```
import random
class Environment :
  def __init__(self):
    self.steps_left=10
  def get_observation(self):
    return[0.0,0.0,0.0,0.0]
  def get_actions(self):
    return[0,1]
  def is_done(self):
    return self.steps_left==0
  def action(self,action):
    if self.is_done():
      print("game over")
    self.steps_left-=1
    return random.random()
```

```
class Agent:
  def __init__(self):
    self.total_reward=0.0
  def steps(self,env):
    current_obs=env.get_observation()
    actions=env.get_observation()
    reward=env.action(random.choice(actions))
    self.total_reward+=reward
```

```
 # Instantiate the classess using constructors
env=Environment()
agent=Agent()
while not env.is_done():
  agent.steps(env)
print("the total reward is %d"% agent.total_reward)
```

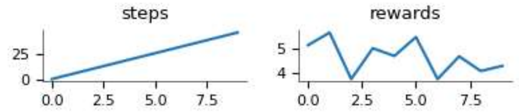
```
     the total reward is 6
```

```
steps=[]
rewards=[]
for i in range(1,50,5):
  env=Environment()
  agent=Agent()
  while not env.is_done():
    agent.steps(env)
  steps.append(i)
  rewards.append(agent.total_reward)
```

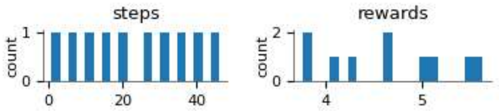
```
import pandas as pd
df=pd.DataFrame({"steps":steps,"rewards":rewards
})
df
```

	steps	rewards
0	1	5.120370
1	6	5.625784
2	11	3.769178
3	16	4.999234
4	21	4.691257
5	26	5.445466
6	31	3.769240
7	36	4.676093
8	41	4.090535
9	46	4.295483

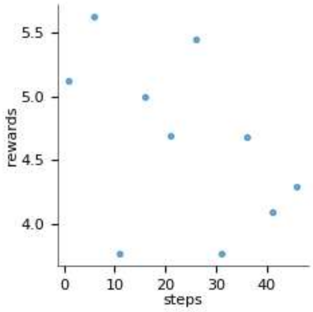
Values



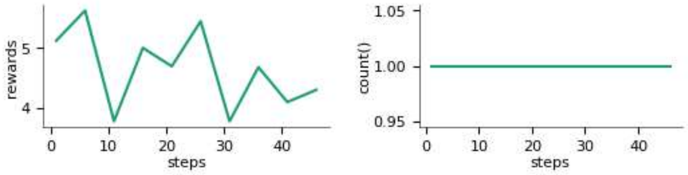
Distributions



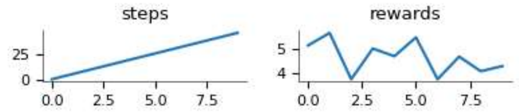
2-d distributions



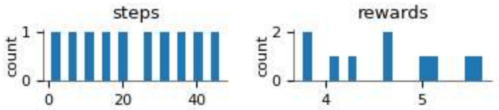
Time series



Values



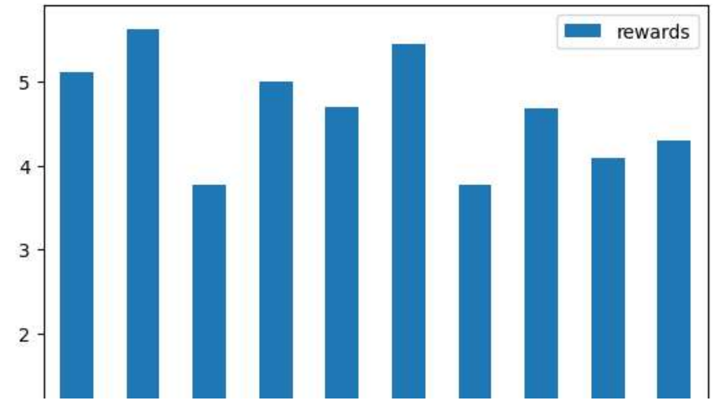
Distributions



2-d distributions

```
df.plot.bar(x="steps")
```

```
<Axes: xlabel='steps'>
```



```
import random  # Import the random module to generate random numbers.

class Environment:
    def __init__(self):
        self.steps_left = 10  # Initialize the number of steps left to 10.

    def get_observation(self):
        return [0.0, 0.0, 0.0, 0.0]  # Return a fixed observation list with four elements all set to 0.0.

    def get_actions(self):
        return [0, 1]  # Return a list of available actions, [0, 1] in this case.

    def is_done(self):
        return self.steps_left == 0  # Return True if there are no steps left (steps_left is 0), otherwise False.

    def action(self, action):
        if self.is_done():
            print("Game over")  # If the game is already over, print "Game over".

        self.steps_left -= 1  # Decrease the number of steps left by 1.
        return random.random()  # Return a random float between 0 and 1.

# The Environment class defines a simple environment with methods to get observations, actions, and perform actions.

class Agent:
    def __init__(self):
        self.total_reward = 0.0  # Initialize the total reward of the agent to 0.0.

    def steps(self, env):
        current_obs = env.get_observation()  # Get the current observation from the environment.
        actions = env.get_actions()  # Get the available actions from the environment.
        reward = env.action(random.choice(actions))  # Choose a random action, perform it in the environment, and get the reward.
        self.total_reward += reward  # Update the total reward of the agent with the obtained reward.

# The Agent class defines a simple agent that can interact with the environment and collect rewards.

# Instantiate the classes using constructors.
env = Environment()  # Create an instance of the Environment class.
agent = Agent()  # Create an instance of the Agent class.

while not env.is_done():  # Continue the loop until the environment is done (steps_left becomes 0).
    agent.steps(env)  # Agent takes a step in the environment.

# Print the total reward obtained by the agent after interacting with the environment.
print("The total reward is %f" % agent.total_reward)
```

```
The total reward is 4.793357
```

```
steps = []    # Create an empty list to store the values of 'i'.
rewards = []  # Create an empty list to store the total rewards obtained by the agent.

# Loop from 1 to 49 (inclusive), with a step size of 5.
for i in range(1, 50, 5):
    env = Environment()  # Create an instance of the Environment class.
    agent = Agent()      # Create an instance of the Agent class.

    # Continue the loop until the environment is done (i.e., all steps are used).
    while not env.is_done():
        agent.steps(env)  # Agent takes a step in the environment.

    steps.append(i)     # Add the value of 'i' to the 'steps' list.
    rewards.append(agent.total_reward)  # Add the total reward obtained by the agent to the 'rewards' list.
```

```
# The loop repeats for different values of 'i' (1, 6, 11, ..., 46).

# After the loop, the 'steps' list will contain [1, 6, 11, ..., 46], and the 'rewards' list will contain the total rewards obtained for e

# The lists 'steps' and 'rewards' will be used to analyze and visualize how the total reward varies with different values of 'i'.

import pandas as pd
df=pd.DataFrame({"steps":steps,"rewards":rewards
})
df
```

	steps	rewards
0	1	4.670371
1	6	5.798093
2	11	4.403257
3	16	4.876947
4	21	4.210805
5	26	5.158621
6	31	4.778753
7	36	4.997821
8	41	3.642197
9	46	4.416673

```
# Assignment 1:

# Define an environment and build a simple agent program to perform random actions. Initialize step count to 100.
# Compute the cumulative reward for the random actions performed by the agent.
# Plot the graph by varying steps in range of 2 and report your observations on actions and rewards.

import random
class Environment :
  def __init__(self):
    self.steps_left=100
  def get_observation(self):
    return[0.0,0.0,0.0,0.0]
  def get_actions(self):
    return[0,1]
  def is_done(self):
    return self.steps_left==0
  def action(self,action):
    if self.is_done():
      print("game over")
    self.steps_left-=1
    return random.random()
 # Instantiate the classess using constructors
env=Environment()
agent=Agent()
while not env.is_done():
  agent.steps(env)
print("the total reward is %d"% agent.total_reward)
```

```
     the total reward is 47
```

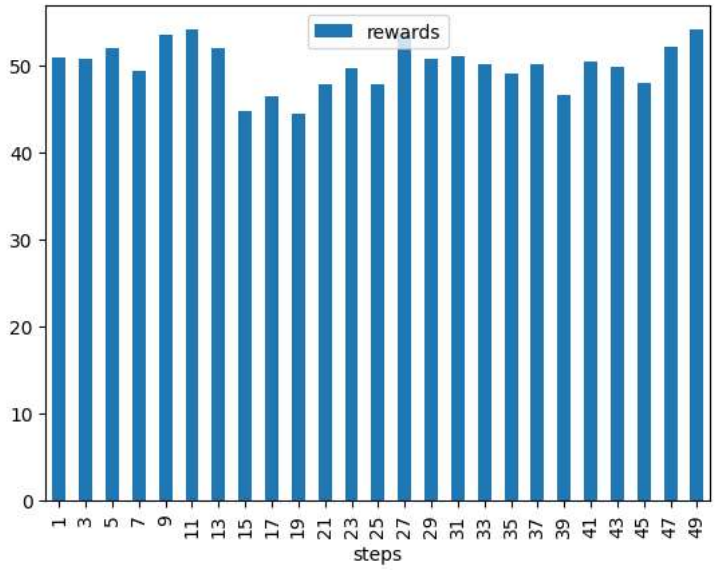
```
steps=[]
rewards=[]
for i in range(1,50,2):
  env=Environment()
  agent=Agent()
  while not env.is_done():
    agent.steps(env)
  steps.append(i)
  rewards.append(agent.total_reward)


import pandas as pd
df=pd.DataFrame({"steps":steps,"rewards":rewards
})
df
```

	steps	rewards
0	1	50.928338
1	3	50.779385
2	5	51.950746
3	7	49.423820
4	9	53.597799
5	11	54.161209
6	13	52.007782
7	15	44.739363
8	17	46.512329
9	19	44.483438
10	21	47.825037
11	23	49.701818
12	25	47.822594
13	27	53.783682
14	29	50.770503
15	31	51.085213
16	33	50.090026
17	35	49.090594
18	37	50.193106
19	39	46.655961
20	41	50.397844
21	43	49.904796

```
df.plot.bar(x="steps")
```

```
<Axes: xlabel='steps'>
```


rewards
steps