DA6400Tutorial1

February 7, 2025

[40]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from IPython.display import display, HTML
from typing import NamedTuple, List
import math
```

0.0.1 Gaussian Bandit Environment

[41]:
```
class GaussianArm(NamedTuple):
  mean: float
  std: float



class Env:
  def __init__(self, num_arms: int, mean_reward_range: tuple, std: float):
    """
    num_arms: number of bandit arms
    mean_reward_range: mean reward of an arm should lie between the given range
    std: standard deviation of the reward for each arm
    """
    self.num_arms = num_arms
    self.arms = self.create_arms(num_arms, mean_reward_range, std)

  def create_arms(self, n: int, mean_reward_range: tuple, std: float) -> dict:
    low_rwd, high_rwd = mean_reward_range
    # creates "n" number of mean reward for each arm
    means = np.random.uniform(low=low_rwd, high=high_rwd, size=(n,))
    arms = {id: GaussianArm(mu, std) for id, mu in enumerate(means)}
    return arms

  @property
  def arm_ids(self):
    return list(self.arms.keys())

  def step(self, arm_id: int) -> float:
    arm = self.arms[arm_id]
    return np.random.normal(arm.mean, arm.std)   # Reward

  def get_best_arm_and_expected_reward(self):
    best_arm_id = max(self.arms, key=lambda x: self.arms[x].mean)
    return best_arm_id, self.arms[best_arm_id].mean

  def get_avg_arm_reward(self):
    arm_mean_rewards = [v.mean for v in self.arms.values()]
    return np.mean(arm_mean_rewards)

  def plot_arms_reward_distribution(self, num_samples=1000):
    """
    This function is only used to visualize the arm's distrbution.
    """
    fig, ax = plt.subplots(1, 1, sharex=False, sharey=False, figsize=(9, 5))
    colors = sns.color_palette("hls", self.num_arms)
    for i, arm_id in enumerate(self.arm_ids):
      reward_samples = [self.step(arm_id) for _ in range(num_samples)]
      sns.histplot(reward_samples, ax=ax, stat="density", kde=True, bins=100,␣
↪color=colors[i], label=f'arm_{arm_id}')
    ax.legend()
    plt.show()
```

0.0.2 Policy

[42]:
```
class BasePolicy:
  @property
  def name(self):
    return 'base_policy'

  def reset(self):
    """
    This function resets the internal variable.
    """
    pass

  def update_arm(self, *args):
    """
    This function keep track of the estimates
    that we may want to update during training.
    """
    pass

  def select_arm(self) -> int:
    """
    It returns arm_id
    """
    raise Exception("Not Implemented")
```

Random Policy

[43]:
```
class RandomPolicy(BasePolicy):
  def __init__(self, arm_ids: List[int]):
    self.arm_ids = arm_ids

  @property
  def name(self):
    return 'random'

  def reset(self) -> None:
    """No use."""
    pass

  def update_arm(self, *args) -> None:
    """No use."""
    pass

  def select_arm(self) -> int:
    return np.random.choice(self.arm_ids)
```

[44]:
```
class EpGreedyPolicy(BasePolicy):
  def __init__(self, epsilon: float, arm_ids: List[int]):
    self.epsilon = epsilon
    self.arm_ids = arm_ids
    self.Q = {id: 0 for id in self.arm_ids}
    self.num_pulls_per_arm = {id: 0 for id in self.arm_ids}

  @property
  def name(self):
    return f'ep-greedy ep:{self.epsilon}'

  def reset(self) -> None:
    self.Q = {id: 0 for id in self.arm_ids}
    self.num_pulls_per_arm = {id: 0 for id in self.arm_ids}

  def update_arm(self, arm_id: int, arm_reward: float) -> None:
    self.num_pulls_per_arm[arm_id] += 1
    self.Q[arm_id] = self.Q[arm_id] + (arm_reward - self.Q[arm_id]) / (self.
↪num_pulls_per_arm[arm_id])

  def select_arm(self) -> int:
    best_arm = max(self.Q, key=lambda x: self.Q[x])
    probabilities = {id:self.epsilon / len(self.arm_ids) for id in self.arm_ids}
    probabilities[best_arm] += 1 - self.epsilon
    return np.random.choice(self.arm_ids, p=list(probabilities.values()))
```

[45]:
```
class SoftmaxPolicy(BasePolicy):
  def __init__(self, tau : float, arm_ids:List[int]):
    self.tau = tau
    self.arm_ids = arm_ids
    self.Q = {id: 0 for id in self.arm_ids}
    self.num_pulls_per_arm = {id: 0 for id in self.arm_ids}

  @property
  def name(self):
    return f'softmax tau:{self.tau}'

  def reset(self):
    self.Q = {id: 0 for id in self.arm_ids}
    self.num_pulls_per_arm = {id: 0 for id in self.arm_ids}

  def update_arm(self, arm_id: int, arm_reward: float) -> None:
    self.num_pulls_per_arm[arm_id] += 1
    self.Q[arm_id] = self.Q[arm_id] + (arm_reward - self.Q[arm_id]) / (self.
↪num_pulls_per_arm[arm_id])

  def select_arm(self) -> int:
    max_q = max(self.Q.values())
    exp_values = np.array([np.exp((q - max_q) / self.tau) for q in self.Q.
↪values()])
    sum_exps = exp_values.sum()
    probabilities = {id: exp_val / sum_exps for id, exp_val in zip(self.
↪arm_ids, exp_values)}
    if sum(probabilities.values()) != 1:
      return max(self.Q, key=lambda x: self.Q[x])
    return np.random.choice(self.arm_ids, p=list(probabilities.values()))
```

[46]:
```
class UCB(BasePolicy):
  def __init__(self,c:float, env):
    self.c = c
    self.env = env
    self.arm_ids = self.env.arm_ids
    self.Q = {id: self.env.step(id) for id in self.arm_ids}
    self.num_pulls_per_arm = {id: 1 for id in self.arm_ids}

  @property
  def name(self):
    return f'UCB c:{self.c}'

  def reset(self):
    self.Q = {id: self.env.step(id) for id in self.arm_ids}
    self.num_pulls_per_arm = {id: 1 for id in self.arm_ids}

  def update_arm(self, arm_selected: int, arm_reward: float):
    self.num_pulls_per_arm[arm_selected] += 1
    self.Q[arm_selected] = self.Q[arm_selected] + (arm_reward - self.
↪Q[arm_selected])/self.num_pulls_per_arm[arm_selected]



  def select_arm(self) -> int:
    number_of_pulls = sum(self.num_pulls_per_arm.values())
    try :
      selected_arm = max(self.Q,key = lambda x : self.Q[x] + self.c*(np.sqrt(np.
↪log(number_of_pulls)/(self.num_pulls_per_arm[x]))))
    except RuntimeError:
      selected_arm = max(self.Q,key = lambda x : self.Q[x])
    return selected_arm
```

Trainer

```
[47]: def train(env, policy: BasePolicy, timesteps):
        policy_reward = np.zeros((timesteps,))
        for t in range(timesteps):
          arm_id = policy.select_arm()
          reward = env.step(arm_id)
          policy.update_arm(arm_id, reward)
          policy_reward[t] = reward
        return policy_reward


      def avg_over_runs(env, policy: BasePolicy, timesteps, num_runs):
        _, expected_max_reward = env.get_best_arm_and_expected_reward()
        policy_reward_each_run = np.zeros((num_runs, timesteps))
        for run in range(num_runs):
          policy.reset()
          policy_reward = train(env, policy, timesteps)
          policy_reward_each_run[run, :] = policy_reward

        # calculate avg policy reward from policy_reward_each_run
        avg_policy_rewards = np.mean(policy_reward_each_run,axis=0) # your code here␣
      ↪(type: nd.array, shape: (timesteps,))
        total_policy_regret = timesteps*expected_max_reward - np.
      ↪sum(avg_policy_rewards)# your code here (type: float)

        return avg_policy_rewards, total_policy_regret
```

[48]:
```
a = np.array([[1,2,3],[4,5,6]])
np.mean(a, axis=0)
```

[48]:
```
array([2.5, 3.5, 4.5])
```

[49]:
```
def plot_reward_curve_and_print_regret(env, policies, timesteps=200,␣
 ↪num_runs=500):
  fig, ax = plt.subplots(1, 1, sharex=False, sharey=False, figsize=(10, 6))
  for policy in policies:
    avg_policy_rewards, total_policy_regret = avg_over_runs(env, policy,␣
 ↪timesteps, num_runs)
    print('regret for {}: {:.3f}'.format(policy.name, total_policy_regret))
    ax.plot(np.arange(timesteps), avg_policy_rewards, '-', label=policy.name)

  _, expected_max_reward = env.get_best_arm_and_expected_reward()
  ax.plot(np.arange(timesteps), [expected_max_reward]*timesteps, 'g-')

  avg_arm_reward = env.get_avg_arm_reward()
  ax.plot(np.arange(timesteps), [avg_arm_reward]*timesteps, 'r-')

  plt.legend(loc='lower right')
  plt.show()
```

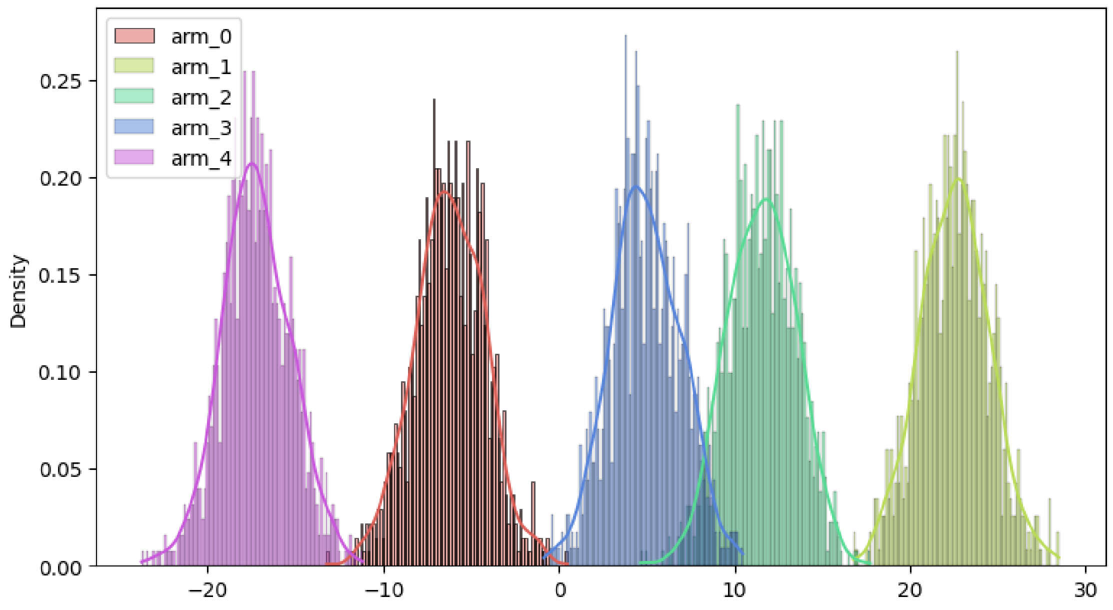
0.0.3 Experiments

[50]:
```
seed = 42
np.random.seed(seed)

num_arms = 5
mean_reward_range = (-25, 25)
std = 2.0
```

[51]:
```
env = Env(num_arms, mean_reward_range, std)

env.plot_arms_reward_distribution()
```



[52]:
```
best_arm, max_mean_reward = env.get_best_arm_and_expected_reward()
print(best_arm, max_mean_reward)
```

```
1 22.53571532049581
```

[53]:
```
print(env.get_avg_arm_reward())
```

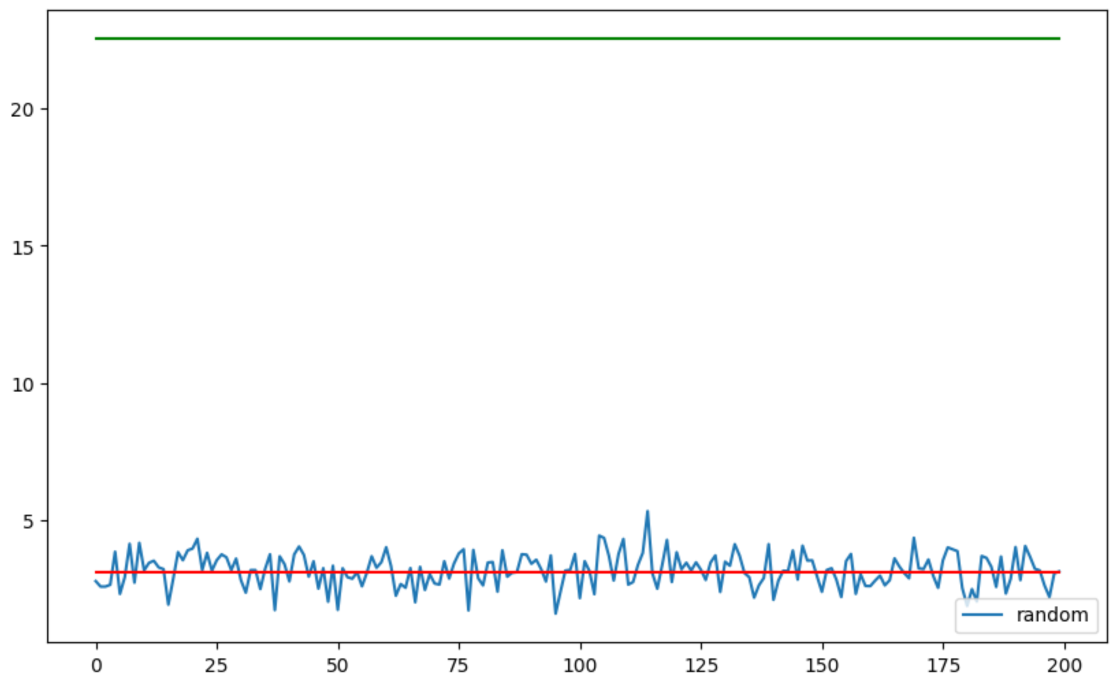
```
3.119254917081568
```

Please explore following values:

- Epsilon greedy: [0.001, 0.01, 0.5, 0.9]
- Softmax: [0.001, 1.0, 5.0, 50.0]

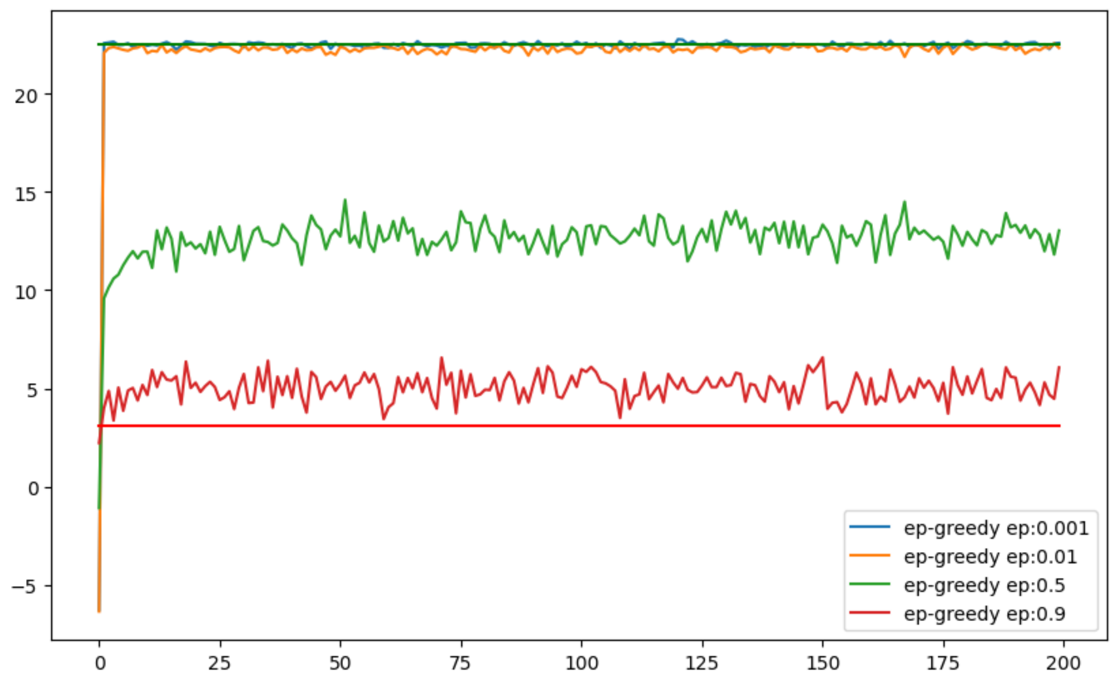
[54]:
```
random_policy = RandomPolicy(env.arm_ids)
plot_reward_curve_and_print_regret(env, [random_policy], timesteps=200,␣
 ↪num_runs=500)
```

```
regret for random: 3871.625
```



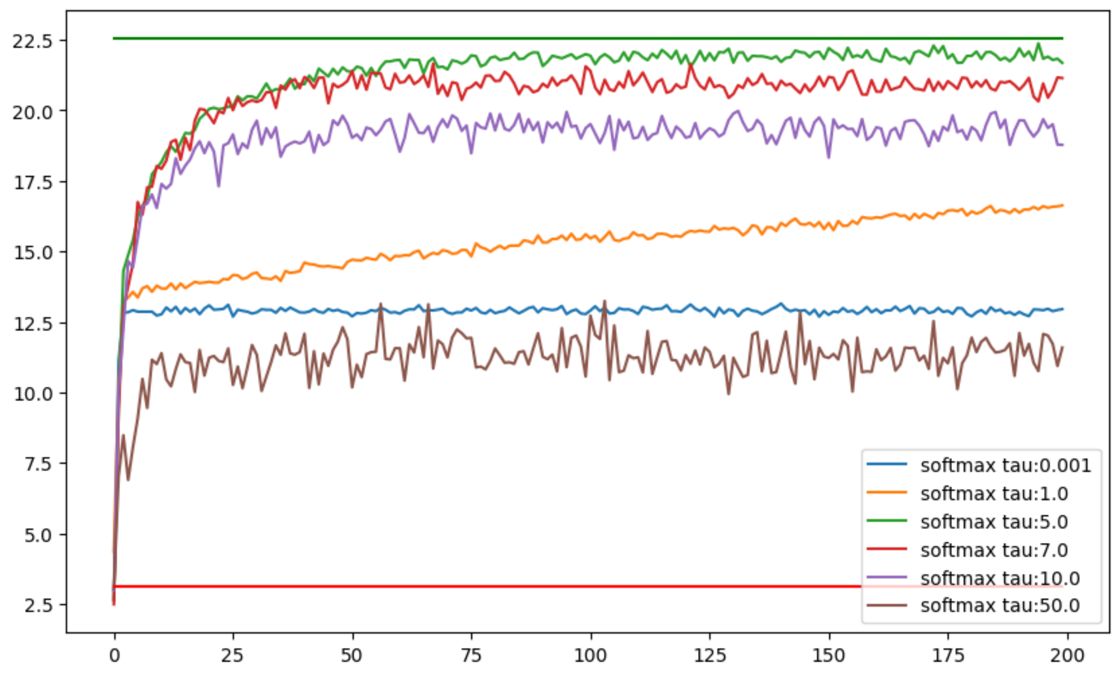
[55]:
```
explore_epgreedy_epsilons =  [0.001, 0.01, 0.5, 0.9]
epgreedy_policies = [EpGreedyPolicy(ep, env.arm_ids) for ep in␣
 ↪explore_epgreedy_epsilons]
plot_reward_curve_and_print_regret(env, epgreedy_policies, timesteps=200,␣
 ↪num_runs=500)
```

```
regret for ep-greedy ep:0.001: 35.471
regret for ep-greedy ep:0.01: 77.678
regret for ep-greedy ep:0.5: 1987.793
regret for ep-greedy ep:0.9: 3503.219
```



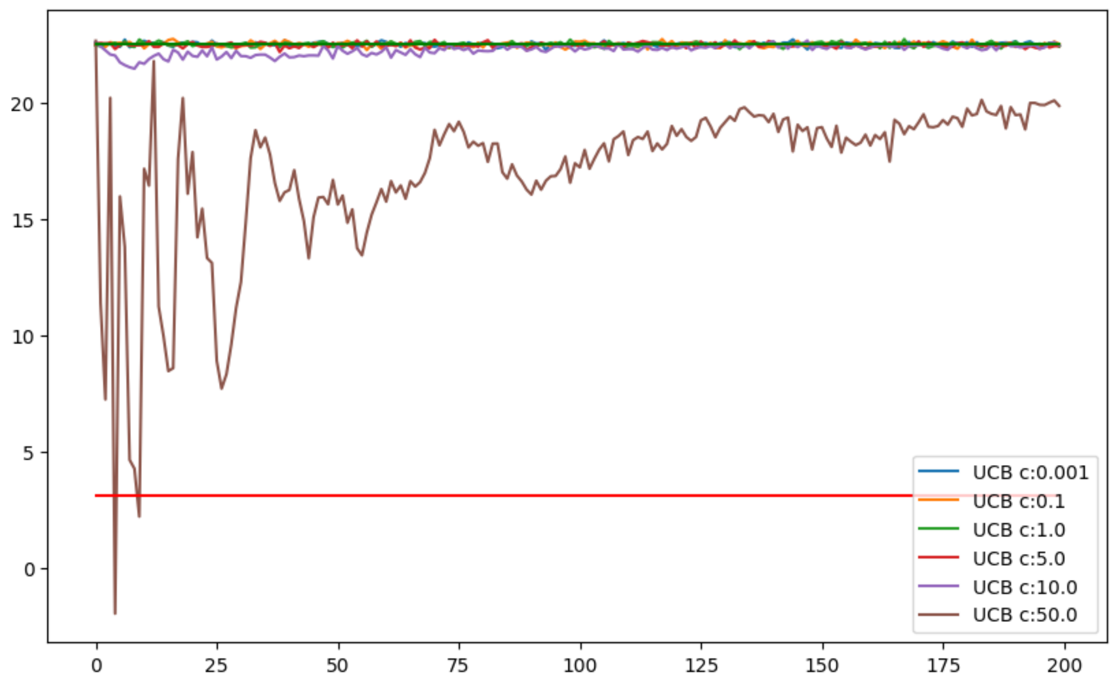
[56]:
```
explore_softmax_taus =  [0.001, 1.0, 5.0, 7.0, 10.0, 50.0]
softmax_polices = [SoftmaxPolicy(tau, env.arm_ids) for tau in␣
 ↪explore_softmax_taus]
plot_reward_curve_and_print_regret(env, softmax_polices, timesteps=200,␣
 ↪num_runs=500)
```

```
regret for softmax tau:0.001: 1940.060
regret for softmax tau:1.0: 1464.134
regret for softmax tau:5.0: 272.380
regret for softmax tau:7.0: 422.107
regret for softmax tau:10.0: 722.201
regret for softmax tau:50.0: 2273.763
```



[57]:
```python
explore_softmax_cs =  [0.001,0.1, 1.0, 5.0, 10.0 ,50.0]
ucb_polices = [UCB(c, env) for c in explore_softmax_cs]
plot_reward_curve_and_print_regret(env, ucb_polices, timesteps=200,␣
 ↪num_runs=500)
```

```
regret for UCB c:0.001: -0.065
regret for UCB c:0.1: -0.567
regret for UCB c:1.0: 0.298
regret for UCB c:5.0: 6.067
regret for UCB c:10.0: 46.867
regret for UCB c:50.0: 1081.013
```



From the setting performance RandomPolicy UCB,SoftmaxPolicy and EpGreedyPolicy were tested and their performance was compared.Following are the conclusions EpGreedyPolicy:

As the value of increases, the expected returns has a high offset from the maximum reward giving arm.(Policy is to explore more than to exploit)

It is clearly visible from the above fact that regret increases with increase in value of .

SoftmaxPolicy:

As the value of increases, intially regret decreases representing need for exploration but as further increases regret increase representing need of exploitation.

It is clearly visible from the above fact that regret increases with increase in value of .

UCB (Upper Confidence Bound)Policy:

As the value of c increases, allowing arms to more explore, it takes larger samples to select to choose maximum rewarding arm.

It is clearly visible from the above fact that regret increases with increase in value of c.

0.1 Generate PDF

[58]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[64]:
```
!apt install -y pandoc texlive-xetex
!pip install nbconvert
!jupyter nbconvert --to pdf /content/drive/MyDrive/ColabNotebooks/
 ↪DA6400Tutorial1.ipynb
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
pandoc is already the newest version (2.9.2.1-3ubuntu2).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 19 not upgraded.
Requirement already satisfied: nbconvert in /usr/local/lib/python3.11/dist-
packages (7.16.6)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.11/dist-
packages (from nbconvert) (4.13.1)
Requirement already satisfied: bleach!=5.0.0 in /usr/local/lib/python3.11/dist-
packages (from bleach[css]!=5.0.0->nbconvert) (6.2.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.11/dist-
packages (from nbconvert) (0.7.1)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.11/dist-
packages (from nbconvert) (3.1.5)
Requirement already satisfied: jupyter-core>=4.7 in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (0.3.0)
Requirement already satisfied: markupsafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (3.0.2)
Requirement already satisfied: mistune<4,>=2.0.3 in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (3.1.1)
Requirement already satisfied: nbclient>=0.5.0 in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (0.10.2)
Requirement already satisfied: nbformat>=5.7 in /usr/local/lib/python3.11/dist-
packages (from nbconvert) (5.10.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-
packages (from nbconvert) (24.2)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (1.5.1)
Requirement already satisfied: pygments>=2.4.1 in
/usr/local/lib/python3.11/dist-packages (from nbconvert) (2.18.0)
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.11/dist-
packages (from nbconvert) (5.7.1)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-
packages (from bleach!=5.0.0->bleach[css]!=5.0.0->nbconvert) (0.5.1)
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from bleach[css]!=5.0.0->nbconvert)
(1.4.0)
Requirement already satisfied: platformdirs>=2.5 in
```

```
/usr/local/lib/python3.11/dist-packages (from jupyter-core>=4.7->nbconvert)
(4.3.6)
Requirement already satisfied: jupyter-client>=6.1.12 in
/usr/local/lib/python3.11/dist-packages (from nbclient>=0.5.0->nbconvert)
(6.1.12)
Requirement already satisfied: fastjsonschema>=2.15 in
/usr/local/lib/python3.11/dist-packages (from nbformat>=5.7->nbconvert) (2.21.1)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.11/dist-packages (from nbformat>=5.7->nbconvert) (4.23.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-
packages (from beautifulsoup4->nbconvert) (2.6)
Requirement already satisfied: typing-extensions>=4.0.0 in
/usr/local/lib/python3.11/dist-packages (from beautifulsoup4->nbconvert)
(4.12.2)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-
packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (25.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.11/dist-packages (from
jsonschema>=2.6->nbformat>=5.7->nbconvert) (2024.10.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.11/dist-packages (from
jsonschema>=2.6->nbformat>=5.7->nbconvert) (0.36.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dist-
packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (0.22.3)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.11/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (24.0.1)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.11/dist-packages (from jupyter-
client>=6.1.12->nbclient>=0.5.0->nbconvert) (2.8.2)
Requirement already satisfied: tornado>=4.1 in /usr/local/lib/python3.11/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (6.4.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.1->jupyter-
client>=6.1.12->nbclient>=0.5.0->nbconvert) (1.17.0)
[NbConvertApp] WARNING | pattern
'/content/drive/MyDrive/ColabNotebooks/DA6400Tutorial1.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 abort
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.export_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.export_format=notebook --FilesWriter.build_directory=
--ClearOutputPreprocessor.enabled=True]
--coalesce-streams
    Coalesce consecutive stdout and stderr outputs into one stream (within each
cell).
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--CoalesceStreamsPreprocessor.enabled=True]
--no-prompt
```

```
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True
--TemplateExporter.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
--TemplateExporter.exclude_input=True
--TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the
system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful
for the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=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', 'qtpdf', 'qtpng', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
```

```
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--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.
                Supports pattern replacements '{notebook_name}'.
    Default: '{notebook_name}'
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                  to output to the directory of each notebook.
To recover
                                  previous default behaviour (outputting to the
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 local
            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 --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides',
'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

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

            > jupyter nbconvert --to html --template lab 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 of
            different ways:

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

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

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

            > jupyter nbconvert --config mycfg.py

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