Hotel Booking Dataset

This dataset contains booking data for two types of hotels, includes data such as when the booking was made, length of stay, the number of adults, children, and/or babies, and the number of available parking spaces, and more.

dataset

We didn't do any transformations to the data so the dataset remains the same. Explanations for each column is provided in the link for the dataset.

Binary Variables:

1. is_repeated_guest - if guest rebooked the hotel
2. hotel - there are two types of hotel (Resort Hotel or City Hotel)

Numerical variables:

1. lead_time - delta time between booking and arrival to hotel
2. adr - #lodging transactions / #nights stayed

We sample randomly (using a seed 42) 5000 rows from the dataset

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from scipy.stats import t
from scipy.stats import chi2

hotel_data = pd.read_csv('hotel_bookings.csv').sample(n=5000,
random_state=42)
categories = ['lead_time', 'adr']
n = len(hotel_data)
hotel_data.head(5)

{"type":"dataframe","variable_name":"hotel_data"}
```

Q1: Estimators

Answers for section a, b

```
avg_estimators, std_estimators = {}, {}
z_value = norm.ppf(0.975)
n = len(hotel_data)

for cat in categories:
```

```
    avg_estimators[cat] = hotel_data[cat].mean()
    std_estimators[cat] = hotel_data[cat].std()

    print(f"Mean and CI for category {cat}:\n")
    print(f"average is: {avg_estimators[cat]:.3f}")

    ci_lower = avg_estimators[cat] - z_value * std_estimators[cat] /
math.sqrt(n)
    ci_upper = avg_estimators[cat] + z_value * std_estimators[cat] /
math.sqrt(n)
    print(f"confidence interval is: [{ci_lower:.3f}, {ci_upper:.3f}]\
n")
```

```
Mean and CI for category lead_time:

average is: 105.707
confidence interval is: [102.698, 108.717]

Mean and CI for category adr:

average is: 100.802
confidence interval is: [99.484, 102.121]
```

Q1 section B

The estimator distributes normally as we showed in class that the MLE estimator is the mean for each category, thus makes it asymptoctially normal.

Therefore showing for expectation:

$$E(\acute{X}) = E(X) = \mu_x$$

For variance:

$$Var(\acute{X}) = \frac{1}{n^2}\sum_{i=1}^{n} \mathrm{Var}[X_i] = \frac{Var(X)}{n} = \frac{\sigma_x^2}{n}$$

Therefore, using the central limit theorem and that MLE's are asymptotic normal we meet the assumptions requirements.

Q1 section C:

The adr confidence level don't **overlap** with the lead_time confidence level. The two variables fundamentally represent different ideas and as such whether they overlap or not **isn't meaningful** in any way.

Q2:

Section A:

Looking the adr variable and the lead_time variable.

Hypthosis:

The null Hypothesis: lead_avg = adr_avg

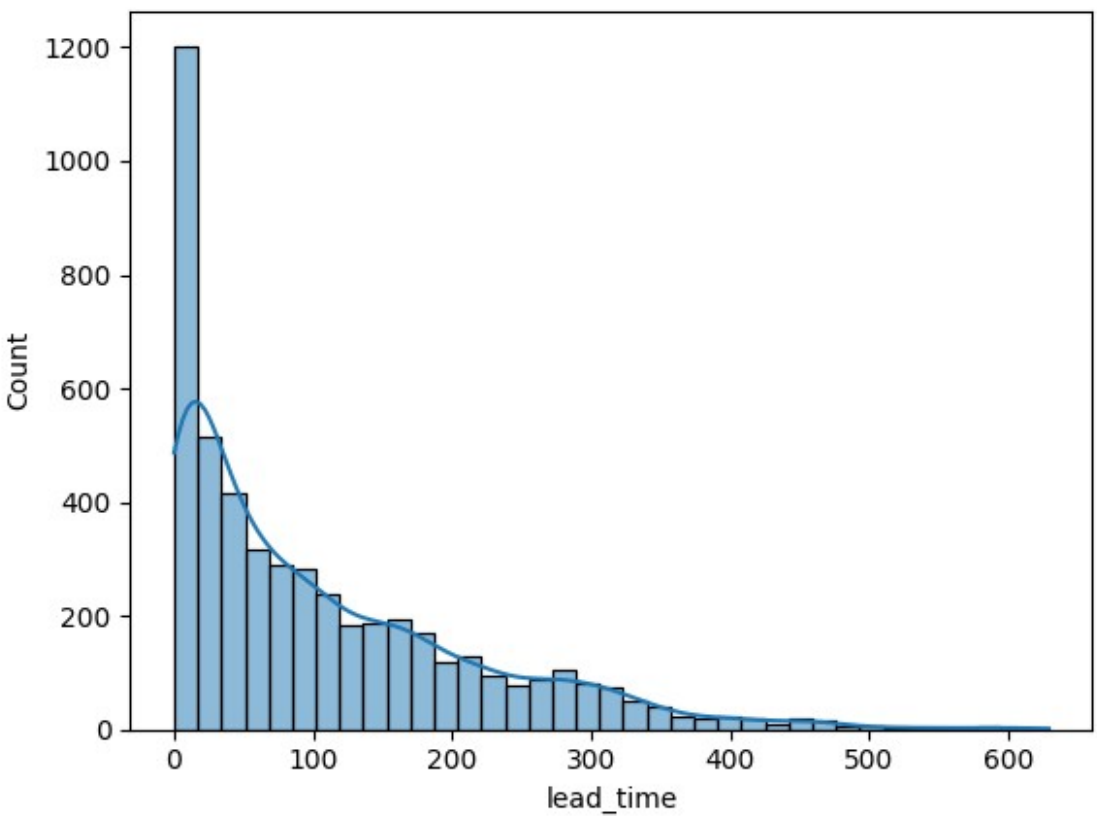
Alternative: lead_avg ≠ adr_avg

Q2 Section B:

The t-test assumptions are not all fufilled.

For example, for the lead_time variable as we saw in the graph from the previous project 1 homework. Reason being- looking at the histogram we see that the variable has a heavy-tailed distribution.

```
sns.histplot(hotel_data['lead_time'], kde=True)  # Add kde=True for a
Kernel Density Estimate
plt.show()
```



Q2 Section C:

```
n1, n2 = len(hotel_data['lead_time']), len(hotel_data['adr'])
avg_diff = avg_estimators['lead_time'] - avg_estimators['adr']
sp = (
        ((n1 - 1) * (std_estimators['lead_time'] ** 2)) +
        ((n2 - 1) * (std_estimators['adr'] ** 2))
    ) / (n1 + n2 - 2)
t_statistic = avg_diff / math.sqrt(sp * (1/n1 + 1/n2))

alpha = 0.05     # Significance level
df = 10000 - 1        # Degrees of freedom (replace with your value)

critical_value = t.ppf(1 - alpha/2, df)

p_value = 2 * (1 - t.cdf(abs(t_statistic), df))
if abs(t_statistic) > critical_value:
    print(f"Reject the null hypothesis (t = {t_statistic:.3f},
critical value = {critical_value:.3f})")
else:
    print(f"Fail to reject the null hypothesis (t = {t_statistic:.3f},
critical value = {critical_value:.3f})")

if (p_value < 0.001):
    print(f"p-value < 0.001")
else:
  print(f"p-value: {p_value:.3f}")


Reject the null hypothesis (t = 2.926, critical value = 1.960)
p-value: 0.003
```

Q2 Section D:

```
se = (std_estimators['lead_time'] ** 2 / len(hotel_data['lead_time']))
+ (std_estimators['adr'] ** 2 / len(hotel_data['adr']))
W_n = avg_diff / math.sqrt(se)

if abs(W_n) > z_value:
    print(f"Reject the null hypothesis (W_n = {W_n:.3f}, critical
value = {z_value:.3f})")
else:
    print("Failed to reject the null hypothesis")

p_value = 2 * (1 - norm.cdf(abs(W_n)))
if (p_value < 0.001):
    print(f"p-value < 0.001")
else:
  print(f"p-value: {p_value:.3f}")
```

```
Reject the null hypothesis (W_n = 2.926, critical value = 1.960)
p-value: 0.003
```

Q2 Part E:

for both tests we get p_value = 0.003

the tests give the same result. The test statistic comes out the same. This true as for large values of n we expect the Wald and t-test to give the same result.

Both tests reject the null hypothesis.

The results from the previous calculations support each other.

Q2 Part F:

Hypothesis:

The Null Hypothesis: lead_avg - adr_avg = 0

Alternative: lead_avg - adr_avg ≠ 0

Lamda test (given in the project instructions) involves a lot of multiplications which isn't computational feasible to calculate since the numbers can easily explode to infinity or zero. Therefore, we reduced the multiplications usong Log to sums to make the calculation feasible.

After doing some reductions we get the following expression which is equivalent to the lambda test:

$$2 \log \left(\frac{\mathcal{L}(\hat{\mu}_1, {\mu}_2)}{\mathcal{L}(\hat{\mu})}\right) = \\ n \log \left(\frac{\hat{\sigma}^2}{S_p^2}\right) + \sum_{i=1}^n \frac{(X_i - \hat{\mu})^2}{\hat{\sigma}^2} - \sum_{i=1}^{n_1} \frac{(X_{i1} - \hat{\mu}_1)^2}{S_p^2} - \sum_{i=1}^{n_2} \frac{(X_{i2} - \hat{\mu}_2)^2}{S_p^2}$$

```
X1 = np.array(hotel_data['lead_time'])
X2 = np.array(hotel_data['adr'])

X1_mean = avg_estimators['lead_time']
X2_mean = avg_estimators['adr']

X = np.concatenate([X1, X2])
X_mean = np.mean(X)
X_var = np.var(np.concatenate([X1, X2]))
s_p = sp #from Q2C

n = len(X1)
m = len(X2)

# Likelihood ratio calculation
Lamda = (n + m) * math.log(X_var ** 2 /(s_p ** 2))
```

```
Lamda += sum(((X[i] - X_mean) ** 2) for i in range(n+m)) / (X_var **
2)
Lamda -= sum(((X1[i] - X1_mean) ** 2) for i in range(n)) / (s_p ** 2)
Lamda -= sum(((X2[i] - X2_mean) ** 2) for i in range(m)) / (s_p ** 2)

# Calculate the critical value, Chi-squared test
critical_value = chi2.ppf(0.95, 1)

# Output results
print(f"Likelihood ratio statistic (lambda): {Lamda:.5}")
print(f"Critical value at 95%: {critical_value:.4}\n")

if Lamda > critical_value:
    print("The Null Hypothesis is rejected")
else:
    print("Failed to reject the Null Hypothesis")
```

```
Likelihood ratio statistic (lambda): 13.119
Critical value at 95%: 3.841

The Null Hypothesis is rejected
```

Q3:

```
def analyze_hotel_data(hotel_data, categories, sample_sizes,
n_repeats, confidence_level=0.95):
    # Initialize results dictionary with consistent keys (size, cat)
    results = {
        'ci_lengths': { (size, cat): [] for cat in categories for size
in sample_sizes },
        'p_values': { (size, cat): [] for cat in categories for size
in sample_sizes },
        'ci_intervals': {}
    }

    # Initialize Wald test results with consistent keys (size, cat)
    wald_test_results = { (size, cat): [] for cat in categories for
size in sample_sizes }

    z_value = norm.ppf((1 + confidence_level) / 2)
    pop_means = {cat: hotel_data[cat].mean() for cat in categories}

    for size in sample_sizes:
        for iteration in range(n_repeats):
            sample = hotel_data.sample(n=size)

            for cat in categories:
                sample_mean = sample[cat].mean()
```

```
                sample_std = sample[cat].std()
                se = sample_std / np.sqrt(size)

                # Confidence Interval
                ci_lower = sample_mean - z_value * se
                ci_upper = sample_mean + z_value * se
                ci_length = ci_upper - ci_lower
                results['ci_lengths'][(size, cat)].append(ci_length)

                # p-value calculation
                z_score = (sample_mean - pop_means[cat]) / se
                p_value = 2 * (1 - norm.cdf(abs(z_score)))
                results['p_values'][(size, cat)].append(p_value)

                # Wald test calculation
                if sample_std == 0:
                    wald_p_value = 1.0  # No variability in the sample
                    z_score_wald = 0
                else:
                    z_score_wald = (sample_mean - pop_means[cat]) / se
                    wald_p_value = 2 * (1 -
norm.cdf(abs(z_score_wald)))
                wald_test_results[(size, cat)].append((wald_p_value,
z_score_wald))

                if iteration == 0:
                    results['ci_intervals'][(size, cat)] =
f"[{ci_lower:.3f}, {ci_upper:.3f}]"

    def sort_dict(results, k):
        return dict(sorted(results[k].items(), key=lambda x: (x[0][0],
x[0][1])))

    # Create DataFrames from sorted dictionaries
    ci_intervals = pd.DataFrame.from_dict(sort_dict(results,
"ci_intervals"), orient='index')
    ci_lengths = pd.DataFrame.from_dict(sort_dict(results,
"ci_lengths"), orient='index')
    p_values = pd.DataFrame.from_dict(sort_dict(results, "p_values"),
orient='index')

    # Align `p_values` structure with `ci_lengths`
    p_values.index = ci_lengths.index
    p_values.columns = ci_lengths.columns

    # Wald test summary calculation with consistent keys
    wald_test_summary = pd.DataFrame.from_dict({
        cat: {
            size: (
                np.mean([result[0] for result in
```

```
wald_test_results[(size, cat)]]),  # Mean p-value
                np.mean([result[1] for result in
wald_test_results[(size, cat)]])   # Mean z-score
            )
            for size in sample_sizes
        }
        for cat in categories
    })

    summaries = {
        'ci_length_summary': ci_lengths,
        'p_value_summary': p_values,
        'ci_intervals': ci_intervals,
        'wald_test_summary': wald_test_summary  # Add the Wald test
summary here
    }

    return summaries
```

Q3 Section A:

```
# Parameters
categories = ['lead_time', 'adr']
sample_sizes = [30, 50, 100, 500]
n_repeats = 100
z_value = norm.ppf(0.975)  # For 95% confidence
summaries = analyze_hotel_data(hotel_data, categories, sample_sizes,
n_repeats)
```

Q3 Section B:

```
print("\nConfidence Intervals:")
print(summaries['ci_intervals'])
```

```
Confidence Intervals:
                                        0
(30, adr)           [76.190, 104.080]
(30, lead_time)     [57.359, 121.775]
(50, adr)           [90.489, 111.877]
(50, lead_time)     [90.338, 136.222]
(100, adr)          [98.285, 117.736]
(100, lead_time)    [74.375, 114.465]
(500, adr)          [95.019, 102.809]
(500, lead_time)   [103.365, 122.355]
```

Comparing to our results in Q1, we can see that the estimator (the mean) is in the CI intervals here and in Q1.

Looking at the CI interval as we sample more examples we get a smaller interval that in 95% confidence. Compared to Q1 where we sample 5000 examples, we get that it's a subset of all the smaller sample intervals which makes sense because we expect to be more accurate with more samples.

Q3 Section C:

```
print("\nWald Test (p-value, W_n):")
print(summaries['wald_test_summary'])
print("NOT an interval/range of numbers. The left number represents
the p-value and the right number the W_n statistic")
```

```
Wald Test (p-value, W_n):
                                        lead_time  \
30    (0.49732475952786137, -0.2308351224833729)
50    (0.5097632223595542, -0.07183936160982979)
100  (0.4728275590510841, -0.050085042458295176)
500    (0.515044903436805, 0.006105387802299389)

                                            adr
30    (0.4721932914501099, 0.019699601461653185)
50   (0.5035556648906969, 0.0028084632894668166)
100     (0.511749754080797, 0.23227452460551692)
500  (0.49629280557169564, 0.028927450533530588)
NOT an interval/range of numbers. The left number represents the p-
value and the right number the W_n statistic
```

Q3 (C) Results:

For the Wald test in Q3 (C), the p-values are generally distributed around 0.5 (e.g., mean p-values: lead_time = 0.508, adr = 0.427 for sample size 30). This suggests the null hypothesis (equality of population means) was not consistently rejected across the different sample sizes.
Q2 (C) Results:

For the Wald test in Q2 (C), the p-value (0.00343) indicates strong evidence against the null hypothesis, leading to its rejection. The test statistic W_n(2.926) exceeds the critical value of z-score (1.960), confirming the rejection of the null hypothesis.

Hypothesis and Variables:

In Q3 (C), the Wald test evaluates whether the sample mean for a single variable (lead_time or adr) equals the population mean for that variable. This is a univariate test. In Q2 (C), the Wald test compares the means of two variables (lead_time and adr). This is a test for the equality of two means (a two-sample test).

Q3 (C) involves multiple repetitions (100) of sampling, allowing a distribution of p-values and confidence interval lengths to assess sampling variability. Q2 (C) focuses on a single hypothesis test (equality of two means) using the overall data (n=5000 for each variable).

The p-values in Q3 (C) are larger, reflecting less consistent evidence against the null hypothesis across samples. The small p-value in Q2 (C) indicates strong evidence against the null hypothesis, suggesting a significant difference between the means of lead_time and adr.

The results are consistent with their respective contexts. The Wald test in Q2 (C) is more sensitive due to the larger sample size and different hypothesis. The p-value of Q3 (C) is not directly comparable to Q2 (C) because the tests serve different purposes (single-variable mean vs. mean comparison across two variables).

Q3 Section D:

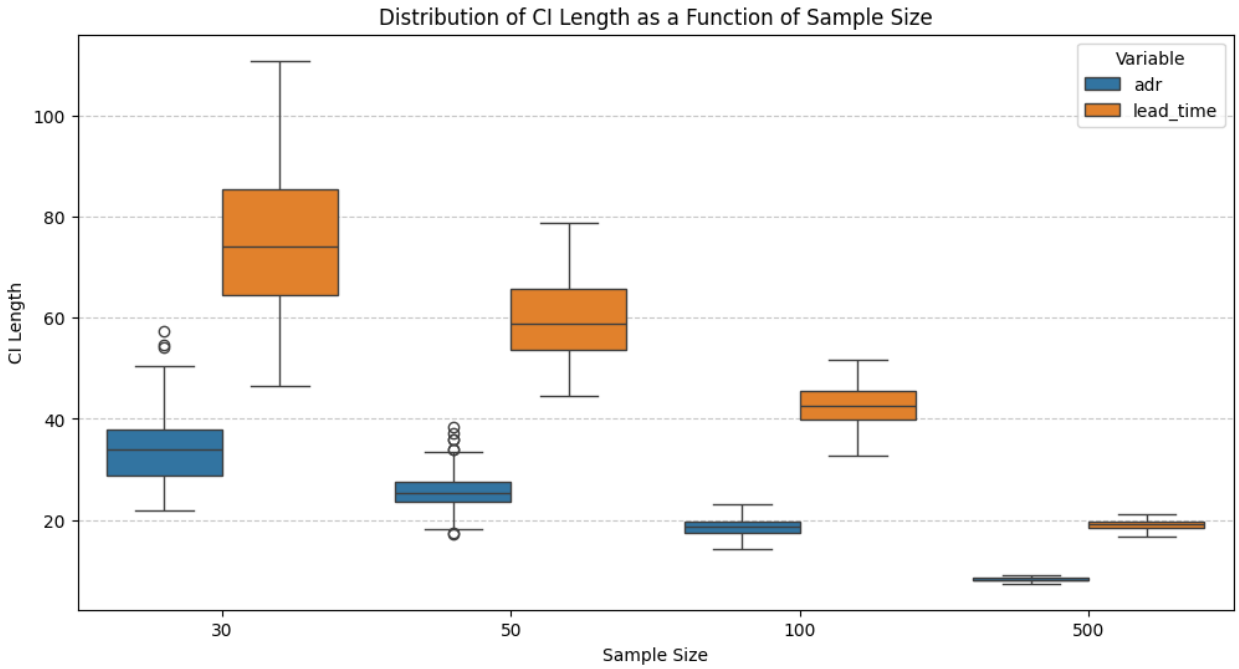
```
def plot_summary(df, name):
    df = df.reset_index()
    df[['sample_size', 'variable']] =
pd.DataFrame(df['index'].tolist(), index=df.index)
    df = df.drop(columns=['index'])

    df = df.melt(
        id_vars=['sample_size', 'variable'],
        var_name='data_point',
        value_name=name
    )

    plt.figure(figsize=(12, 6))
    sns.boxplot(data=df, x='sample_size', y=name, hue='variable')
    plt.title(f'Distribution of {name} as a Function of Sample Size')
    plt.xlabel('Sample Size')
    plt.ylabel(name)
    plt.legend(title='Variable')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()
```

CI lengths:

```
ci_lengths = summaries['ci_length_summary']
ci_lengths.head(6)
```

{"type":"dataframe","variable_name":"ci_lengths"}

```
plot_summary(ci_lengths, 'CI Length')
```

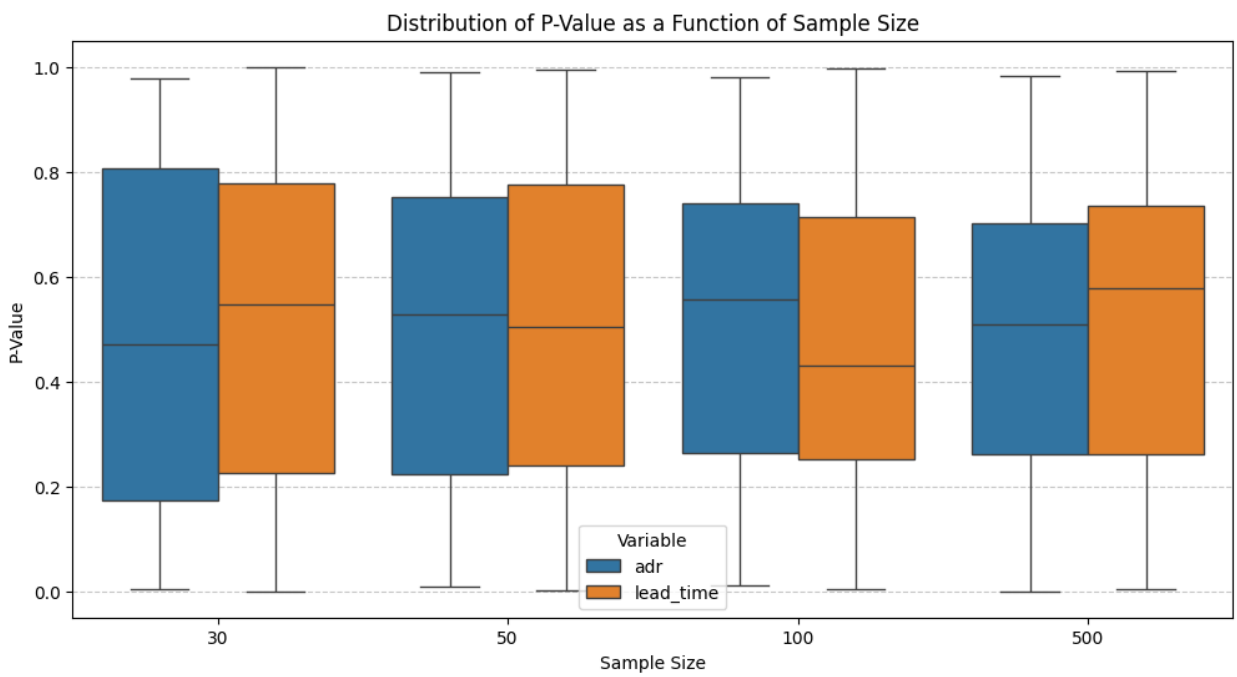


P-Value:

```
p_values = summaries['p_value_summary']
p_values.head(6)
```

{"type":"dataframe","variable_name":"p_values"}

```
plot_summary(p_values, 'P-Value')
```



The percentage of time that the mean/average found in **Q1** is in the CI intervals calulated is 100% i.e. the average/mean of the variable was always in the CI interval.