index

January 10, 2022

1 The Cumulative Distribution Function - Lab

1.1 Introduction

In the previous lesson, you learned how you can create a cumulative distribution function for discrete and continuous random variables. In this lab, you'll try to calculate a CDF for a dice roll yourself, and visualize it.

1.2 Objectives

You will be able to:

- Calculate CDF in Python for a given discrete variable with a limited set of possible values
- Visualize and inspect a CDF in order to make assumptions about the underlying data

1.3 Calculating CDF in Python

Recall the formula to calculate the cumulative probability from the previous lesson:

$$F(x) = P(X \leq x)$$

So given a list of all possible values of x, We can easily calculate the CDF for a given possible value X by performing the following steps:

- Build a function `calculate_cdf(lst,X)`, where `lst` is a list of all possible values in a discrete variable x (6 values for a dice roll), and X is the value for which we want to calculate the cumulative distribution function
- Initialize a variable called `count`
- For all values in `lst`, if a value is less than or equal to X, add one to `count` - do nothing otherwise. (this will tell us the total number of values less than X)
- Calculate the cumulative probability of X dividing `count` by the total number of possible values
- Round by 3 decimals and return the cumulative probability of X

[45]:
```
import collections
def calculate_cdf(lst, X):
    count = 0
    dic = collections.Counter(lst)
    for key in dic:
```

```
        count += dic[key]/len(lst)
        if key == X:
            break
    return count


# test data
test_lst = [1, 2, 3]
test_X = 2

calculate_cdf(test_lst, test_X)

# 0.667
```

[45]: 0.6666666666666666

[46]:
```
## From GitHub

# def calculate_cdf(lst, X):
#     count = 0
#     for value in lst:
#         if value <= X:
#             count += 1

#     cum_prob = count / len(lst) # normalizing cumulative probabilities (as␣
 ↪with pmfs)
#     return round(cum_prob, 3)

# # test data
# test_lst = [1,2,3]
# test_X = 2

# calculate_cdf(test_lst, test_X)
```

Now, use this function to calculate a CDF for each value in a dice roll so you can plot it later on.

Perform the following steps in the cell below: * Create a list `dice_lst` with all possible values of a fair dice * Initialize an empty list `dice_cum` for storing cumulative probabilities for these values. * For each value in the `dice_lst` calculate its cumulative probability using the function above and store in `dice_cum` list.

[48]:
```
dice_lst = list(range(1,7))
dice_cum = [calculate_cdf(dice_lst, i) for i in dice_lst]

dice_cum

# [0.167, 0.333, 0.5, 0.667, 0.833, 1.0]
```

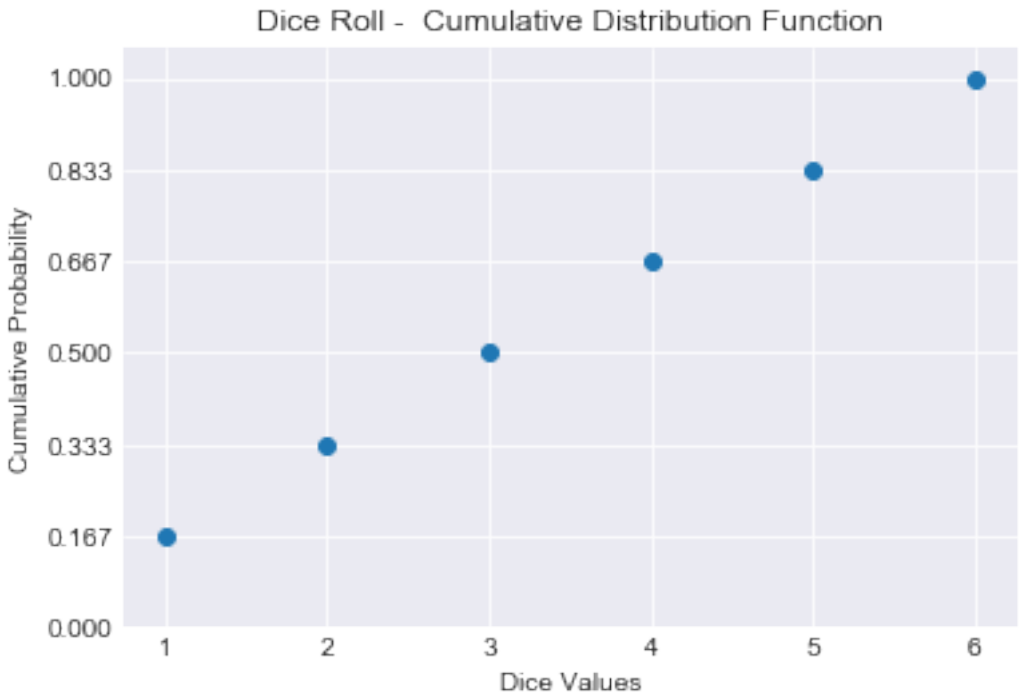
[48]:
```
[0.16666666666666666,
 0.3333333333333333,
 0.5,
 0.6666666666666666,
 0.8333333333333333,
 0.9999999999999999]
```

[49]:
```
# # From GitHub Solution

# dice_lst = [1,2,3,4,5,6]
# dice_cum = []
# for X in dice_lst:
#     dice_cum.append(calculate_cdf(dice_lst, X))
# dice_cum
```

> CDFs are implemented with two sorted lists: one list which contains the potential outcome values of your discrete distribution, and another list which contains cumulative probabilities.

Following this, we now have a list of possible values and a second list containing cumulative probabilities for each value. Let's go ahead and plot these values in matplotlib using a bar plot. * Use `dice_lst` for x-axis and `dice_cum` for y-axis

[54]:
```
# Your code here
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')

plt.bar(dice_lst, dice_cum, color = "tab:red", width = 0.3)
plt.xlabel("Dice Number")
plt.ylabel("Cumulative Probability")
plt.title("Cumulative Distribution Function For a Fair Dice")
plt.show();
```



1.4 Level Up (optional)

CDFs (and PMFs) can be calculated using built-in NumPy and matplotlib methods. So we don't have create custom functions to calculate these. We can draw a histogram styled CDF as shown below using the following steps

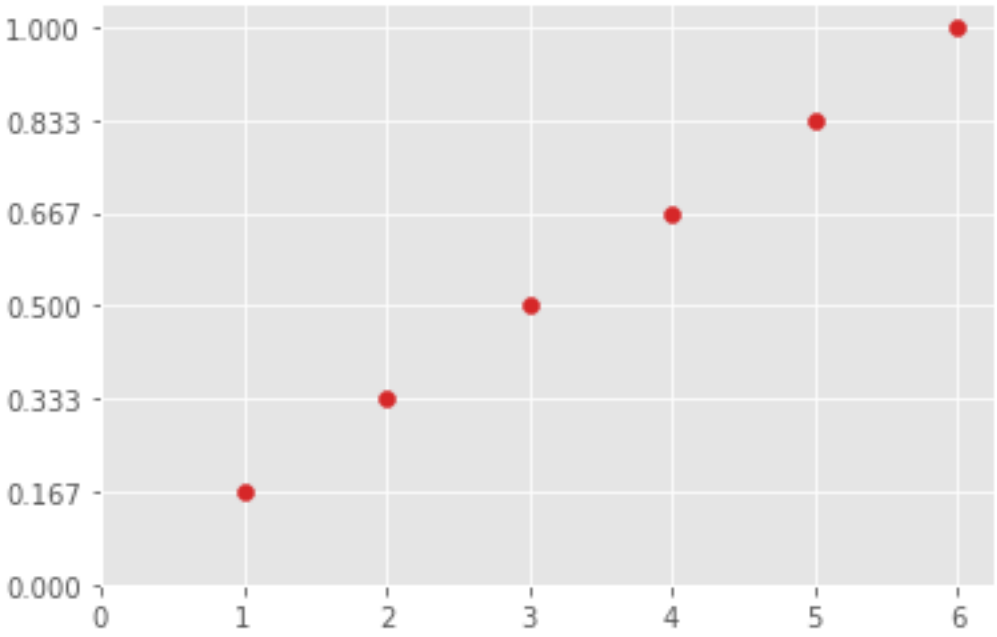
You would need to perform these steps * Use `np.histogram()` to automatically calculate the histogram with probabilities. Here is numpy histogram documentation to help you dig deeper.

- Use `plt.scatter()` method with `np.cumsum()` to calculate and plot cumulative probabilities (just like we did above).

[126]:
```
# Your code here
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')

data = [1, 2, 3, 4, 5, 6]
hist, bins = np.histogram(data, bins = 6, range = (1,7), normed = True)
hist_cumsum = np.cumsum(hist)
plt.scatter(data, hist_cumsum, color = "tab:red");
plt.yticks(np.linspace(0,1,num=7))
plt.xticks(np.linspace(0,6,num=7))
# plt.bar(data, hist_cumsum, color = "tab:red", width = 0.3);
```

```
plt.show()
```

```
<ipython-input-126-c717549d138d>:8: VisibleDeprecationWarning: Passing
`normed=True` on non-uniform bins has always been broken, and computes neither
the probability density function nor the probability mass function. The result
is only correct if the bins are uniform, when density=True will produce the same
result anyway. The argument will be removed in a future version of numpy.
  hist, bins = np.histogram(data, bins = 6, range = (1,7), normed = True)
```



1.5 Summary

In this lesson, we looked at developing a CDF - a percentile function of a discrete random variable. We looked at how to calculate and visualize a CDF. This technique can also be applied to continuous random variables which we shall see later in this section.