Statistics 2 - Project - Assignment #4

Jeremy (931215248) and Uri (300691367)

In [31]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
from scipy import stats
import random

plt.style.use('ggplot')
```

texte en italique## Load Data

load the data from a file and create the df used in previous project exercises.

Recall that the data comprises recorded observations of booking information of a hotel.

The resulted df will contain 20k rows. This will be considered as the **"entire population"** from which we sample.

In [2]:
```
# load data
# file downloaded from
# https://www.kaggle.com/datasets/khairullahhamsafar/hotels-booking-data-cleaned-version
# and dropped as is in the notebook

df = pd.read_csv("hotel_booking_data_cleaned.csv")


# Data preprocessing

# 1. sum of all guests in reservation
df['total_guests'] = df['adults'] + df['children'] + df['babies']

# 2. sum of all nights in reservation
df['number_of_nights'] = df['stays_in_week_nights'] + df['stays_in_weekend_nights']

# 3. binary indicator whether a weekend night included in reservation
df['weekend_included'] = df['stays_in_weekend_nights'].map(lambda x: 1 if x > 0 else 0)

# 4. First, concatenate the relevant columns and convert the type to obtain a datetime value.
df['arrival_date'] = df['arrival_date_month'].astype(str) + ' ' + df['arrival_date_day_of_month'].astype(str) + ', ' + df['arrival_date_year'].astype(str)
df['arrival_date'] = df['arrival_date'].apply(lambda x: datetime.strptime(x, "%B %d, %Y"))
# Next, identify day_0 and calculate the new column values based on it.
day_0 = df['arrival_date'].sort_values().values[0]
df['arrival_day'] = df['arrival_date'].map(lambda x: (x - day_0) / np.timedelta64(1, 'D'))

# 5. extract day of the week from the date of arrival
df['arrival_day_of_week'] = df['arrival_date'].map(lambda x: x.strftime('%a'))

# remove rows with 'total_guests' = 0 or 'number_of_nights' = 0 or 'adr' = 0
df = df.loc[(df['total_guests'] > 0) & (df['number_of_nights'] > 0) & (df['adr'] > 0)]

# remove extreme outlier (removing 1 row)
df = df.loc[df['adr'] < 5000] #.reset_index(drop=True)

# 6. create another new column
df['adr_per_guest'] = df['adr'] / df['total_guests']


# narrowing down data

# select only records from top 1 country, and top 1 hotel
df = df.loc[(df['hotel'] == 'City Hotel') & (df['country'] == 'PRT')]

# selecting features
selected_columns = [
    'is_canceled', 'lead_time', 'arrival_date_year',
    'arrival_date_month', 'arrival_day_of_week', 'arrival_day',
    'number_of_nights', 'weekend_included', 'is_repeated_guest',
    'deposit_type', 'adr', 'total_guests', 'adr_per_guest'
    ]
df = df[selected_columns].copy()


# sample 20000 random data points
n = 20000
df = df.sample(n, random_state=27).reset_index(drop=True)
```

Part 2 - TESTS

QUESTION 1

Our question in Project 2 was : Do guests who return (i.e., with is_repeated_guest equals 1) tend to pay a different average nightly rate (i.e., different 'adr') ?

Now, our question is : Is the distribution of the average nightly rate (i.e., 'adr') among non returning guests (i.e., with is_repeated_guest equals 0) stochasticly higher than the distribution of the average nightly rate (i.e., 'adr') among returning guests (i.e., with is_repeated_guest equals 1) ?

QUESTION 2

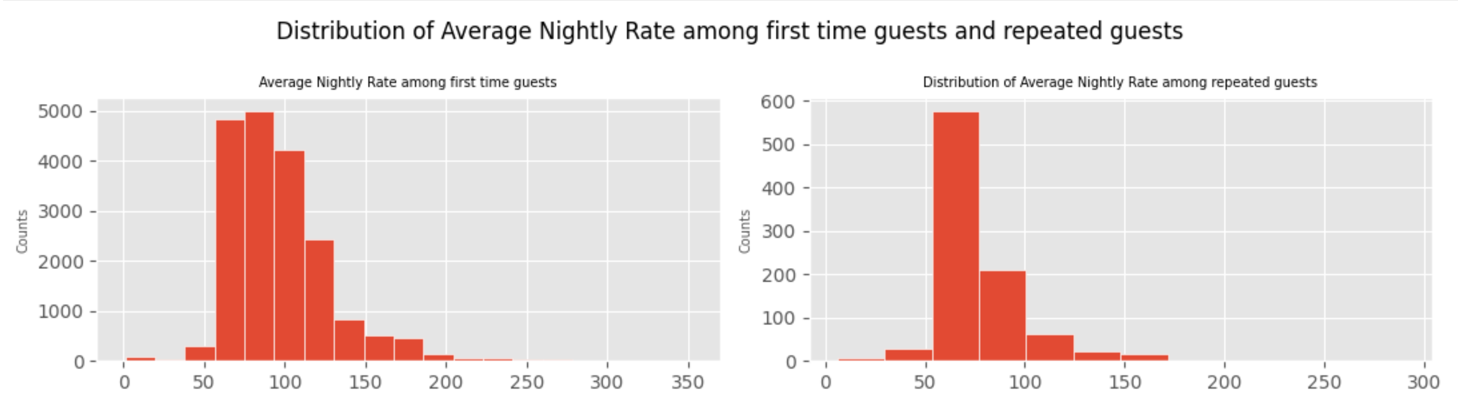
In [4]:
```
# The 'is_repeated_guest' frequency is unbalanced among all the data. Because of that,
# we balance it in the sample data

df_1 = df[df['is_repeated_guest'] == 1]
df_0 = df[df['is_repeated_guest'] == 0]

df_rep_sample = df_1.sample(n=100, random_state=10)
df_first_sample = df_0.sample(n=100, random_state=10)

df_all_sample = pd.concat([df_rep_sample, df_first_sample])

df_rep_sample = df_rep_sample.sample(frac=1, random_state=1).reset_index(drop=True)
df_first_sample = df_first_sample.sample(frac=1, random_state=1).reset_index(drop=True)
```

QUESTION 3

In [6]:
```
rep_adr_sample = df_rep_sample['adr']
first_adr_sample = df_first_sample['adr']

mu1_estimator = np.mean(first_adr_sample)
mu2_estimator = np.mean(rep_adr_sample)

delta_mu1_mu2 = mu1_estimator - mu2_estimator

print(f"The mu1 estimator is : {round(mu1_estimator,2)}")
print(f"The mu2 estimator is : {round(mu2_estimator,2)}")
print(f"The delta estimator is : {round(delta_mu1_mu2,2)}")
```

```
The mu1 estimator is : 94.72
The mu2 estimator is : 77.25
The delta estimator is : 17.47
```

In [7]:
```
repeated_guest_df = df[df['is_repeated_guest'] == 1 ]
first_time_guest_df = df[df['is_repeated_guest'] == 0 ]

fig, ax = plt.subplots(1, 2, figsize=(11,3))
fig.suptitle('Distribution of Average Nightly Rate among first time guests and repeated guests')
ax[0].hist(first_time_guest_df['adr'], edgecolor = "white", bins=19)
ax[0].set_ylabel("Counts", fontsize=7)
ax[0].set_title('Average Nightly Rate among first time guests', fontsize=7)
ax[1].hist(repeated_guest_df['adr'], edgecolor = "white", bins=12)
ax[1].set_ylabel("Counts", fontsize=7)
ax[1].set_title('Distribution of Average Nightly Rate among repeated guests', fontsize=7)
plt.tight_layout()
plt.show()
```



The assumption that the distributions of 'adr' among first time guests and repeated guests are normal is reasonable in accordance to the plots. Now that we assumed this, we can say that mu1 and mu2 are MLE because the esperancy's MLE of a normal distribution is the sample's mean. As we already know, the difference of two normal distribution is also a normal distribution. So, the estimator of the difference of the two distribution is also a Maximum Likelihood Estimator beacuse we defined it as the mean of the differences.

In [36]:
```
sample_variance_rep = rep_adr_sample.var(ddof=0)
sample_variance_first = first_adr_sample.var(ddof=0)
sample_variance_delta = sample_variance_rep + sample_variance_first
delta_se = sample_variance_delta**(1/2)
norm_quantile = stats.norm.ppf(0.975)


CI = [round(delta_mu1_mu2 - norm_quantile*delta_se,4), round(delta_mu1_mu2 + norm_quantile*delta_se,4)]
print(f"The Confidence Interval of the difference of the 'adr' value of both distributions among the sample is: {CI}")
```

```
The Confidence Interval of the difference of the 'adr' value of both distributions among the sample is: [-56.1468, 91.0876]
```

In [37]:
```
mu1_all = np.mean(first_time_guest_df['adr'])
mu2_all = np.mean(repeated_guest_df['adr'])

delta_mu1_mu2_all = mu1_all - mu2_all

print(f"The delta estimator among all the data is : {round(delta_mu1_mu2_all,4)}")
```

```
The delta estimator among all the data is : 19.1101
```

The delta estimator among all the data is included in the Confidence Interval from above.

In [30]:
```
def F_test(first_sampled_df, rep_sampled_df):
  n_first = first_sampled_df.count()
  n_rep = rep_sampled_df.count()

  F_statistic = first_sampled_df.var()/rep_sampled_df.var()
  p_value = 1-stats.f.cdf(F_statistic, n_first-1, n_rep-1)

  F_test_quantile = stats.f.ppf(0.95, n_first-1, n_rep-1)

  print(F_statistic)
  print(F_test_quantile)

  if (F_statistic > F_test_quantile):
    return "Reject (variances are not equal)"
  else:
    return "Do not reject (variances are equal)"

# Wald Test
statistic = delta_mu1_mu2/delta_se
quantile = stats.norm.ppf(0.975)
p_value = 2*stats.norm.cdf(-abs(statistic))
print(f"Variance equivalence by F test:")
print(F_test(first_adr_sample, rep_adr_sample) + "\n")
print("Wald Test:")
print(f"Reject H0? : {abs(statistic) > quantile}")
print("statistic: ", statistic)
print("p_value: ", p_value)
print("quantile: ", quantile)
```

```
Variance equivalence by F test:
0.7443755011611838
1.3940612573481483
Do not reject (variances are equal)

Wald Test:
Reject H0? : False
statistic:  0.46512700069359697
p_value:  0.6418405112089278
quantile:  1.959963984540054
```

In [34]:
```
# PERMUTATION TEST

num_permutations = 400

counter = 0

all_adr = np.array(df_all_sample["adr"])

# Perform the permutation test
for _ in range(num_permutations):
    random.shuffle(all_adr)

    group1 = all_adr[:len(first_adr_sample)]
    group2 = all_adr[len(rep_adr_sample):]

    permuted_difference = np.mean(group1) - np.mean(group2)

    if permuted_difference >= delta_mu1_mu2:
        counter += 1

# Calculate the p-value
p_value = counter / num_permutations

# Print the results
print(f"Observed Difference: {delta_mu1_mu2}")
print(f"P-value: {p_value}")
print(f"Reject H0? {p_value < 0.05}")
```

```
Observed Difference: 17.470400000000026
P-value: 0.0
Reject H0? True
```

QUESTION 4

In [35]:
```
median_first_est = np.median(first_adr_sample)
median_rep_est = np.median(rep_adr_sample)
delta_median = median_first_est - median_rep_est

print(f"estimator to the first time guests's adr median is: {median_first_est}")
print(f"estimator to the repeated guests's adr median is: {median_rep_est}")
print(f"estimator to the difference between both is: {delta_median}")
```

```
estimator to the first time guests's adr median is: 97.5
estimator to the repeated guests's adr median is: 67.0
estimator to the difference between both is: 30.5
```

We assumed that both distributions are normal. Normal distributions are symetric and as we already know, the MLE estimator of the median is the mean in symetric distributions. From our discussion in QUESTION 3, our delta estimator is a MLE.

We can infer that the Confidence Interval of the delta median is the same as the Confidence Interval of the expectation. We already computed it in QUESTION 3 : The Confidence Interval is: [-56.1468, 91.0876]

In [38]:
```
median1_all = np.median(first_time_guest_df['adr'])
median2_all = np.median(repeated_guest_df['adr'])

delta_median1_median2_all = median1_all - median2_all

print(f"The delta estimator among all the data is : {round(delta_median1_median2_all,4)}")
```

```
The delta estimator among all the data is : 23.0
```

The delta estimator among all the data is included in the Confidence Interval from above.

In [39]:
```
num_permutations = 400

counter = 0

all_adr = np.array(df_all_sample["adr"])

# Perform the permutation test
for _ in range(num_permutations):
    random.shuffle(all_adr)

    group1 = all_adr[:len(first_adr_sample)]
    group2 = all_adr[len(rep_adr_sample):]

    permuted_difference = np.median(group1) - np.median(group2)

    if permuted_difference >= delta_median:
        counter += 1

# Calculate the p-value
p_value = counter / num_permutations

# Print the results
print(f"Observed Difference: {delta_median}")
print(f"P-value: {p_value}")
print(f"Reject H0? {p_value < 0.05}")
```

```
Observed Difference: 30.5
P-value: 0.0
Reject H0? True
```

Because our estimator of the median is a MLE, he is asymptotic normal. So, we can use Wald Test.

We will check all the needed conditions to proceed to a T test :

- According to F test in QUESTION 3, the variances of both categories are equal.
- The adr value of both categories are independant.
- We already assumed that both categories samples follow a normal distribution.

So, we can use the T test.

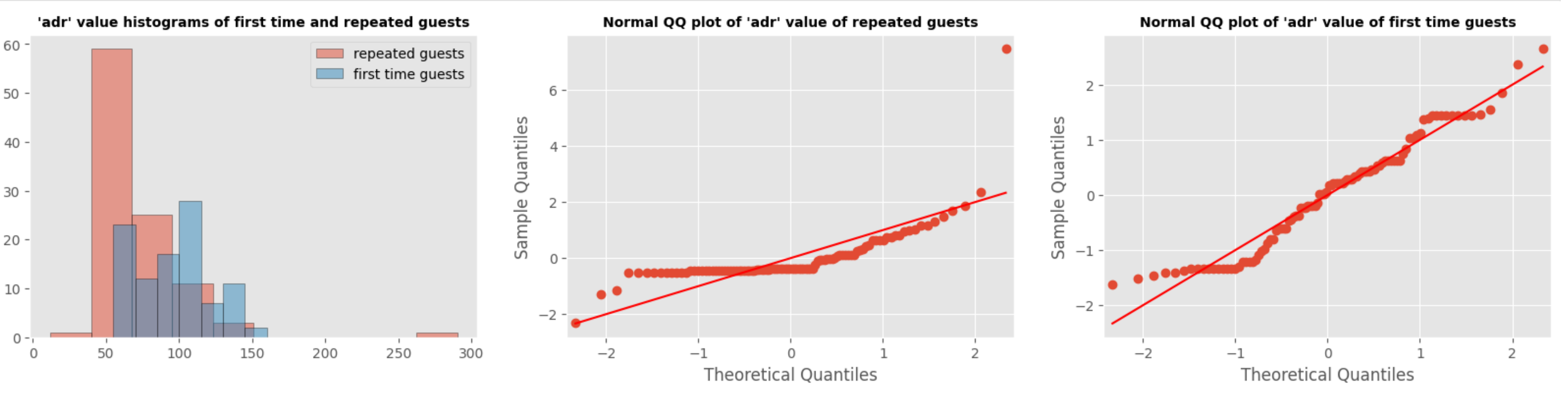
QUESTION 5

In [43]:
```
# Ranking the sample
df_all_sample['adr_rank'] = df_all_sample['adr'].rank()

# Define the combined group
combined_group = df_all_sample

# Define the control group : the first time guests
control_group = combined_group[combined_group['is_repeated_guest'] == 0]

# Define the treatment group : the repeated guests
treatment_group = combined_group[combined_group['is_repeated_guest'] == 1]

# Define your observed sum of ranks in the treatment group
S1 = np.sum(treatment_group['adr_rank'].values)

num_permutations = 400

counter = 0

# Perform the permutation test
for _ in range(num_permutations):
    # Shuffle rank values
    shuffled_ranks = np.random.permutation(combined_group['adr_rank'].values)
    combined_group['shuffled_rank'] = shuffled_ranks

    permuted_sum_of_ranks = np.sum(combined_group[combined_group['is_repeated_guest'] == 1]['shuffled_rank'].values)

    if permuted_sum_of_ranks >= S1:
        counter += 1

# Calculate the p-value
p_value = counter / num_permutations

# Print the results
print(f"Observed sum of ranks: {S1}")
print(f"P-value: {p_value}")
print(f"Reject H0 by Permutations test Resampling? {p_value < 0.05}")

# Using normal approximation
total = len(combined_group['adr'])
rep = len(treatment_group['adr'])
first = len(control_group['adr'])

expected = rep*(total+1)/2
var = first*rep*(total+1)/12
statistic = (S1-expected)/(var**(1/2))

print(f"Reject H0 by Normal approximation? {statistic >= stats.norm.ppf(0.95)}")
```

```
Observed sum of ranks: 8114.0
P-value: 1.0
Reject H0 by Permutations test Resampling? False
Reject H0 by Normal approximation? False
```

QUESTION 6

In [52]:
```
from statsmodels.graphics.gofplots import qqplot

# Create subplots with three axes for side-by-side plotting
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20, 4))

rep_adr_sample.hist(edgecolor="black", grid=False, alpha=0.5, label="repeated guests", ax=ax1, bins=10)
first_adr_sample.hist(edgecolor="black", grid=False, alpha=0.5, label="first time guests", ax=ax1, bins=7)
ax1.set_title("'adr' value histograms of first time and repeated guests", fontweight="bold", fontsize=10)
ax1.legend()

qqplot(rep_adr_sample, stats.norm, fit=True, line="s", ax=ax2)
ax2.set_title("Normal QQ plot of 'adr' value of repeated guests", fontsize=10, fontweight="bold")

qqplot(first_adr_sample, stats.norm, fit=True, line="s", ax=ax3)
ax3.set_title("Normal QQ plot of 'adr' value of first time guests", fontsize=10, fontweight="bold")

# Display the plots
plt.show()
```



We did not get a singular result from all the tests. But we can infer from the first histogram that the frequency on low 'adr' values among repeated guests is significantly higher than the frequency of first time guests on those values. Nevertheless, all tests did not agree on the results.