Random Variables and Statistical Inferences

Session-4

1

Agenda

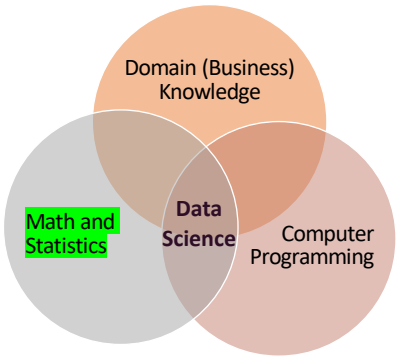
Lecture -4: Random Variables and Statistical Inferences

- Use Python to apply fundamental concepts of probability theory to examine random variables and quantify probability
- Use Python to draw a random sample
- Understand Probability Density and Mass Functions (PDF and PMF)

2

Where We Are Now

- For Data Science, you need
  - Business understanding,
  - Programming skills, and
  - Math and Statistics skills
- Today and in the next lecture we will focus on Statistics
- Python SciPy library is commonly used for statistics



3

Introduction to SciPy

- SciPy stands for Scientific Python.
- It's an optimized computation library that uses Numpy underneath.
- The same NumPy's creator Travis Olliphant created SciPy.
- It has useful utility functions that are frequently used in Data Science.
- SciPy is predominantly written in Python, but a few segments are written in C.

4

Statistical Significance Test With SciPy

- In Statistics, "Statistical Significance" means that the result that was produced can be explained with a reason behind it, it was not produced randomly, or by chance.
- SciPy provides us with a module called scipy.stats, which has functions for performing statistical significance tests.

Statistics and Probability

Probability Vs. Statistics

Statistics: Analyzing the frequency of **past** events

Probability: Predicting the likelihood of **future** events

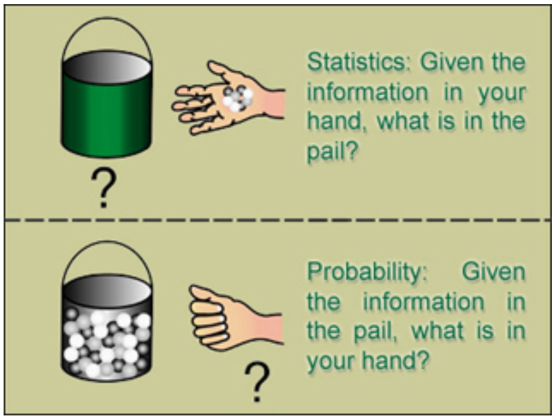


Diagram showing the difference between statistics and probability. (Image by MIT OpenCourseWare. Based on Gilbert, Norma. *Statistics.* W.B. Saunders Co., 1976.)

7

Statistics in Python

Think about a coin toss:

- The **sample space** contains **two events**:
  1. Flipping a heads, and
  2. Flipping a tails
- So, the probability of getting a head is one-half (½) or 50% as tossing a **fair coin** will have a 1-in-2 chance of being heads or tails.
- However, if we perform the experiment of flipping a coin, we may get two or three heads or tails consecutively which may look different from 50% probability.
- So, we need to repeat an experiment many times to get useful **statistics** to calculate probabilities that may match with the expected or **ideal** probabilities

8

Statistics in Python (Contd.)

- We can use Python **Random()** function to do this experimentation of coin flip
- The built-in function random() generates a random fraction number between 0 an 1 with equal probability of each number.
- As the 0.5 is the midpoint of 0 and 1, the numbers between 0 and 0.5 has a total of 50% probability, and, we can take this as the probability of getting a head of a coin toss.

```
import random
```

```
random.random() # Generates a number between 0 (inclusive) and 1 (exclusive)
```

Out[31]: 0.7029259239462168

Statistical Experiment in Python

[13]:
```
# A function to count the number of heads in 100 flips
def coin_toss():
    heads = 0
    for i in range(100):
        if random.random() < 0.5:
            heads +=1
    return heads
```

[14]:
```
# The expected value is 50 but a single run may not provide the value
coin_toss()
```

[14]: 51

Statistical Experiment in Python

```
def simulate(n):
    heads_in_trials = []
    for i in range(n):
        heads_in_trials.append(coin_toss())
    return(sum(heads_in_trials)/n)
```

```
# 10, 100, 1000, 10000, 100000 coin flips - what
is the average %
print("10 simulations: ", simulate(10))
print("100 simulations: ", simulate(100))
print("1000 simulations: ", simulate(1000))
print("10000 simulations: ", simulate(10000))
print("100000 simulations: ", simulate(100000))
```

As sample size increases, we get results closer to the ideal probability

Out[]:

10 simulations: 49.5

100 simulations: 49.77

1000 simulations: 49.839

10000 simulations: 50.0343

100000 simulations: 50.01029

Probability in Python

Let's use the coin toss example again:

- Tossing a coin has only two outcomes - a head or a tail
- The **binomial distribution** model deals with finding the probability of success of an event which has only two possible outcomes in a series of experiments. This model is useful to find the probability of exactly H # of heads when tossing a coin repeatedly for n times.
- In Python, we can use **binom** module from **scipy stats** package to generate the binomial distribution

```
from scipy.stats import binom
```

- **numpy random.binomial** module is also useful to draw samples from a binomial distribution

```
>>> n, p = 10, .5  # number of trials, probability of each trial
>>> s = np.random.binomial(n, p, 1000)
# result of flipping a coin 10 times, tested 1000 times.
```

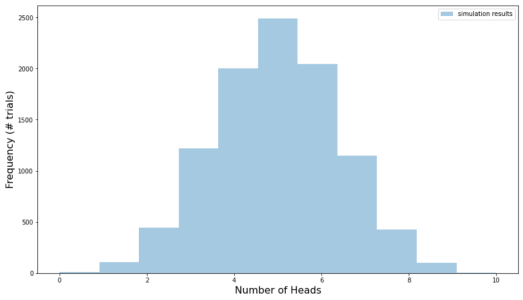
Probability in Python (Cont.)

Before using binom module, let's see how this can be done just with Python code

```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Function that runs our coin toss trials
# heads is a list of the number of successes from each trial of # tosses
def run_simulated_coin_toss(trials, num_tosses):
    heads = []
    for i in range(trials):
        tosses = [np.random.random() for i in range(num_tosses)]
        heads.append(len([i for i in tosses if i>=0.50]))
    return heads
```

```
# Let's run this with high number of trials
heads = run_simulated_coin_toss(10000, 10)
# Plot the results as a histogram
```



13

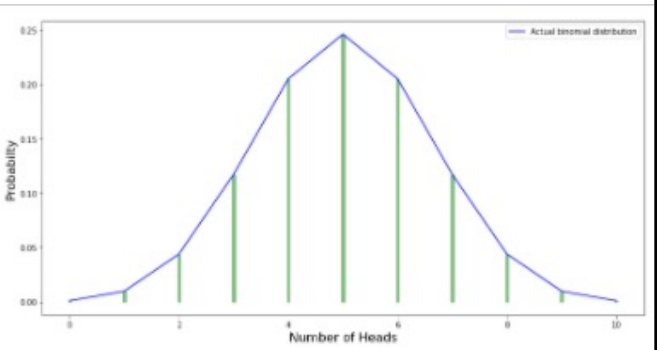
Binom module from scipy stats package

```
from scipy.stats import binom
x = range(0,11)# range is 0 to 10
n = 10  # the number of trials
p = 0.5 # the probability of a heads
```

Probability distribution of getting 0 to 10 heads (x), after 10 throws (n) with a fair coin (p=0.5)

```
# Plot the actual binomial distribution as a sanity check

# Plot the results as a histogram
fig, ax1 = plt.subplots(figsize=(14,7))
# ax1 = sns.distplot(heads, bins=11, label='simulation results')
ax1.set_xlabel("Number of Heads",fontsize=16)
ax1.set_ylabel("Probabilty",fontsize=16)
# draw the actual binomal distribution
ax1.plot(x, binom.pmf(x, n, p), 'b', label='Actual binomial distribution')
ax1.vlines(x, 0, binom.pmf(x, n, p), colors='g', lw=5, alpha=0.5)
plt.legend()
plt.show()
```



14

Quiz

What will this code block will do if *np.random.binomial(n, p, size=runs)* returns a series of number between 0 and 10 (e.g. n=10, p=0.5, size=1000) ?

```
sum([1 for i in np.random.binomial(n, p, size=runs)
     if i==7])/runs
```

a. Sum of all 7s in this Binomial distribution
b. The probability of the occurrence of 7 in this Binomial distribution
c. The probability of the occurrence of 1 in this Binomial distribution
d. All of the above

15

Sampling

16

Sampling : What and Why

- **Sampling** is the process by which **inference** is made by examining only a part of the whole population or set.

- Purpose of Sampling:
  - In most cases, we cannot gather data from the entire population.
  - Even it is possible, an organization will not have time and money to collect data for the entire population
  - **Sampling** saves money by allowing data scientists to gather the same or similar answers from a **sample** that they **would** receive from the population.

17

Sampling Process

- **Probability sampling** is based on the fact that every member of a population has a known and equal chance of being selected. For example, if you had a population of 100 people, each person would have odds of 1 out of 100 of being chosen.

- With **non-probability sampling**, those odds are not equal. For example, a person might have a better chance of being chosen if they live close to the researcher or have access to a computer.

- **Probability sampling gives you the best chance to create a sample that is truly representative of the population.**

- **Simple Random Sample:** Every member and set of members has an equal chance of being included in the sample.

18

Random Samples

Random samples are usually fairly representative since they don't favor certain members.

Syntax: random.sample(population, k)

The population can be any sequence or set from which you want to select a k length numbers.

```
# Select items from a list
import random

print("random module's sample in Python ")
list = [20,40,80,100,120]
print ("choosing 3 random items from a list using sample function",
       random.sample(list,k=3))

list = [20,40,20,20,20]
print ("choosing 3 random items from a list using sample function",
       random.sample(list,k=3))
```

```
random module's sample in Python
choosing 3 random items from a list using sample function [20, 100, 80]
choosing 3 random items from a list using sample function [40, 20, 20]
```


Random samples (Contd.)

```
# select items from a set
weight_set = {40, 50, 55, 65, 75,80}
print ("choosing 4 random items from a set using sample method ",
       random.sample(weight_set, k=4))
```

choosing 4 random items from a set using sample method [40, 65, 75, 80]

```
# select items from a dictionary
marks_dict = {
  "Kelly": 55,
  "jhon": 70,
  "Donald": 60,
  "Lennin" :50
}
print ("choosing 2 random items from a dictionary using sample method ",
       random.sample(marks_dict.items(), k=2))
```

choosing 2 random items from a dictionary using sample method [('Lennin', 50), ('Kelly', 55)]

Stratified Random Sample

A stratified sample is one that ensures that subgroups (strata) of a given population are each adequately represented within the whole sample population of a research study.

Example—A student council surveys 100 students by getting random samples of 25 freshmen, 25 sophomores, 25 juniors, and 25 seniors.

Why It is Needed

A stratified sample guarantees that members from each group will be represented in the sample, so this sampling method is good when we want some members from every group.

21

Example - sklearn.model_selection.StratifiedShuffleSplit

```
>>> import numpy as np
>>> from sklearn.model_selection import StratifiedShuffleSplit
>>> X = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [1, 2], [3, 4]])
>>> y = np.array([0, 0, 0, 1, 1, 1])
>>> sss = StratifiedShuffleSplit(n_splits=5, test_size=0.5, random_state=0)
>>> sss.get_n_splits(X, y)
5
>>> print(sss)
StratifiedShuffleSplit(n_splits=5, random_state=0, ...)
>>> for train_index, test_index in sss.split(X, y):
...     print("TRAIN:", train_index, "TEST:", test_index)
...     X_train, X_test = X[train_index], X[test_index]
...     y_train, y_test = y[train_index], y[test_index]
TRAIN: [5 2 3] TEST: [4 1 0]
TRAIN: [5 1 4] TEST: [0 2 3]
TRAIN: [5 0 2] TEST: [4 3 1]
TRAIN: [4 1 0] TEST: [2 3 5]
TRAIN: [0 5 1] TEST: [3 4 2]
```


22

Random Variables

23

Random Variables

Random variable - a numerical description of the outcome of a statistical experiment

- **Discrete Random Variable:** A discrete random variable is one which may take on only a countable number of distinct values (or an infinite sequence of values)

  Example : The number of eggs that a hen lays in a given day

- **Continuous random variable: N**umeric **variables** that have an infinite number of values between any two positive values. For example, between 3 and 4 we can have infinite number of values like 3.1, 3.112, 3.1112 ……

  Example: We can define a continuous random variable X to be the height of students in a class.

  Since the continuous random variable is defined over an **interval** of values, it is **represented by the area under a curve (or the integral)**.

24

Quiz

A random variable X to be the number which comes up when you roll a fair dice. X can take values : [1,2,3,4,5,6]

Are these discrete or continuous random variables?

a. Continuous

b. Discrete

25

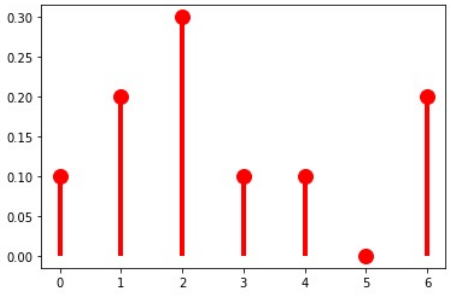
Discrete Random Variables (Optional)

rv_discrete - A base class to construct specific distribution classes and instances for discrete random variables.

Creating a discrete distribution from a list of probabilities

```
import numpy as np
from scipy import stats
xk = np.arange(7)
pk = (0.1, 0.2, 0.3, 0.1, 0.1, 0.0, 0.2)
custm = stats.rv_discrete(name='custm', values=(xk, pk))

import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1)
ax.plot(xk, custm.pmf(xk), 'ro', ms=12, mec='r')
ax.vlines(xk, 0, custm.pmf(xk), colors='r', lw=4)
plt.show()
```



26

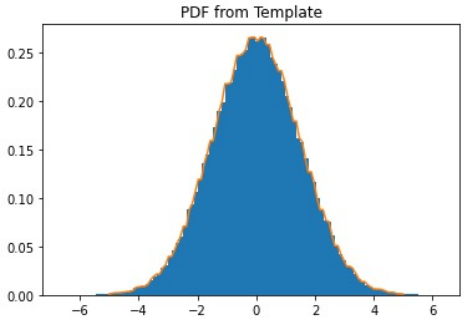
Continuous Random variables (Optional)

rv_continuous - a base class to construct specific distribution classes for continuous random variables

Python Code for Continuous Random Variable Distribution with a Histogram

```
import scipy.stats
data = scipy.stats.norm.rvs(size=100000, loc=0, scale=1.5, ran
dom_state=123)
hist = np.histogram(data, bins=100)
hist_dist = scipy.stats.rv_histogram(hist)

# this will behave according to the rv_continuous distribution
hist_dist.pdf(x)
X = np.linspace(-5.0, 5.0, 100)
plt.title("PDF from Template")
plt.hist(data, density=True, bins=100)
plt.plot(X, hist_dist.pdf(X), label='PDF')
plt.show()
```



27

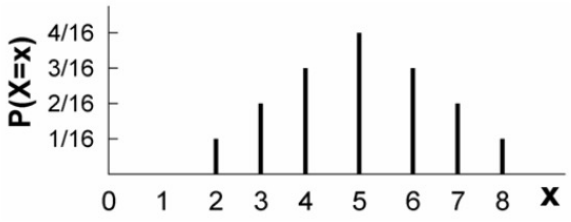
Probability Density and Mass Functions

28

Probability Mass Function

Probability mass function (PMF) gives the probability that a discrete random variable is exactly equal to some value.

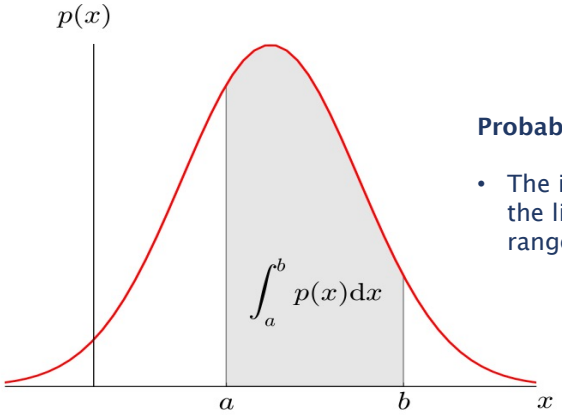
x	P(x)
2	1/16
3	2/16
4	3/16
5	4/16
6	3/16
7	2/16
8	1/16



29

Probability Density Function

Probability density function (PDF) describes the relative likelihood for a continuous random variable (X) to take on a given value (x).



Probability Density Function (PDF) of a value x

- The integral over the PDF between a and b gives the likelihood of finding the value of x in that range.

30

Quiz

The probability of a specific value of a continuous random variable will be zero or almost zero.

a. True

b. False

31

Thank You

32