William Hazen
MEng - MIE
1009231225
2023-02-01

MIE 1613 HW1

1. Assume X is continuous & uniformly distributed in $[2, 10]$. We are intrested in $\theta = E[(X-5)^+]$. Note $a^+ = Max(a, 0)$

a) Compute θ exactly using the definition of expected value. Hint $f(x) = \frac{1}{b-a}$, $x \in [a,b]$ & 0 otherwise

$$x \in [2,10] \rightarrow f(x) = \frac{1}{10-2} = \frac{1}{8}$$

$$\theta = E[(x-5)^+] = \int_2^{10} (x-5)^+ \cdot f(x) dx$$

$$(x-5)^+ = \begin{cases} 0 \text{ if } 2 \leq x \leq 5 \\ a^+ \text{ if } 5 \leq x \leq 10 \end{cases}$$

$$\theta = \int_2^{10} (x-5)^+ \cdot f(x)\, dx = \int_2^{10} (x-5)^+ \cdot \tfrac{1}{8}\, dx$$

$$= \frac{1}{8}\left[\int_2^5 (x-5)^+ dx + \int_5^{10} (x-5)^+ dx\right]$$

$$= \frac{1}{8}\left[\int_2^5 0\, dx + \int_5^{10} (x-5)^+ dx\right]$$

$$= \frac{1}{8}\left[0 + \int_5^{10} (x-5)^+ dx\right]$$

$$= \frac{1}{8}\left[\frac{1}{2}x^2\Big|_5^{10} + 5x\Big|_5^{10}\right]$$

$$= \frac{1}{8}\left[\left(\frac{75}{2}\right) - (25)\right] = \frac{25}{16} \approx 1.5625$$

$$\theta = \frac{25}{16}$$

4. What is the standard error of the estimator $\overline{x_n}$ where $\bar{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$.

$$\bar{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{\& } x_i\text{'s are iid}$$

$$E[\overline{x_n}] = E\left[\frac{1}{n}\sum_{i=1}^{n} x_i\right] = \frac{1}{n}E\left[\sum_{i=1}^{n} x_i\right] = \frac{1}{\cancel{n}}\cdot \cancel{n}\, E[x]$$

↳ linearity of expectation property

$$E[\bar{x}_n] = E[x]$$

$$Var(\bar{x}_n) = Var\left[\frac{1}{n}\sum_{i=1}^{n} x_i\right] = \frac{1}{n^2}Var\left[\sum_{i=1}^{n} x_i\right]$$

↳ constant multiplication property of variance

$$= \frac{1}{n^2}\sum_{i=1}^{n} Var(x_i)$$

→ sum of the variance of iid variables is equal to sum of each var

$$= \frac{1}{n^{\cancel{2}}}\cdot \cancel{n}\, Var(X)$$

↳ since X_i's are iid

$$Var(\bar{x}_n) = \frac{1}{n^2}Var(X)$$

let $\sigma^2 = Var(X)$

Thus the standard error (SE) of the sample mean is

$$SE = \sqrt{\frac{Var(X)}{n}} = \frac{\sqrt{Var(X)}}{\sqrt{n}} = \frac{\sigma}{\sqrt{n}}$$

$$SE = \frac{\sigma}{\sqrt{n}} \quad //$$

5. Prove the following

Let $\mu_X = E[X]$ & $\sigma^2 = Var(X)$

a) $Var(aX+b) = a^2 Var(X)$

$Var(ax+b) = E[((aX+b) - E[aX+b])^2]$

↳ Property of Variance (i)
$V(X) = E[(X-\mu_X)^2]$

$= E[((aX+\cancel{b}) - aE[X] - \cancel{b})^2]$

↳ (ii) expected value of a constant b is equal to b

→ (iii) linearity of expectation

$= E[(aX - a\mu_X)^2]$

$= E[a^2(X-\mu_X)^2]$

$= a^2 E[(X-\mu_X)^2]$ ← (iii)

$= a^2 Var(X)$ ← (i)

∴ Thus $Var(aX+b) = a^2 Var(X)$ //

b) $Var(X+Y) = Var(X) + Var(Y) = 2Cov(X,Y)$

Let $\mu_X = E[X]$, $\mu_Y = E[Y]$, $cov(X,Y) = E[(X-\mu_X)(Y-\mu_Y)]$

$$Var(X+Y) = E[((X+Y) - E(X+Y))^2] \leftarrow (i)$$

$$= E[(X+Y-\mu_X-\mu_Y)^2]$$

$$= E[((X-\mu_X) + (Y-\mu_Y))^2]$$

$$= E[(X-\mu_X)^2 + (Y-\mu_Y)^2 + 2(X-\mu_X)(Y-\mu_Y)]$$

$$= E[(X-\mu_X)^2] + E[(Y-\mu_Y)^2] + 2E[(X-\mu_X)(Y-\mu_Y)]$$

From (i) $\Rightarrow Var(X) + Var(Y) + 2Cov(X,Y)$

$\therefore$ Thus, $Var(X+Y) = Var(X) + Var(Y) + 2Cov(X,Y)$ //

- Name: William Hazen
- Student ID: 1009231225
- Department: MIE
- Program: M.Eng.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

# fix random number seed
np.random.seed(1)
```

1.

1b)

In [2]:
```
# X is cont. U[2,10]
# theta = E[(X-5)^+]
# a^+ = Max(a,0) -> a^+ = 0 if a < 0
#                -> a^+ = a if a >= 0

ab_value_list = []  # initalize list
a = 2
b = 10
n = 100000  # 100k samples

for i in range(n):
    random_value = np.random.random()  # create random float between 0.0 & 1.0
    # scale by difference between a & b and shift by a
    ab_value = (b-a) * random_value + a
    # output is uniformed list of values between 2 & 10
    ab_value_list.append(ab_value)

theta = []
for x in ab_value_list:
    if x < 5:   # if x is less than 5 then a^+ = 0
        theta.append(0)
    else:       # if x >= 5 then a^+ = a
        theta.append((x-5))
```

In [3]:
```
average_theta_list = []
n = 0
for value in theta:  # Loop through the values in theta
    n += 1
    if n == 1:
        average_theta_list.append(value)  # append the first value of theta
    else:
        average_theta_list.append(
            (1/n) * ((n-1)*average_theta_list[-1] + value))
        # running average = adding the new number to the old average and divid
ing by the total number of samples

theta_mean = np.mean(theta)  # mu of theta list
theta_std = np.std(theta)  # std of theta list


print("Average theta =", theta_mean)
# 95% confidence interval: alpha = mu +- 1.96 std/sqrt(n)
print('95% Confidence interval theta:', np.mean(theta),
      "+/-", 1.96*np.std(theta, ddof=1)/np.sqrt(n))

# 95% confidence interval: alpha = mu +- 1.96 std/sqrt(n)
alpha_neg = theta_mean - (1.96*theta_std)/np.sqrt(n)
alpha_plus = theta_mean + (1.96*theta_std)/np.sqrt(n)

print(" negative bound =", alpha_neg)
print(" Positive bound =", alpha_plus)
```

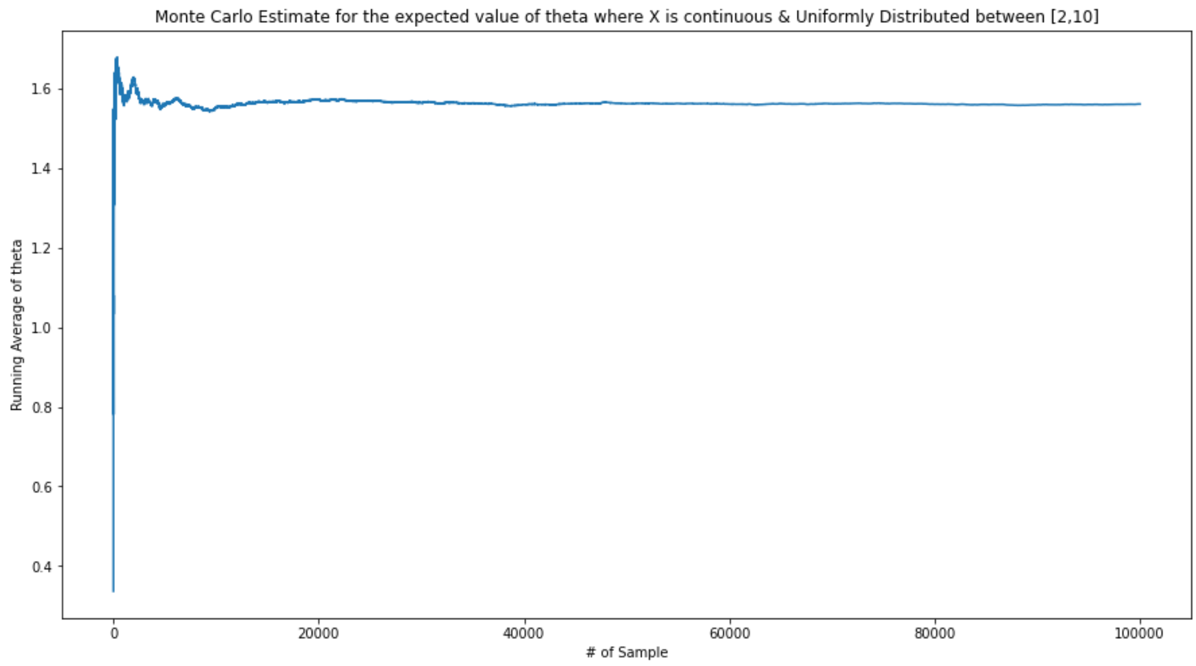
```
Average theta = 1.5613895874744266
95% Confidence interval theta: 1.5613895874744266 +/- 0.010318273591054833
 negative bound = 1.5510713654748687
 Positive bound = 1.5717078094739845
```

- 1a) theta = 1.5625

1c)

In [4]:
```
plt.figure(figsize=(15, 8))
plt.plot(average_theta_list)
plt.title(
    "Monte Carlo Estimate for the expected value of theta where X is continuou
s & Uniformly Distributed between [2,10]")
plt.xlabel("# of Sample")
plt.ylabel("Running Average of theta")
```

Out[4]: `Text(0, 0.5, 'Running Average of theta')`



2.

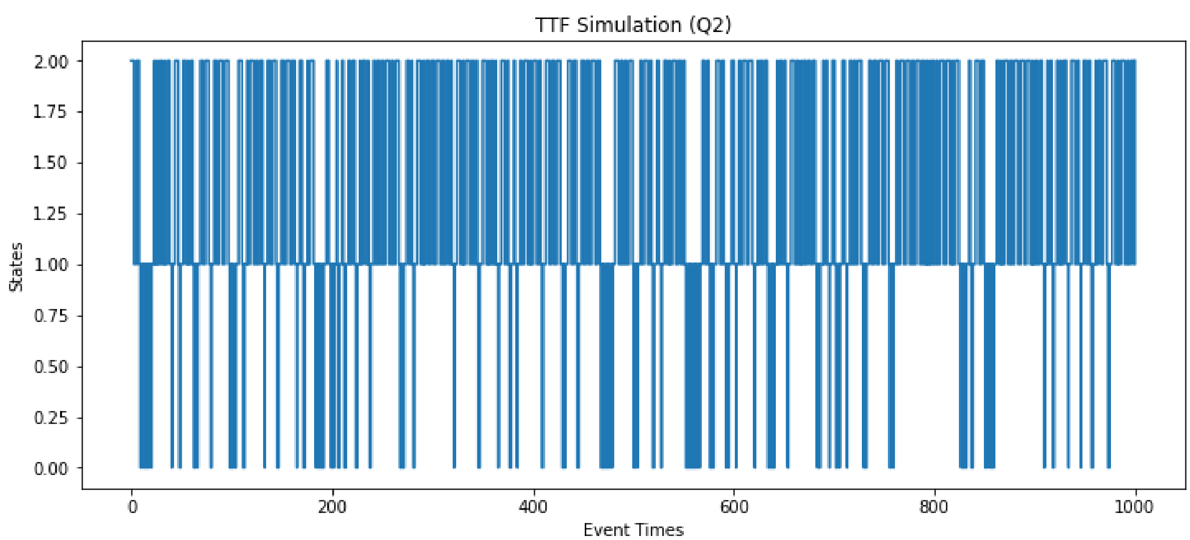
In [5]:
```
def Q2_TTF(T):
    # pesudo random
    np.random.seed(1)

    # start with 2 functioning components at time 0
    clock = 0
    S = 2
    # initialize the time of events
    NextRepair = float('inf')
    NextFailure = np.ceil(6*np.random.random())
    # lists to keep the event times and the states
    EventTimes = [0]
    States = [S]

    A_t = {}

    '''
    Logic: While the runtime is less than the value of T, clock is stored as t
he min value of either the repair or failure time.
        If the repair value is less than the failure, we initalize the NextRep
air to inf, however if S = 1 then we reset and schedule the next repair and fa
ilure time.
        Similarly if the repair value is greater than the failure, we have the
same condition but initalize NextFailure to inf.
    '''

    while clock < T:  # changed while loop from S>0 to clock<T
        # advance the time till next event

        clock = min(NextRepair, NextFailure, T)  # Stops when clock = T

        if NextRepair < NextFailure:  # next event is completion of a repair
            S = S + 1
            if S == 2:
                # Can only be 1 or 2 thus NextRepair = inf if S!=1
                NextRepair = float('inf')
            elif S == 1:  # When S = 1, schedule next repair and failure
                NextRepair = clock + 2.5
                NextFailure = clock + np.ceil(6*np.random.random())

        else:  # next event is a failure
            S = S - 1
            if S == 0:
                # Can only be 1 or 0 thus NextFailure = inf if S!=1
                NextFailure = float('inf')
            elif S == 1:
                NextRepair = clock + 2.5
                NextFailure = clock + np.ceil(6*np.random.random())

        # save the time and state
        EventTimes.append(clock)
        States.append(S)
        last_clock = clock

        # Stores value of 1 when State = 1 at time t
        for idx, state in enumerate(States):
```

```
            if state == 2:
                A_t[EventTimes[idx]] = 1
            else:
                A_t[EventTimes[idx]] = 0

    # plot the sample path
    print('T =', int(T))
    print('Clock =', last_clock)
    print('Event Time length =', len(EventTimes))
    print('A_t =', sum(A_t.values())/len(EventTimes))
    plt.figure(figsize=(12, 5))
    plt.plot(EventTimes, States, drawstyle='steps-post')
    plt.title("TTF Simulation (Q2)")
    plt.xlabel("Event Times")
    plt.ylabel("States")
    plt.show()
```

2a)

In [6]:
```
Q2_TTF(1000)
```

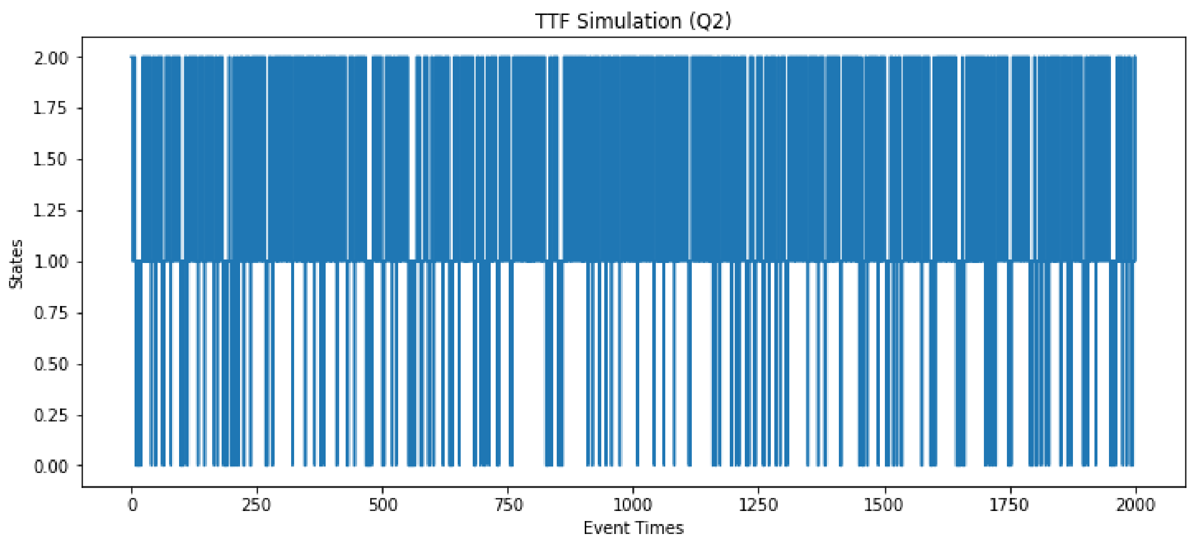
```
T = 1000
Clock = 1000
Event Time length = 513
A_t = 0.32748538011695905
```



2b)

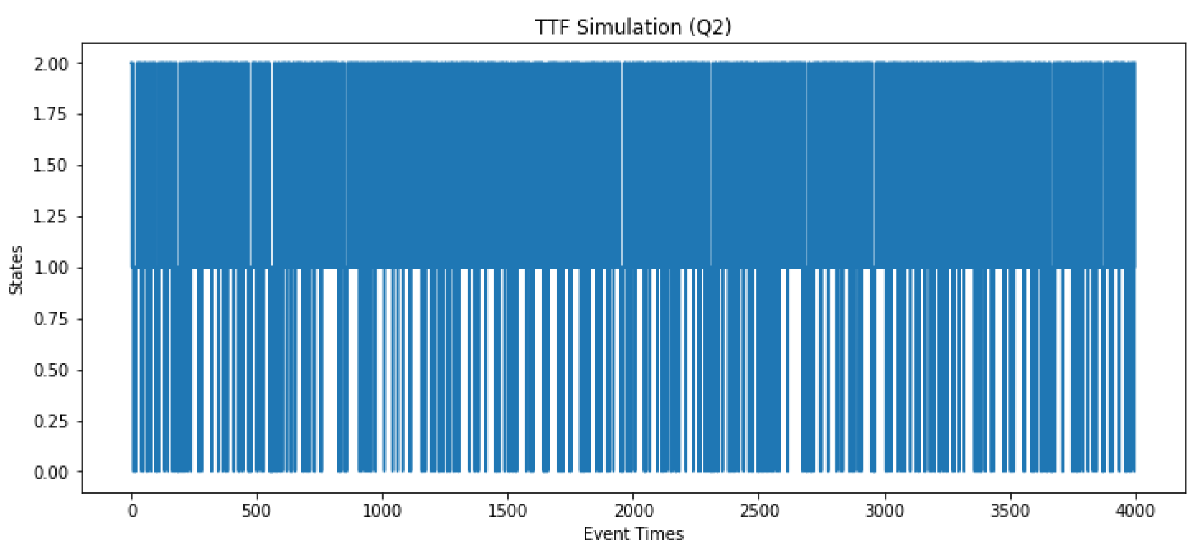
In [7]:
```
Q2_TTF(2000)
```

```
T = 2000
Clock = 2000
Event Time length = 1031
A_t = 0.3326867119301649
```



In [8]:
```
Q2_TTF(4000)
```

```
T = 4000
Clock = 4000.0
Event Time length = 2083
A_t = 0.33461353816610656
```



- T=1k: 0.32748538011695905
- T=3K: 0.3326867119301649
- T=4k: 0.33461353816610656

Observations

- The fraction of the system being fully functional increases with time.

3.

In [9]:
```
def Q3_TTF(N):
    # Set number of replications
    Rep = 1000
    # Define lists to keep samples of the outputs across replications
    TTF_list = []

    # fix random number seed
    np.random.seed(1)

    for rep in range(0, Rep):
        clock = 0
        S = N
        Spare = N - 1  # number of spare components

        # initialize the time of events
        NextRepair = float('inf')
        NextFailure = np.ceil(6*np.random.random())
        EventTimes = [0]
        States = [S]

        '''
        Logic: For each replication, while S > 0, if the repair value is less
than the failure, if S = N then there is no need to schedule a repair.
            However if not then the NextRepair will be scheduled if there is a
spare avaliable.
            If NextRepair >= NextFailure then the next failure is scheduled in
itally unless S = 0.
            If S!=0 then we schedule the next repair [The reason why the NextF
ailure is outside is
              because scheduling nextfailure does not depend on the spare comp
onents]
            If the repair value is greater than the failure, we have the same
condition but initalize NextFailure to inf.

            I accounted for each value of S in the event that Repair<Fail,
            But In order to make sure only one repair can happen at a time
            when Fail<Repair, a NextRepair is scheduled when S = N-1
        '''

        while S > 0:
            # advance the time till next event
            clock = min(NextRepair, NextFailure)

            if NextRepair < NextFailure:  # next event is completion of a repa
ir
                S = S + 1
                Spare = Spare + 1 # Spare components increases with S

                if Spare == N: # Max spare components = N-1
                    Spare = N-1

                if S == N: # Max components -> no repairs
                    NextRepair = float('inf')

                else:
                    if Spare > 0:  # If there are spare components available
```

```
                        Spare = Spare
                        NextRepair = clock + 2.5

            else:  # next event is a failure
                S = S - 1
                NextFailure = clock + np.ceil(6*np.random.random())

                if S == 0:
                    NextFailure = float('inf')
                else:
                    if Spare > 0:
                        Spare = Spare - 1
                        if S == (N-1):
                            NextRepair = clock + 2.5

            # save the time and state
            EventTimes.append(clock)
            States.append(S)

        # save the TTF and average # of func. components
        TTF_list.append(clock)

        #From Q1
        average_TTF_list = []
        n = 0
        for value in TTF_list:
            n += 1
            if n == 1:
                average_TTF_list.append(value)
            else:
                average_TTF_list.append((1/n) * ((n-1)*average_TTF_list[-1] +
value))

    print("For N = ", N)
    print('TTF List Length =', len(TTF_list))
    print('Estimated expected TTF:', np.mean(TTF_list))
    print('95% CI for TTF:', np.mean(TTF_list), "+/-", 1.96*np.std(TTF_list, d
dof = 1)/np.sqrt(Rep))
```

In [10]:
```
Q3_TTF(2)
```

```
For N =  2
TTF List Length = 1000
Estimated expected TTF: 14.193
95% CI for TTF: 14.193 +/- 0.7212458114694061
```

In [11]:
```
Q3_TTF(3)
```

```
For N =  3
TTF List Length = 1000
Estimated expected TTF: 87.648
95% CI for TTF: 87.648 +/- 5.523633399450503
```

In [12]:
```
Q3_TTF(4)
```

```
For N =  4
TTF List Length = 1000
Estimated expected TTF: 648.805
95% CI for TTF: 648.805 +/- 42.26686864295534
```

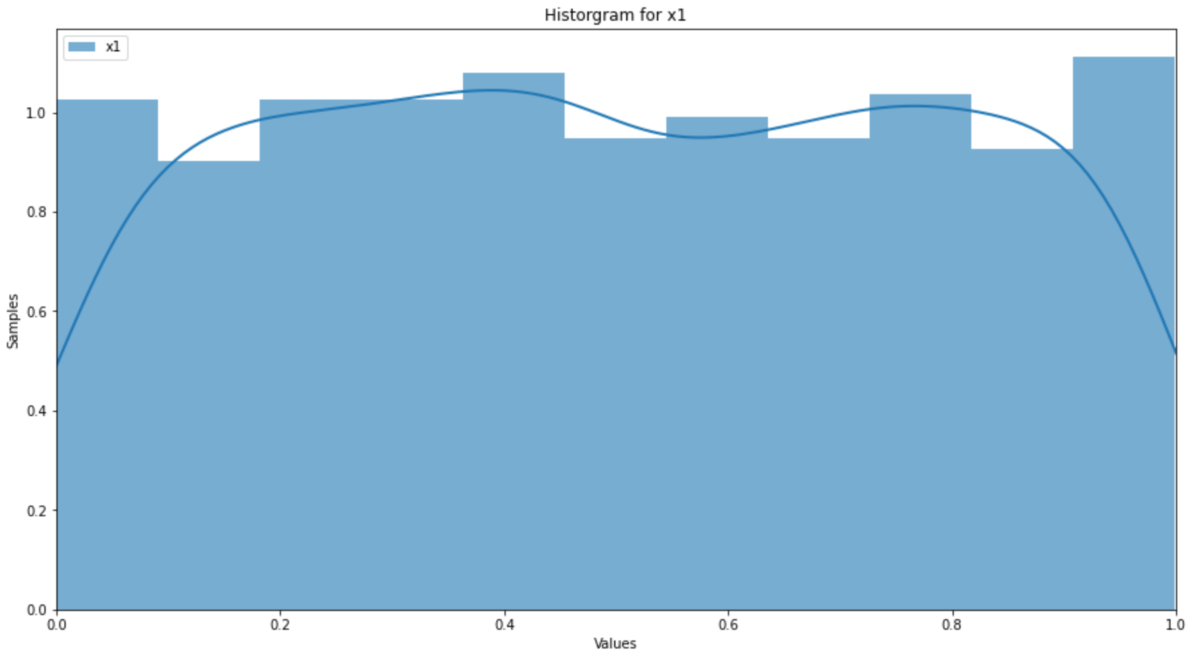
6.

In [13]:
```
# X_n = 1/n sum(X_i)
def X_n(n):  # create a X_n function
    sum_xn = 0
    for i in range(n):
        # Random uniformed values between 0 & 1
        X = np.random.uniform(0, 1, 1000)
        sum_xn += X  # sum
    X_n = (1/n)*sum_xn
    return X_n


x1 = X_n(1)
x2 = X_n(2)
x30 = X_n(30)
x500 = X_n(500)

x_s = {"x1": x1, "x2": x2, "x30": x30, "x500": x500}

for x in (x_s):
    print("Length of " + x + ":", len(x_s[x]))
```

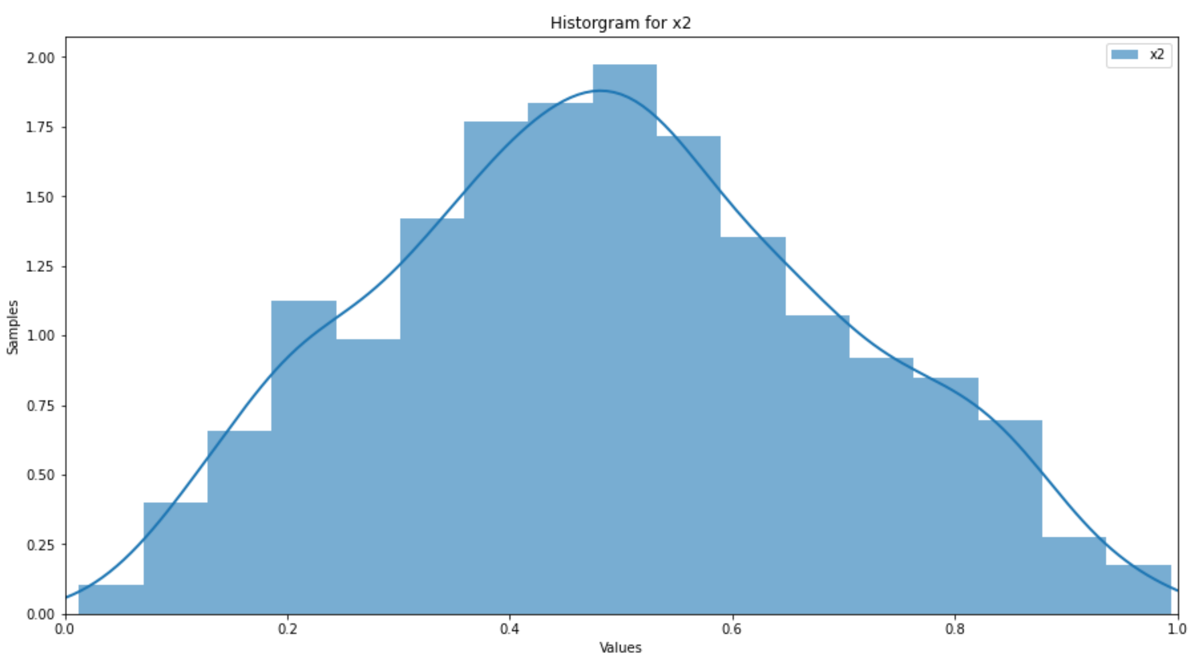
```
Length of x1: 1000
Length of x2: 1000
Length of x30: 1000
Length of x500: 1000
```

In [14]:
```
import seaborn as sns
kwargs = dict(hist_kws={'alpha': .6}, kde_kws={'linewidth': 2})

for key, value in x_s.items():
    plt.figure(figsize=(15, 8))
    sns.distplot(value, **kwargs, label=key)
    plt.xlim([0, 1])
    plt.title(f"Historgram for {key}")
    plt.xlabel('Values')
    plt.ylabel('Samples')
    plt.legend()
    plt.show()
```

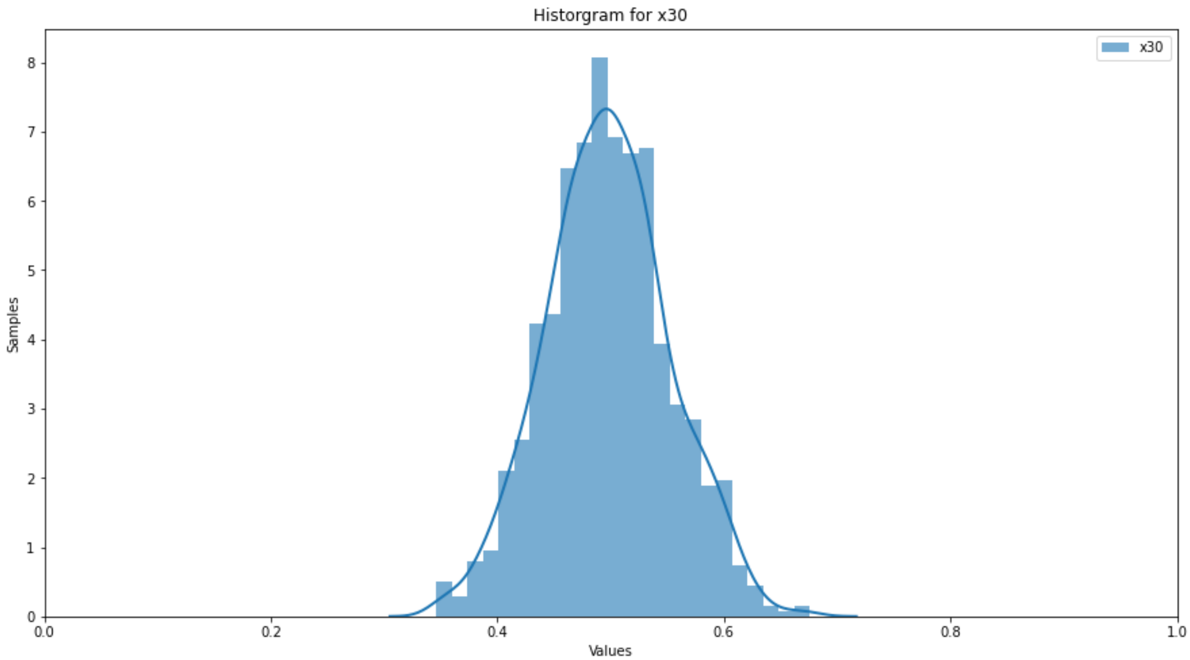
```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```



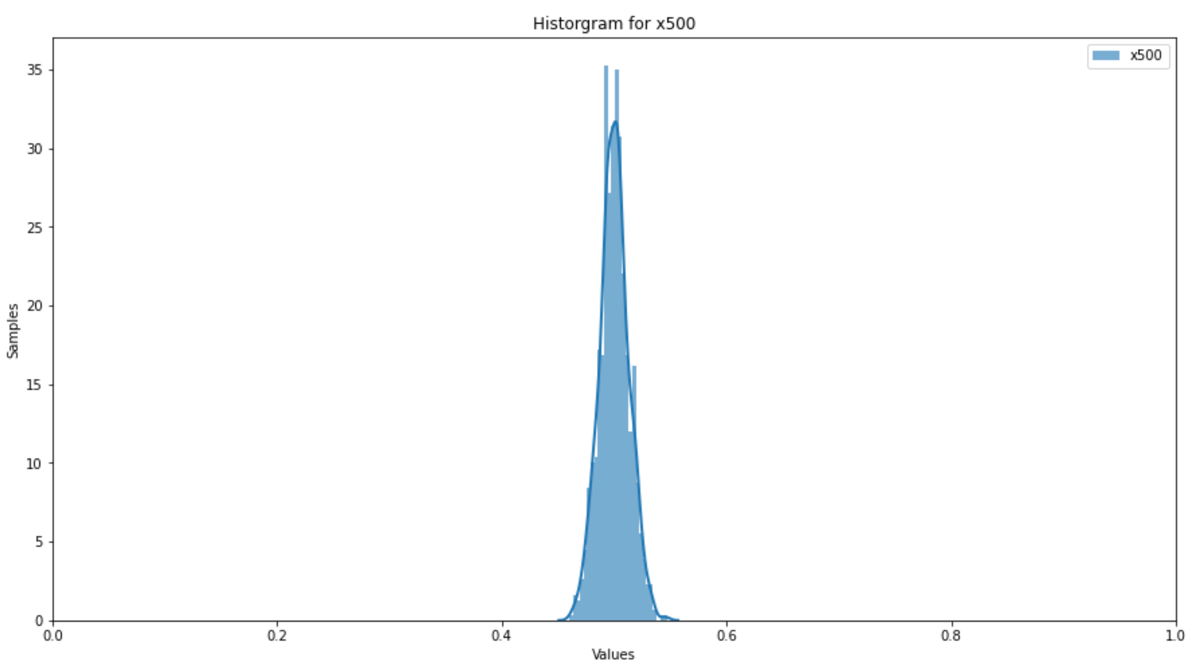

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```



```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```

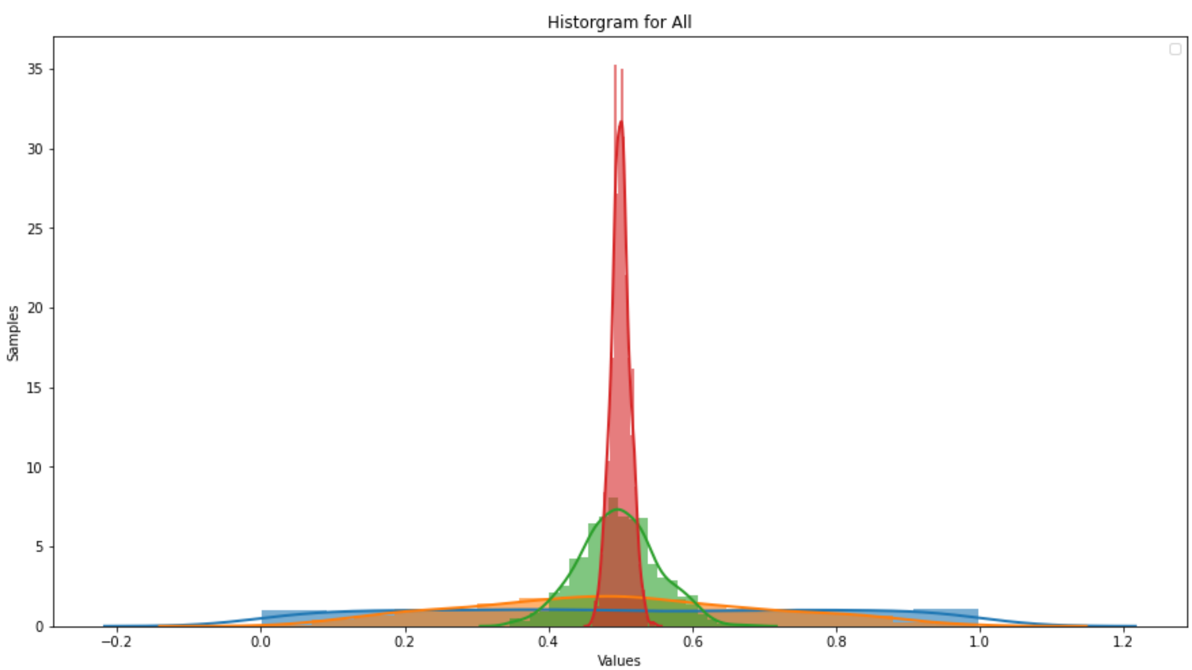



```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```



In [15]:
```
plt.figure(figsize=(15, 8))
for key, value in x_s.items():
    sns.distplot(value, **kwargs)
plt.title("Historgram for All")
plt.xlabel('Values')
plt.ylabel('Samples')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
WARNING:matplotlib.legend:No handles with labels found to put in legend.
```



What happens to the mean and variability of the sample means as n increases? Relate your observations to the Central Limit Theorem discussed in the class.

- As n increases the mean and variability converge to a central point. In X_1, we can see high variability and as n increases we can see the variability decrease and the mean hovers around 0.5 at X_500. Thus as the sample size increases the distributation closely resembles a normal distribution and becomes tightly clustered around the mean, which is a behaviour that agrees with the Central Limit Theorem.

In [16]:
```
%%shell
jupyter nbconvert --to html /content/HW1.ipynb
```

```
[NbConvertApp] WARNING | pattern '/content/HW1.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abo
rt conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.exp
ort_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.exp
ort_format=notebook --FilesWriter.build_directory= --ClearOutputPreprocessor.
enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateEx
```

```
porter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateE
xporter.exclude_input=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook',
'pdf', 'python', 'rst', 'script', 'slides']
            or a dotted object name that represents the import path for an
            `Exporter` class
    Default: 'html'
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template file to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_file]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                  to output to the directory of each noteboo
k. To recover
                                  previous default behaviour (outputting to t
he current
                                  working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a
copy
            of reveal.js.
```

```
            For speaker notes to work, this must be a relative path to a loca
l
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-
html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb

            which will convert mynotebook.ipynb to the default format (probab
ly HTML).

            You can specify the export format with `--to`.
            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdow
n', 'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX incl
udes
            'base', 'article' and 'report'.  HTML includes 'basic' and 'ful
l'. You
            can specify the flavor of the format used.

            > jupyter nbconvert --to html --template basic mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple o
f
            different ways:
```

```
            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containin
g::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.


---------------------------------------------------------------------------
CalledProcessError                        Traceback (most recent call last)
<ipython-input-16-ede384e11680> in <module>
----> 1 get_ipython().run_cell_magic('shell', '', 'jupyter nbconvert --to htm
l /content/HW1.ipynb\n')

/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshell.py in ru
n_cell_magic(self, magic_name, line, cell)
   2357             with self.builtin_trap:
   2358                 args = (magic_arg_s, cell)
-> 2359                 result = fn(*args, **kwargs)
   2360             return result
   2361

/usr/local/lib/python3.8/dist-packages/google/colab/_system_commands.py in _s
hell_cell_magic(args, cmd)
    107   result = _run_command(cmd, clear_streamed_output=False)
    108   if not parsed_args.ignore_errors:
--> 109     result.check_returncode()
    110   return result
    111

/usr/local/lib/python3.8/dist-packages/google/colab/_system_commands.py in ch
eck_returncode(self)
    132   def check_returncode(self):
    133     if self.returncode:
--> 134       raise subprocess.CalledProcessError(
    135           returncode=self.returncode, cmd=self.args, output=self.outp
ut)
    136

CalledProcessError: Command 'jupyter nbconvert --to html /content/HW1.ipynb
' returned non-zero exit status 255.
```