # decimal (not %)
SP500_simple_returns = monthly_price[['date_som', 'SP500_Returns']]
else:
    # If provided file uses other names, normalize to our convention
    SP500_simple_returns, ret_date = _to_datetime(SP500_simple_returns)
    if 'SP500_Returns' not in SP500_simple_returns.columns:
        # try to infer a return-like column
        guess = [c for c in SP500_simple_returns.columns if 'return' in c.lower() or 'ret' in c.lower()]
        if not guess:
            raise ValueError("Could not find a return column in SP500_simple_returns.")
        SP500_simple_returns = SP500_simple_returns.rename(columns={guess[0]: 'SP500_Returns'})
    SP500_simple_returns['date_som'] = SP500_simple_returns[ret_date].values.astype('datetime64[M]')
    SP500_simple_returns = SP500_simple_returns[['date_som', 'SP500_Returns']]

# 4) Normalize column names for the macro series
# We want: Personal_Savings_Rate and Unemployment
# savings rate
if 'Personal_Savings_Rate' not in psavert_df.columns:
    # try common names: 'psavert', 'personal_savings_rate', 'psr', etc.
    candidates = [c for c in psavert_df.columns if 'save' in c.lower() or 'psavert' in c.lower()]
    if not candidates:
        raise ValueError("Personal savings rate column not found in psavert_df.")
    psavert_df = psavert_df.rename(columns={candidates[0]: 'Personal_Savings_Rate'})

# unemployment level/rate
if 'Unemployment' not in unemploy_df.columns:
    # try to pick a reasonable unemployment column
    candidates = [c for c in unemploy_df.columns if 'unemploy' in c.lower()]
    if not candidates:
        raise ValueError("Unemployment column not found in unemploy_df.")
    unemploy_df = unemploy_df.rename(columns={candidates[0]: 'Unemployment'})

# Retain only what we need
psavert_df = psavert_df[['date_som', 'Personal_Savings_Rate']]
unemploy_df = unemploy_df[['date_som', 'Unemployment']]

# 5) Join to a wide monthly table on date_som
df_wide = (
    SP500_simple_returns
    .merge(psavert_df, on='date_som', how='left')
    .merge(unemploy_df, on='date_som', how='left')
    .sort_values('date_som')
    .reset_index(drop=True)
)

# 6) Quick facet lines before standardizing
# This mirrors the usual ggplot "sanity check" of each series.
g = pd.melt(df_wide, id_vars='date_som',
            value_vars=['SP500_Returns', 'Personal_Savings_Rate', 'Unemployment'],
            var_name='Variable', value_name='Value')
g = g.sort_values(['Variable', 'date_som'])

gfacet = sns.FacetGrid(g, col='Variable', sharex=True, sharey=False, height=2.8, aspect=1.4)
gfacet.map_dataframe(sns.lineplot, x='date_som', y='Value')
gfacet.set_xlabels("Date (Monthly)")
gfacet.set_ylabels("Value")
gfacet.set_titles(col_template="{col_name}")
plt.tight_layout()
plt.show()

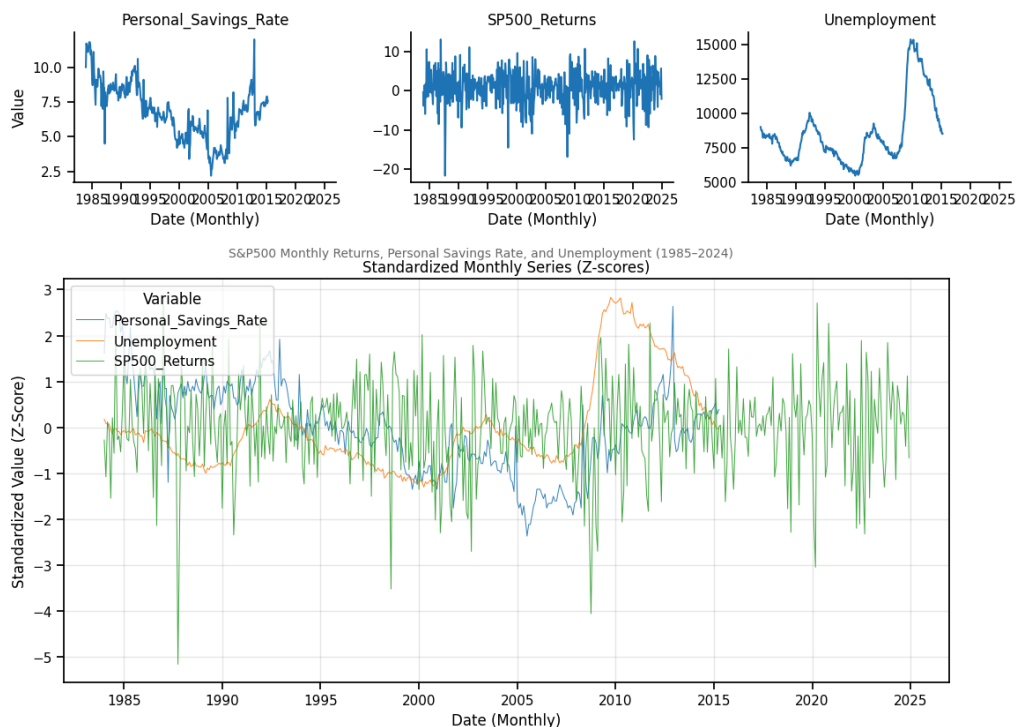
# 7) Z-score within each Variable (exact R logic: group_by -> mutate)
# R's sd uses ddof=1
g['Z_Score'] = g.groupby('Variable')['Value'].transform(
    lambda s: (s - s.mean()) / s.std(ddof=1)
)

# In case some groups are constant (std==0), guard to avoid inf/NaN in plot
g['Z_Score'] = g['Z_Score'].replace([np.inf, -np.inf], np.nan)

#
# 8) Final standardized overlay plot
#
plt.figure(figsize=(11,6))
sns.lineplot(data=g.sort_values('date_som'),
             x='date_som', y='Z_Score', hue='Variable',
             linewidth=0.7, alpha=0.9)

plt.title("Standardized Monthly Series (Z-scores)")
plt.suptitle("S&P500 Monthly Returns, Personal Savings Rate, and Unemployment (1985-2024)",
            y=0.94, fontsize=10, color='dimgray')
plt.xlabel("Date (Monthly)")
plt.ylabel("Standardized Value (Z-Score)")
plt.legend(title="Variable", loc='upper left')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

```



- What did you notice (3-4 sentences)?

Since the monthly returns are probably not fractionally differenced, I see very little drift for the monthly_return variable of SP500_Returns, but there could be drift in other variables. The goal of using Epstein Zin optimization is to minimize overfitting in an inherently volatile learning environment.

Problem 2. When there are many samples (22 points)

In HW4, we sampled 10% of penguins to demonstrate good sampling practices. In this problem, we'll use the same dataset to demonstrate Law of Large Numbers and Central Limit Theorem.

In [15]:

```
# Run this cell
import pandas as pd
penguins = pd.read_csv('~/.110-F25/demos/palmer_penguins.csv')
penguins.shape
penguins.head()
```

Out[15]:

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Sex	Delta 15 N (o/oo)	Delta 13 C (o/oo)	Comments
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1	18.7	181.0	3750.0	MALE	NaN	NaN	Not enough blood for isotopes.
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5	17.4	186.0	3800.0	FEMALE	8.94956	-24.69454	NaN
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3	18.0	195.0	3250.0	FEMALE	8.36821	-25.33302	NaN
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Adult not sampled.
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7	19.3	193.0	3450.0	FEMALE	8.76651	-25.32426	NaN

Problem 2.1 (2 points) Population and sample.

Here is our fictional situation. There are exactly 344 penguins left in the world (RIP). You're a researcher camping out in Antarctica, hoping to meet penguins and take their measurements. Due to logistical difficulties, you won't be able to gather all 344 of them in one place and record their characteristics. Select the most appropriate choices to complete the following sentences about this fictional situation. Make edits directly to the text.

- 344 penguins are (the population of interest) or (the sample)
- The average weight of the 344 penguins is (population mean) or (sample mean)
- The average weight of 34 out of 344 penguins is (a parameter) or (a statistic)

Problem 2.2 (2 points) Population mean and standard deviation.

Calculate the **population mean** and **population standard deviation** of **Body Mass**. Assign them to variables `pop_mean` and `pop_std`.

In [27]:

```
... # write your code here, replacing ...
import numpy as np

pop_mean = penguins["Body Mass (g)"].mean().tolist() # population mean
pop_std = penguins["Body Mass (g)"].std().tolist() # population std uses ddof=0

print(pop_mean)
print(pop_std)

4201.754385964912
801.9545356980956
```

Out [27]:

In [28]:

```
grader.check("Q2.2")
```

Out[28]:

Q2.2 passed!

Problem 2.3 (6 points) Law of Large Numbers

In lecture, we simulated a die roll by sampling from uniform distribution. Here, we will be sampling rows from the penguins table.

(1 points) Should these rows be sampled with replacement (`replace=True`) or without replacement (`replace=False`) to get closer to the assumptions of LLN? Why?

With replacement because the central limit theorem states, as the sample size gets larger, its distribution will be more Gaussian and standard error closer to the population standard deviation / square root of the sample size.

(3 points) Complete the code below that demonstrates LLN on penguins data. Similar to lecture demo, instead of sampling one row at a time inside the for loop, let's:

1. sample `max_k` of them all at once, then
2. inside the for loop as `k` changes,
3. take the mean of first `k` samples.

Make sure to have a horizontal line indicating the population mean.

Take a look at lecture slides and demo code for help.

In [34]:

```
import matplotlib.pyplot as plt
import seaborn as sns

max_k = 4000
# sample max_k penguins
# max_k_penguins = ...
all_means = []
col = 'Body Mass (g)' if 'Body Mass (g)' in penguins.columns else 'body_mass_g'
max_k_penguins = (
    penguins[col].dropna()
    .sample(n=max_k, replace=True, random_state=42)
    .to_numpy().astype(float)
)

for k in range(1, max_k):
    # first k out of max_k penguins
    # sampled_penguins = ...
    # calculate sample mean
    # sample_mean = ...
    # first k out of max_k penguins
    sampled_penguins = max_k_penguins[:k]
    # calculate sample mean (scalar float)
    sample_mean = sampled_penguins.mean()
    all_means.append(sample_mean) # save sample mean

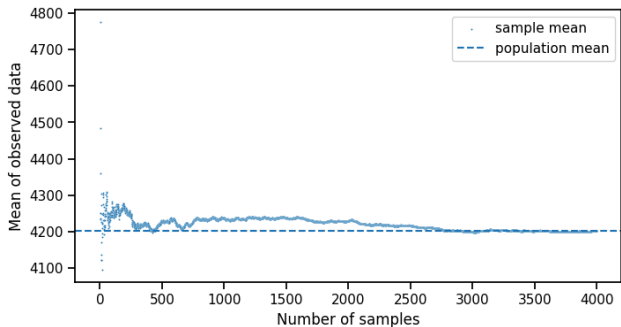
# population mean for horizontal reference
# pop_mean = penguins[col].dropna().mean()

# fig, ax = plt.subplots(1)
# sns.scatterplot(x=range(1, max_k), y=all_means, label='sample mean', s=2)
# ax.set_xlabel('Number of samples')
# ax.set_ylabel('Mean of observed data')
# horizontal line code goes here, replacing ...

# 3) plot
fig, ax = plt.subplots(1, figsize=(8,4))
sns.scatterplot(x=range(1, max_k), y=all_means, s=2, label='sample mean', ax=ax)
ax.set_xlabel('Number of samples')
ax.set_ylabel('Mean of observed data')
ax.axhline(pop_mean, linestyle='--', linewidth=1.5, label='population mean')
ax.legend()
```

Out[34]:

<matplotlib.legend.Legend at 0x7f3a3a3a3da0>



(2 points). Why is it enough to measure some penguins instead of all 344? How many penguins do you think you'd need to see before your estimate is "good enough"? Why?

Actually, the number of entries in a sample for the mean to converge is greater than 344, when sampling with replacement, according to the plot. Therefore, I believe approximately 3000 is a good number of penguins to select with replacement before my estimate is good enough.

Problem 2.4 (12 points) Central Limit Theorem

(3 points) What we explored in the LLN problem was the idea of sampling Penguins during one Antarctica trip. What happens if we made 2, 4, 10, 100s of Antarctica experiments? What can we say about the sample means from all of the different trips?

Complete the code below that demonstrates CLT on penguins data. It is very similar to lecture demo. Your `num_exp` and `k` can be set to any reasonable number at time of submission.

```
In [48]: import matplotlib.pyplot as plt
import seaborn as sns

num_exp = 200 # number of times you repeat the experiment

k = 320 # number of cute penguins you meet

all_means = []
col = 'Body Mass (g)' if 'Body Mass (g)' in penguins.columns else 'body_mass_g'
# population values to sample from each experiment (no pre-sampling)
max_k_penguins = penguins[col].dropna().to_numpy().astype(float)

all_means = []

for exp in range(num_exp):
    # sample k penguins (fresh i.i.d. sample each experiment)
    sampled_penguins = np.random.choice(max_k_penguins, size=k, replace=True)
    # calculate sample mean
    sample_mean = sampled_penguins.mean()
    all_means.append(sample_mean)

fig, ax = plt.subplots(1, figsize=(8,4))
sns.scatterplot(x=range(1, len(all_means)+1), y=all_means, s=2, label='sample mean', ax=ax)
ax.set_xlabel('Number of samples')
ax.set_ylabel('Mean of observed data')
ax.axhline(pop_mean, linestyle='--', linewidth=1.5, label='population mean')
ax.legend()

#### theoretical distribution according to CLT ####
from scipy.stats import norm
import numpy as np

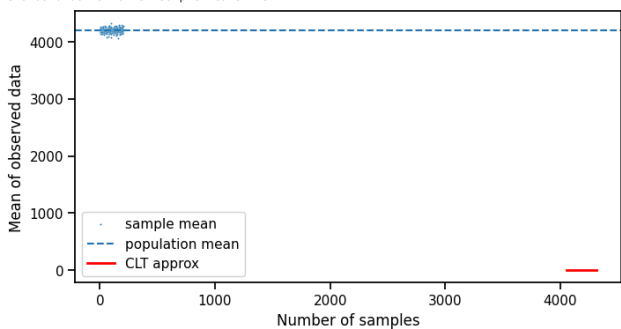
sigma_sampling = pop_std / np.sqrt(k)

xs = np.linspace(min(all_means), max(all_means), 200)
ys = norm.pdf(xs, pop_mean, sigma_sampling)
ax.plot(xs, ys, 'r-', lw=2, label='CLT approx')
ax.legend()

#####

print('num_experiment:', num_exp)
print('k (number of penguins):', k)
print('mean of sample means:', np.array(all_means).mean().round(1))
print('standard deviation of sample means:', np.array(all_means).std().round(1))
```

num_experiment: 200
k (number of penguins): 320
mean of sample means: 4195.3
standard deviation of sample means: 43.2



(2 points) Varying k

With `num_exp=200` fixed, run the code with different values of `k`, then fill out this table. The numbers are calculated for you in the last two print statements.

k	mean (of sample means)	std (of sample means)
20	4217.0	176.3
80	4191.3	97.4
320	4195.3	43.2

What trends did you notice? Try to be as detailed as possible.

The std decreases and the mean of samples becomes more and more a better approximator of the population mean.

(2 points) Modeling

I propose three models that might fit our observation of std (of sample means):

\$\$\$1. std \propto k^{-1}\$\$. \$\$\$2. std \propto k^{-1/2}\$\$. \$\$\$3. std \propto k^{-2}\$\$. \$\$\$

Using your previously calculated `pop_mean` value, fill in the table with what `std` values would look like according to the different models. This should feel similar to the ROUSE Age and Weight modeling exercises we did in the previous homework.

k	1. pop_mean/k	2. pop_mean/sqrt(k)	3. pop_mean/k^2
20			
80			
320			

Which model fits the data best and why?

[Write your answer here]

(2 points) Varying num_exp

This time with `k=80` fixed, run the code with different values of `num_exp`, then fill out this table.

num_exp	mean (of sample means)	std (of sample means)
200		
400		
800		

What trends did you notice?

[Write your answer here]

(1 points) Make a **histogram** of the body mass of all penguins with appropriate labels. This should be the 344 population penguins, not the sampled penguins.

In []: `# Your code here`

(2 points) Shape.

Does this population distribution look **skewed** or **symmetrical**? Does this population distribution look like **Gaussian**?

How does it compare with the shape of the **sample mean distribution** we plotted before? Why is that okay or not okay?

[Write your answer here]

Problem 3. Loaded question (24 points)

In this problem, you'll be analyzing the fairness of a 6-sided die.

A street vendor asks you if you'd like to play a game. The game is:

- pay \$10 to roll a 6-sided die 3 times
- if any of those rolls is a 6 he pays you \$11.

Problem 3.1 (4 points) Hypothesis Testing.

You decide to play the game 20 times, and out of your 60 total rolls of the die, you roll a 6 **four times**. You start to suspect something is not right, and you wonder if perhaps the die is NOT fair. In particular, you feel that there might be **less than fair chance** of rolling a 6, even though the street vendor insists it is fair. You decide to conduct a hypothesis test.

1. State the null hypothesis for this situation. Remember, you'll need to be able to "simulate under the null".
2. State the alternative hypothesis.
3. Which of the following statistics would you be appropriate for evaluating these hypotheses?

- a) cost to play the game
- b) number of 5's rolled
- c) proportion of 6's rolled
- d) total number of rolls

4. What is the value of this statistic for the 20 games you played? Assign it to `measured_statistic1` in the provided cell. (1 point autograder)

Write your answers to 1-4 here

In []: `# Assign the value of the statistic for the 20 games here
measured_statistic1 = ...`

In []: `grader.check("Q3.1.4")`

Problem 3.2 (10 points) Simulating under the null.

In this problem, you'll be writing code to simulate the test statistic for 60 rolls, and you'll run that simulation 1000 times.

Problem 3.2.1 (2 points).

First, write a code that returns a `list` named `rolls` containing simulated 6-sided die rolls for 60 rolls. Import whatever Python package you need to do this.

In []: `# write your code here
...
rolls = ...
This is sampling from uniform distribution
rolls[:10]`

In []: `grader.check("Q3.2.1")`

Problem 3.2.2 (2 points).

Next, write code to simulate 1000 times the test statistic for 60 die rolls, and store the test statistic in a table named `statistic1`.

In []: `import pandas as pd
statistic1 = pd.DataFrame(columns=['test statistic'])
for simulation in range(1000):
 # finish the code below
 simulated_rolls = pd.DataFrame(random.choices([1,2,3,4,5,6], k=60), columns=['roll']) # create a 1-column table of randomly simulated rolls
 # sixes is an array of True or False. True if roll is 6, False otherwise.
 sixes = ...
 # calculate the statistic for this simulated data using sixes
 sim_t1 = ...
 statistic1.loc[len(statistic1)] = [sim_t1] # add simulated result to end of table
statistic1`

In []: `grader.check("Q3.2.2")`

Problem 3.2.3 (2 points).

Plot the empirical distribution of your statistics simulated under the null. Plot a vertical line for the value of the measured statistic for your actual games. Label your plot using the parameter `label=`.

In []: `# 3. Plot empirical distribution of t1 (under the null)
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1)
write your code here
...
ax.legend()`

Problem 3.2.4 (1 point).

Looking at the shape of your distribution, what type of distribution is your simulation data, approximately? We're referring to the distribution of the statistics simulated under the null.

Write your answer here

Problem 3.2.5 (2 points).

Write code to compute the p-value from the `statistic1` table.

In []: `p_value1 = ...
p_value1`

In []: `grader.check("Q3.2.5")`

Problem 3.2.6 (1 point).

Does your simulation provide evidence against the null? (yes or no) If so, is the evidence statistically significant? Is it extremely statistically significant? Explain the convention you are using to answer these questions.

Write your answers here

Problem 3.3 (4 points) Hypothesis testing revisited.

Instead of investigating if there is a less than fair chance of rolling a 6 like you just did, now evaluate if there is an **unfair chance** of rolling a 6 (either more likely or less likely than fair).

1. State the null hypothesis for this situation.
2. State the alternative hypothesis.
3. Which of the following statistics would you be appropriate for evaluating these hypotheses?
 - a) number of 6's rolled in a row
 - b) $\text{abs}(\text{proportion of 6's rolled} - 1/6)$
 - c) $\text{abs}(\text{proportion of 6's rolled} + 1/6)$
 - d) $\text{abs}(\text{total number of rolls})$
4. What is the value of this statistic for the 20 games you played? Assign it to `measured_statistic2` in the provided cell.

Write your answers to 1-4 here

```
In [ ]: # Assign your measured statistic here
measured_statistic2 = ...
```

```
In [ ]: grader.check("Q3.3")
```

Problem 3.4 (6 points) Simulating under the null revisited

Problem 3.4.1 (2 points).

Now, write code to simulate 1000 times this new test statistic for 60 die rolls, and store the results in a table named `statistic2`. You can use the same `sample_die()` function you wrote before.

```
In [ ]: statistic2 = pd.DataFrame(columns=['test statistic'])

for simulation in range(1000):
    # finish the code below
    simulated_rolls = ...
    sixes = ...
    sim_t2 = ...
    statistic2.loc[len(statistic2)] = [sim_t2] # add simulated result to end of table
statistic2
```

```
In [ ]: grader.check("Q3.4.1.")
```

Problem 3.4.2 (2 points).

Plot the empirical distribution of your statistics simulated under the null. Plot a vertical line for the value of the measured statistic for your actual games. Label your plot using the parameter `label=`.

```
In [ ]: fig, ax = plt.subplots(1)
# write your code here
...
ax.legend()
```

Problem 3.4.3 (1 points).

Write code to compute the p-value from the `statistic2` table.

```
In [ ]: p_value2 = ...
p_value2
```

```
In [ ]: grader.check("Q3.4.3")
```


Problem 3.4.4 (1 points).

Does your simulation provide evidence against the null? (yes or no) If so, is the evidence statistically significant? Is it extremely statistically significant?

Write your answers here

Submission to Gradescope

Step 1 – Run All Cells and Save

- My favorite thing to do in Jupyter Notebook is to click the  button on the toolbar. This refreshes the memory, then runs all cells in order.
- Then, click **File** → **Save Notebook** to make sure your notebook file (`.ipynb`) is up to date. Or run `command` (or `control`) + `s`.

Step 2 – Export to PDF

1. Choose **File** → **Print**.
2. In your browser's print dialog, change the **Destination** to "Save as PDF."
3. Click **Save** and name the file clearly (e.g., `lab1.pdf`).

Step 3 - Run the Last Cell

1. Run the last cell (below)
2. Download the zip file to your computer.

Step 4 – Submit to Gradescope

- Log into **Gradescope** and open the assignment. You can get there from Canvas assignment as well.
- Upload **both files** by highlighting both items and draggin and dropping them to the submission box:
 1. Your **.zip file** (generated by otter).
 2. Your **.pdf file** (exported from the notebook).
- Make sure your submission is **complete and readable** before clicking submit.

Step 5 – Confirm Your Submission

- After submitting, check Gradescope's preview to ensure your files display properly.
- If anything is missing or unreadable, re-export and resubmit before the deadline.

Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

Submit PDF to Gradescope Homework5

```
In [ ]: # Save your notebook first, then run this cell to export your submission.
grader.export(pdf=False, run_tests=True)
```