Midterm

**PSTAT 134/234 (Spring 2023)

Data description: Insurance Claims

The data given in the file *Insurance.csv* consists of the number of car insurance claims made by policyholders in the third quarter of 2020 and 2021.

Read Data into Python

Numpy and Pandas is used to read in the csv file into python.

In [1]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
Claims = pd.read_csv("Insurance.csv")
Claims.head()
```

Out[1]:

	claims	year
0	38	2020
1	35	2020
2	20	2020
3	156	2020
4	63	2020

Question 1a: Subset Data

- Filter the data to only include rows in which the `year` is 2020. **From now on, all questions must be answered based on this filtered data set**.
- Assume that the number of claims is a random variable X with unknown distribution $F(X)$. Suppose we want to estimate the probability that the number of claims exceeds 50 ($\theta = P(X > 50)$).

Based on the given data calculate $\hat{\theta}$ as an estimate of this probability:

In [2]:
```
# Fill-in ...
filtered_claims = Claims[Claims['year'] == 2020]

n = len(filtered_claims)
x = sum(filtered_claims['claims'] > 50)
theta_hat = x / n

print(theta_hat)
```

```
0.27419354838709675
```

Question 1b: Model-free resampling

Use non-parametric bootstrap to generate the epirical distribution of $\hat{\theta}$.

1. Write a function named `bootstrap_data_theta` that can take `data_in`, as input. Inside `bootstrap_data_theta` function, you will:

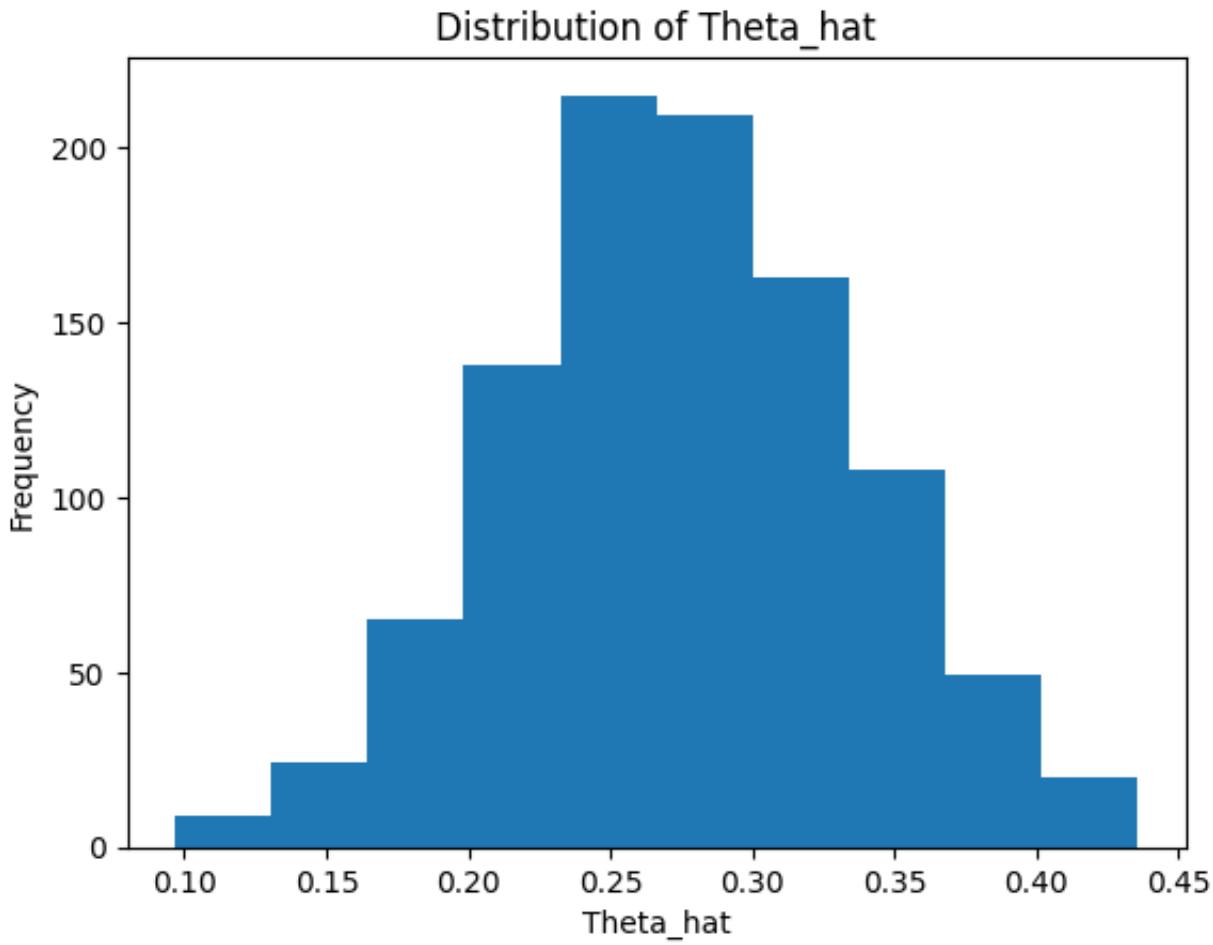
- Create a pseudo-data by using numpy.random.choice
- Calculate and return $\hat{\theta}$

2. Then, run the function `bootstrap_data_theta` function 1000 times, storing the resulting 1000 estimates of theta in a list.

3. Make a histogram showing the distribution of $\hat{\theta}$.

In [3]:

```
# Fill-in ...

def bootstrap_data_theta(data_in):

    from numpy.random import choice

    n = len(data_in)

    # randomly sample with replacement
    pseudo_data = np.random.choice(data_in, size=n, replace=True)
    # compute probabily theta
    bootstrap_theta = (sum(pseudo_data > 50))/n

    return(bootstrap_theta)

repeat_resampling = 1000

#list of all 1000 estimates
theta_list = []
for i in range(repeat_resampling):
    theta_estimate = bootstrap_data_theta(filtered_claims['claims'])
    theta_list.append(theta_estimate)

## Histogram
plt.hist(theta_list)
plt.title('Distribution of Theta_hat')
plt.xlabel('Theta_hat')
plt.ylabel('Frequency')
plt.show()
```



Type your answer here, replacing this text.

Question 1c: Bootstrap Confidence Interval

Construct a 95% confidence interval for θ based on the boostrap samples of $\hat{\theta}$. You can use any method (normal interval, pivotal interval or percentile interval).

Hint: In order to calculate quantiles for any variable you can refer to numpy.quantile

In [4]:
```
# Fill-in ...
upper= np.quantile(theta_list, 0.975)
lower= np.quantile(theta_list, 0.025)
print(lower, upper)
```

```
0.16129032258064516 0.3870967741935484
```

Question 1d: Bias

Calculate the Bootstrap bias estimate.

In [5]:
```
# Fill-in ...
bias = theta_hat - np.mean(theta_list)
print(bias)
```

```
-0.0004193548387096957
```

(PSTAT 234) Question 1e: Model-based bootstrap

Now suppose we want to estimate $\mu = E(X)$. Use parametric bootstrap to generate the empirical distribution of $\hat{\mu}$. For this strategy we assume a population distribution $f(x \mid \mu)$; pick an addequate distribution (Normal, poisson or Binomial) and follow the steps:

1. Estimate $\hat{\mu}$ based on the data.
2. Sample from $f(x \mid \hat{\mu})$
3. repeat steps 1 and 2, 1000 times.

- Make a histogram showing the distribution of $\hat{\mu}$.

In [6]:
```
# Fill in...

def bootstrap_data_mu(data_in):

    n = len(data_in)

    # randomly sample with replacement
    pseudo_data =  ...
    # compute mu hat
    bootstrap_mu = ...

    return(bootstrap_mu)

repeat_resampling = ...

# Histogram
...
```

Out[6]:
```
Ellipsis
```

(PSTAT 234) Question 1f: Expected value and Variance

Based on the bootstrap resampling calculate estimates for $E(\hat{\mu})$ and $Var(\hat{\mu})$. How these values compare to the theoretical ones? (*Theoretical*: $E(\hat{\mu}) = \mu$, $Var(\hat{\mu}) = \frac{Var(X)}{n}$)

In [7]:
```
## Fill in:

E_theta_hat= ...
Var_theta_hat = ...
print(E_theta_hat,Var_theta_hat)
```

```
Ellipsis Ellipsis
```

Type your answer here, replacing this text.

Intentionally Blank

Question 2a:

Suppose $X \sim F$, with $X \in \{-5, -4, -3, -2, -1, 0, 1, 2\}$.

By using np.random.choice, generate 100 `samples` of the random variable X, according to the given `probabilities`.

In [8]:
```
## Fill in...

probabilities = [0.1, 0.1, 0.1, 0.2, 0.2, 0.1, 0.1, 0.1]
X = [-5, -4, -3, -2, -1, 0, 1, 2]
samples = np.random.choice(X, size = 100, p = probabilities)
```

Question 2b:

- Explain line by line what the following snippet is doing:

In [9]:
```
def Ecdf(data):
    sorted_sample = np.sort(data)
    n = sorted_sample.size
    unique_values, counts = np.unique(sorted_sample, return_counts=True)
    cumulative_counts = np.cumsum(counts)

    return pd.DataFrame({'X': unique_values, 'F(X)': cumulative_counts / n})
```

ANSWER

First line defines a function Ecdf taking an input of data.

Second line sorts the input data in acending order

Third line gets the number of sorted data values. Specifically, this is the total number of points in the sample.

Fourth line counts and returns the number of unique occurances of each variable. The unique_variable stores each unique value and the counts stores the corresponding count

Fifth line stores the sum of each unique values, that is how many times did it occur.

Sixth line the function finally returns a data frame with the unique values in the x collumns and the cumulative count divided by the total number of data points as the as the f(x) collumn. The f(x) is the probability that the specific unique value will occur since we are getting the total count of it happening and dividing it by the total number of values.

Question 2c:

- By using function Ecdf and the sample that you generated, calculate $P(X = 0)$:

In [10]:
```
## Fill in..

prob_0 = Ecdf(samples).loc[Ecdf(samples)['X']==0]['F(X)'].values[0]
print(prob_0)
```

```
0.79
```

Submission Checklist

1. Save file to confirm all changes are on disk
2. Run *Kernel > Restart & Run All* to execute all code from top to bottom
3. Save file again to write any new output to disk
4. Select *File > Save and export Notebook as/ > HTML*.
5. Open in Google Chrome and print to PDF.
6. Submit to Gradescope