Case Study: Hacker Statistics

Friday, October 9, 2020 1:15 PM

Hacker Statistics

Datacamp

Simulating Using Rand()

- You can simulate a situation to answer complex hypothetical situations (many probabilities) using numpy's pseudo randomization tool, rand()

```
import numpy as np
np.random.seed(123)
coin = np.random.randint(0,2)  # Randomly generate 0 or 1
print(coin)
if coin == 0:
    print("heads")
else:
    print("tails")
```

```
0
heads
```

  - random.seed() is useful to reproduce the data given by a pseudo-random number generator. By re-using a seed value, we can regenerate the same data multiple times as multiple threads are not running.
  - When we supply a specific seed to the random generator, every time you execute a program, you will get the same numbers. That is useful when you need a predictable source of random numbers

Random Walk

- You can produce a list of the random results, but that's not a random walk. A random walk accumulates the steps for tails

```
import numpy as np
np.random.seed(123)
tails = [0]
for x in range(10) :
    coin = np.random.randint(0, 2)
    tails.append(tails[x] + coin)
print(tails)
```

```
[0, 0, 1, 1, 1, 1, 1, 2, 3, 3]
```

  - Notice how there's 11 items on the list since you initialized the list already with a 0

Distribution of Final Point in a random walk

- To answer the question, what is the chance of being past the 60th floor given a set of movement dictated by a dice roll
  - 1 or 2: go down a floor (-1)
  - 3 or 4 or 5: go up a floor (+1)
  - 6: roll the dice and go up that dice roll (+1, +2, +3, +4, +5, +6)
  - And a .1% chance of falling all the way back down to the 0th floor

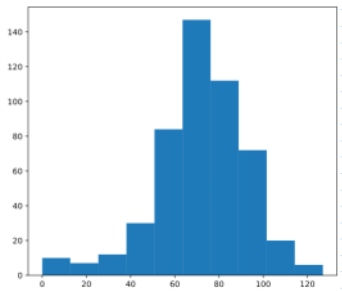
```
# numpy and matplotlib imported, seed set

# Simulate random walk 500 times
all_walks = []
for i in range(500) :
    random_walk = [0]
    for x in range(100) :
        step = random_walk[-1]
        dice = np.random.randint(1,7)
        if dice <= 2:
            step = max(0, step - 1)
        elif dice <= 5:
            step = step + 1
        else:
            step = step + np.random.randint(1,7)
        if np.random.rand() <= 0.001 :
            step = 0
        random_walk.append(step)
    all_walks.append(random_walk)

# Create and plot np_aw_t
np_aw_t = np.transpose(np.array(all_walks))

# Select last row from np_aw_t: ends
ends = np_aw_t[-1,:]

# Plot histogram of ends, display plot
plt.hist(ends)
plt.show()
```



  - This makes a 2D array of all the 500 simulated random step lists (i.e. step) of 100 steps!
  - You can store a final 1D array of the final value of each of the 500 walks and plot them on a histogram to help you answer the question: what are the odds of finishing at or above 60!

Ends represents the total list of final steps. To find the probability of winning (final value > 60):

Np.means(ends >= 60)

= 78.2% which is a pretty high chance of winning!

Jupyter Notebook

Generating a random value bw 0-1

```
# Import numpy as np
import numpy as np

# Set the seed
np.random.seed(123)

# Generate and print random float
print(np.random.rand())
```

Random Walk Algorithm

```
1  # Numpy is imported, seed is set
2
3  # Initialize random_walk
4  random_walk = [0]
5
6  for x in range(100) :
7      step = random_walk[-1]
8      dice = np.random.randint(1,7)
9
10     if dice <= 2:
11         # Replace below: use max to make sure step can't go below 0
12         step = max(0, step - 1)
13     elif dice <= 5:
14         step = step + 1
15     else:
16         step = step + np.random.randint(1,7)
17
18     random_walk.append(step)
19
20 print(random_walk)
```

  - On line 7, you set step to be the most recent value, which is to say you index into the last item
  - On line 12, use max function to make sure the random walk doesn't go below 0.

Displaying your random walk:

```
19 # Import matplotlib.pyplot as plt
20 import matplotlib.pyplot as plt
21
22 # Plot random_walk
23 plt.plot(random_walk)
24
25 # Show the plot
26 plt.show()
```