tasks2020

December 18, 2020

1 Higher Diploma in Science in Computing (Data Analytics)

1.1 #### Programme Module: Fundamentals of Data Analysis (COMP07084)

1.2 Task 1 - counts

Write a Python function called counts that takes a list as input and returns a dictionary of unique items in the list as keys and the number of times each item appears as values. So, the input [‘A’,‘A’,‘B’,‘C’,‘A’] should have output {‘A’: 3,‘B’: 1,‘C’: 1}.

Your code should not depend on any module from the standard library or otherwise. You should research he task first and include a description with references of your algorithm in the notebook.

Addendum: We were also asked to see if we could create a solution that could could use not just strings in the input but also integers or floats.

Clarification: You cannot use any part of the standard library that has to be imported. The Python documentation technically includes the built-in, non-language features in the standard library - even though they do not need to be imported: https://docs.python.org/3/library/. You can use any of these built-in features - just not anything you need to import.

1.3 1. Solution

In order to solve this task I did some initial research looking for sample code that would provide a solution. After a few hours I had sourced five code samples before deciding on one from geeksforgeeks.org [2].

I modified the input to check how the function would deal with integers and floats as well. Initally the code failed, this I found out was due to a format issue.

In Python the `%` operator is used to format a set of variables enclosed in a “tuple” (a fixed size list), together with a format string, which contains normal text together with “argument specifiers”,

special symbols like %s and %d. So in this case I had to change the original code from %d to %s. [5]

[1]:
```
# Define the function
def counts(my_list):

    # Create a blank dictionary
    freq = {}

# Create for loop
    for item in my_list:
        if (item in freq):
            freq[item] += 1
        else:
            freq[item] = 1

# Iterate over key/value pairs in dictionary and print them
    for key, value in freq.items():
        print("% s : % s" % (key, value))


# List of items to count
my_list = ['A', 'A', 'B', 'C', 'A', 1, 2, 3, 2, 'python', 4.5, 6.5, 'B']
```

Lets have a look at the output of the function.

[2]:
```
counts(my_list)
```

```
A : 3
B : 2
C : 1
1 : 1
2 : 2
3 : 1
python : 1
4.5 : 1
6.5 : 1
```

1.4 2. Explanation

2.1 Function I've outlined below how the algorithm works and broken it in to sections for ease of use. First up was to define the function.

```
def counts(my_list):
```

A function in Python is defined using the `def` keyword. A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result. In this example the function is called `counts`. Information can be passed into functions as arguments. Arguments are specified after the function name, inside the

parentheses. You can add as many arguments as you want, just separate them with a comma. In this function we have one argument called `(my_list)`.

Additional information: A function definition defines a user-defined function object. If you are looking for more detail on functions you can run the help('def') statement in your own Jupyter notebook.

2.2 Dictionary The code also creates a dictionary called `freq`. A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

```
freq = {}
```

It is best to think of a dictionary as a set of key: value pairs, with the requirement that the keys are unique (within one dictionary). Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary; this is also the way dictionaries are written on output.[6]

Additional information: If you are looking for more detail on dictionaries(dicts) you can run the help('dict') statement in your own Jupyter notebook.

2.3 For Loop Next up the code uses a for loop to iterate over each item in the list. At each iteration, check if item is in an initially empty dictionary. If it is, use dictionary `item` += 1 to add 1 to the value to which item maps. Otherwise, use dictionary `item` = 1 to add item to dictionary with the value 1. The `items()` method is used to return the list with all dictionary keys with values. [8]

```
for item in my_list:
        if (item in freq):
            freq[item] += 1
        else:
            freq[item] = 1
```

Additional information: The "for" statement is used to iterate over the elements of a sequence (such as a string, tuple or list) or other iterable object. If you are looking for more detail on for loops you can run the help('for') statement in your own Jupyter notebook.

2.4 Keys and Values The next section captures the items as the keys of the dictionary `freq` and their frequencies as the values.

```
for key, value in freq.items():
    print ("% s : % s"%(key, value))
```

2.5 List This section of code contains the list of items to count in the function.

```
my_list = ['A', 'A', 'B', 'C', 'A', 1, 2, 3, 2, 'python', 4.5, 6.5, 'B']
```

Additional information: If you are looking for more detail on lists you can run the help('list') statement in your own Jupyter notebook.

1.5 3. References

[1] GeeksforGeeks. 2020. Python | *Get Unique Values From List Of Dictionary - Geeksforgeeks*. [online] Available at: https://www.geeksforgeeks.org/python-get-unique-values-from-list-of-dictionary/ [Accessed 7 October 2020].

[2] GeeksforGeeks. 2020. *Counting The Frequencies In A List Using Dictionary In Python - Geeksforgeeks*. [online] Available at: https://www.geeksforgeeks.org/counting-the-frequencies-in-a-list-using-dictionary-in-python/ [Accessed 6 October 2020].

[3] Elance, P., 2019. *Counting The Frequencies In A List Using Dictionary In Python*. [online] Tutorialspoint.com. Available at: https://www.tutorialspoint.com/counting-the-frequencies-in-a-list-using-dictionary-in-python [Accessed 6 October 2020].

[4] Techie Delight. 2020. *Count Occurrences Of An Item In A Python List - Techie Delight*. [online] Available at: https://www.techiedelight.com/count-occurrences-of-an-item-in-a-python-list/ [Accessed 7 October 2020].

[5] Learnpython.org. 2020. *String Formatting - Learn Python - Free Interactive Python Tutorial*. [online] Available at: https://www.learnpython.org/en/String_Formatting [Accessed 8 October 2020].

[6] W3schools.com. 2020. *Python Dictionaries*. [online] Available at: https://www.w3schools.com/python/python_dictionaries.asp [Accessed 11 October 2020].

[7] Docs.python.org. 2020. 5. *Data Structures — Python 3.9.0 Documentation*. [online] Available at: https://docs.python.org/3/tutorial/datastructures.html [Accessed 11 October 2020].

[8] Kite.com. 2020. *Code Faster With Line-Of-Code Completions, Cloudless Processing*. [online] Available at: https://www.kite.com/python/answers/how-to-count-item-frequency-in-python [Accessed 10 October 2020].

[9] GeeksforGeeks. 2020. *Python Dictionary | Items() Method - Geeksforgeeks*. [online] Available at: https://www.geeksforgeeks.org/python-dictionary-items-method/ [Accessed 11 October 2020].

[10] Edureka. 2020. *What Is The Main Function In Python And How To Use It | Edureka*. [online] Available at: https://www.edureka.co/blog/python-main-function/ [Accessed 11 October 2020].

[11] Cite This For Me. 2020. *Save Time And Improve Your Marks With Citethisforme, The No. 1 Citation Tool*. [online] Available at: https://www.citethisforme.com/ [Accessed 7 October 2020].

[12] Okada, S., 2020. *7 Essential Tips For Writing With Jupyter Notebook*. [online] Medium. Available at: https://towardsdatascience.com/7-essential-tips-for-writing-with-jupyter-notebook-60972a1a8901#7aff [Accessed 11 October 2020].

[13] Pythontutor.com. 2020. *Python Tutor - Visualize Python, Java, Javascript, C, C++, Ruby Code Execution*. [online] Available at: http://www.pythontutor.com/visualize.html#mode=display [Accessed 7 October 2020].

1.6 Task 2 - dicerolls

Write a Python function called dicerolls that simulates rolling dice. Your function should take two parameters: the number of dice `k` and the number of times to roll the dice `n`. The function should simulate randomly rolling `k` dice `n` times, keeping track of each total face value. It should then return a dictionary with the number of times each possible total face value occurred. So, calling the function as diceroll `(k=2, n=1000)` should return a dictionary like: `{2:19,3:50,4:82,5:112,6:135,7:174,8:133,9:114,10:75,11:70,12:36}` You can use any module from the Python standard library you wish and you should include a description with references of your algorithm in the notebook.

1.7 1. Solution

[3]:
```
# Import numpy library
import numpy as np

# Use numpy random number generator
rng = np.random.default_rng(1)

# Define the function


def dicerolls():

    # create a blank dictionary
    diceroll = {}

# Create for loop
    for i in range(1000):
        d1 = rng.integers(1, 7)
        d2 = rng.integers(1, 7)
        diceroll[d1+d2] = diceroll.setdefault(d1+d2, 0)+1

# Iterate over key/value pairs in dictionary and print them
    for roll, count in diceroll.items():
        print("% s : % s" % (roll, count))
```

Lets have a look at the output of the function.

[4]:
```
dicerolls()
```

```
7 : 153
11 : 69
2 : 29
4 : 91
```

```
9 : 118
5 : 112
8 : 145
10 : 76
3 : 50
6 : 120
12 : 37
```

1.8 2. Explanation

2.1 Numpy library and random number generator I've outlined below how the algorithm works and broken it in to sections for ease of use. First up was to import numpy and the random number generator (RNG). This will be used to simulate the dicerolls in the function.

`python import numpy as np rng = np.random.default_rng(1)`

Additional information: If you are looking for more detail on random number generation in NumPy you can enter the following statement np.info(rng) statement in your own Jupyter notebook.

2.2 Function In Task 1 I covered what a function is, so I won't repeat that here. For this task I start by defining the function `dicerolls` unlike in Task 1 there are no arguments.

`python def dicerolls():`

2.3 For Loop Next up the code uses a for loop to iterate over each item in the list. At each iteration, check if item is in an initially empty dictionary. The `setdefault` method is used for setting defaults while or after filling the dictionary. This will store the results of the simulate dice rolls.

```
for i in range(1000):
        d1=rng.integers(1,7)
        d2=rng.integers(1,7)
        diceroll[d1+d2] = diceroll.setdefault(d1+d2,0)+1
```

Additional information: If you are looking for more detail on dictionaries(dicts) and (setdefault) you can run the help('dict') statement in your own Jupyter notebook.

2.4 Keys and Values The next section captures the items as the keys of the dictionary `diceroll` and their frequencies as the values.

```
    for roll, count in diceroll.items():
        print("% s : % s" % (roll, count))
```

1.9 3. References

[1] Data Science Unlimited. 2020. *Step By Step: Coding A Dice Roll Simulator In Python / Examples For Beginners.* [online] Available at: https://datascienceunlimited.tech/step-by-step-coding-a-dice-roll-simulator-in-python/ [Accessed 4 November 2020].

[2] Medium. 2020. *How To Simulate A Dice Roll And Guess The Result In Python*. [online] Available at: https://medium.com/an-amygdala/how-to-simulate-a-dice-roll-and-guess-the-result-in-python-9785079af6f3 [Accessed 4 November 2020].

[3] Codegrepper.com. 2020. *Dice Rolling Simulator Python Code Example*. [online] Available at: https://www.codegrepper.com/code-examples/python/dice+rolling+simulator+python [Accessed 5 November 2020].

[4] Pythonfiddle.com. 2020. *Mike's Dice Stats | Python Fiddle*. [online] Available at: http://pythonfiddle.com/mikes-dice-stats/ [Accessed 5 November 2020].

[5] Pieters, M., 2020. *Simulating Rolling 2 Dice In Python*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/33069476/simulating-rolling-2-dice-in-python [Accessed 5 November 2020].

[6] Lin, S., 2020. *Program That Simulates Rolling A Dice, And Tells You How Many Times You Roll Each Number*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/27220489/program-that-simulates-rolling-a-dice-and-tells-you-how-many-times-you-roll-eac [Accessed 6 November 2020].

[7] Hplgit.github.io. 2020. *Random Numbers And Simple Games*. [online] Available at: http://hplgit.github.io/primer.html/doc/pub/random/._random-readable004.html [Accessed 6 November 2020].

[8] Darklinkdigital.com. 2020. *Python Dice Functions*. [online] Available at: http://www.darklinkdigital.com/py-dice-functions.html [Accessed 6 November 2020].

[9] Iditect.com. 2020. *Python - Rolled 2 6-Side Die 1000 Times*. [online] Available at: https://www.iditect.com/how-to/55151873.html [Accessed 5 November 2020].

[10] BBC Bitesize. 2020. *Functions In Python - Procedures And Functions - KS3 Computer Science Revision - BBC Bitesize*. [online] Available at: https://www.bbc.co.uk/bitesize/guides/zqh49j6/revision/6 [Accessed 5 November 2020].

[11] Stack Overflow. 2020. *Rolling 2 Dice 1000 Times And Counting The Number Of Times The Sum Of The Two Dice Hit*. [online] Available at: https://stackoverflow.com/questions/60343980/rolling-2-dice-1000-times-and-counting-the-number-of-times-the-sum-of-the-two-di [Accessed 5 November 2020].

1.10 Task 3 - numpy.random.binomial

The `numpy.random.binomial` function can be used to simulate flipping a coin with a 50/50 chance of heads or tails. Interestingly, if a coin is flipped many times then the number of heads is well approximated by a bell-shaped curve. For instance, if we flip a coin 100 times in a row the chance of getting 50 heads is relatively high, the chances of getting 0 or 100 heads is relatively low, and the chances of getting any other number of heads decreases as you move away from 50 in either direction towards 0 or 100.

Write some python code that simulates flipping a coin 100 times. Then run this code 1,000 times, keeping track of the number of heads in each of the 1,000 simulations. Select an appropriate plot to depict the resulting list of 1,000 numbers, showing that it roughly follows a bell-shaped curve. You should explain your work in a Markdown cell above the code.

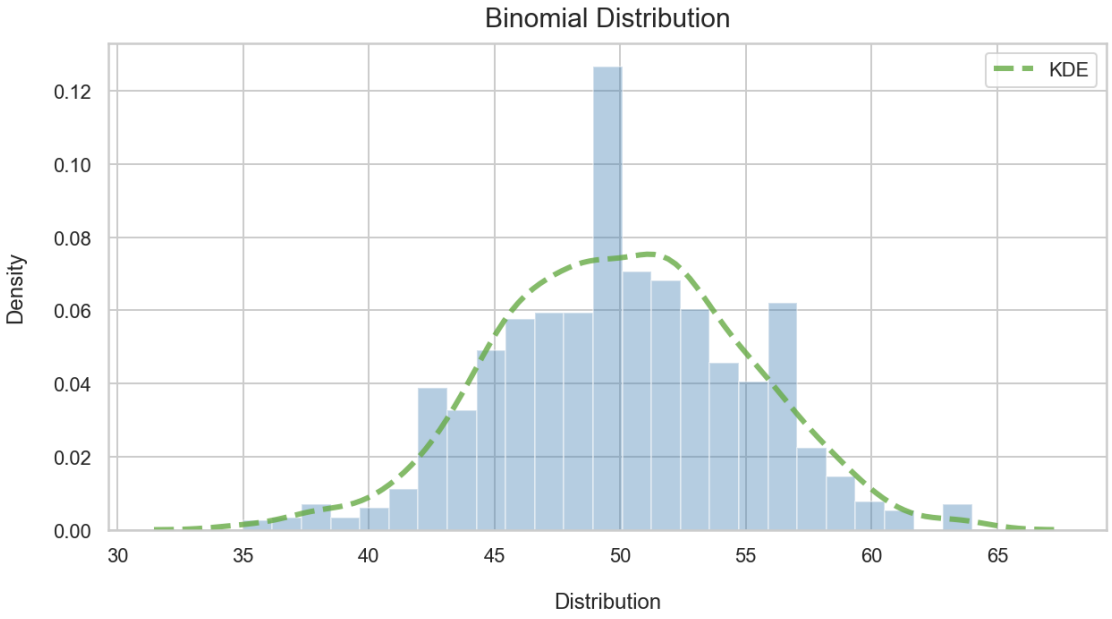
1.11 1. Solution

[5]:
```
# Import the required libraries
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Graphics in retina format are more sharp and legible.
%config InlineBackend.figure_format = 'retina'

# Sets the backend of matplotlib to the 'inline' backend.
%matplotlib inline

# Number of coin toss
n = 100

# Set the probability
p = 0.5

# Number of experiments
s = 1000

# Generate the results
x = np.random.default_rng(1).binomial(n, p, s)

# View the outcome
x
```

[5]:
```
array([45, 52, 57, 46, 50, 52, 49, 52, 54, 59, 45, 45, 44, 46, 42, 47, 46,
       47, 53, 39, 55, 58, 49, 49, 47, 55, 57, 57, 54, 52, 57, 51, 47, 51,
       40, 57, 45, 54, 48, 52, 54, 53, 55, 50, 53, 51, 47, 53, 49, 45, 48,
       46, 50, 47, 49, 44, 49, 45, 54, 53, 46, 46, 48, 50, 47, 53, 51, 46,
       41, 48, 55, 43, 49, 45, 49, 47, 54, 49, 49, 50, 59, 49, 47, 53, 55,
       48, 52, 50, 50, 51, 56, 57, 44, 54, 49, 49, 57, 51, 42, 44, 53, 43,
       50, 50, 51, 53, 58, 52, 49, 49, 51, 53, 56, 49, 51, 50, 56, 48, 54,
       48, 52, 47, 47, 44, 49, 52, 43, 58, 49, 53, 51, 51, 44, 49, 50, 52,
       54, 49, 55, 53, 59, 52, 51, 55, 48, 50, 52, 51, 52, 45, 52, 37, 49,
       48, 47, 50, 37, 45, 47, 53, 49, 47, 49, 59, 56, 41, 53, 55, 49, 49,
       51, 49, 49, 58, 47, 49, 48, 56, 54, 53, 41, 54, 49, 45, 52, 51, 53,
       48, 44, 59, 53, 46, 50, 55, 53, 51, 45, 44, 45, 43, 51, 53, 63, 41,
       43, 57, 52, 53, 54, 53, 56, 41, 60, 44, 47, 39, 48, 54, 51, 49, 45,
```

```
 50, 47, 43, 47, 48, 54, 51, 47, 52, 64, 52, 49, 48, 52, 51, 52, 56,
 56, 48, 45, 56, 48, 50, 50, 43, 54, 56, 52, 53, 41, 39, 50, 46, 56,
 55, 57, 47, 48, 57, 49, 57, 56, 51, 50, 44, 47, 47, 59, 50, 53, 45,
 56, 47, 56, 52, 55, 60, 44, 46, 44, 59, 43, 56, 46, 56, 46, 48, 52,
 46, 58, 51, 50, 44, 49, 51, 58, 48, 53, 57, 58, 53, 55, 49, 38, 44,
 52, 52, 49, 51, 61, 49, 56, 45, 46, 51, 50, 50, 52, 43, 56, 47, 56,
 57, 49, 60, 52, 57, 49, 51, 54, 54, 63, 53, 45, 48, 46, 48, 38, 51,
 50, 55, 55, 47, 41, 42, 51, 51, 55, 50, 48, 54, 50, 54, 38, 50, 54,
 46, 40, 55, 57, 44, 48, 49, 48, 52, 49, 46, 51, 43, 53, 50, 54, 53,
 60, 49, 51, 52, 47, 47, 53, 52, 52, 41, 49, 51, 47, 50, 53, 53, 53,
 48, 49, 60, 43, 52, 56, 46, 41, 52, 51, 51, 58, 45, 57, 44, 46, 63,
 53, 45, 56, 45, 54, 50, 48, 45, 64, 58, 44, 53, 38, 35, 42, 46, 46,
 58, 52, 49, 53, 56, 52, 50, 55, 52, 54, 40, 55, 59, 50, 59, 55, 52,
 55, 45, 53, 55, 56, 52, 46, 42, 41, 45, 46, 51, 44, 53, 45, 50, 56,
 54, 55, 55, 49, 48, 53, 50, 51, 55, 52, 48, 44, 53, 58, 59, 54, 48,
 46, 56, 52, 61, 42, 46, 51, 58, 53, 46, 50, 48, 53, 43, 48, 48, 49,
 51, 54, 45, 51, 50, 56, 44, 54, 53, 53, 59, 47, 53, 51, 50, 46, 47,
 46, 52, 46, 46, 45, 51, 51, 48, 53, 51, 50, 51, 56, 45, 54, 55, 51,
 51, 47, 50, 47, 42, 50, 45, 49, 48, 55, 52, 48, 47, 51, 45, 50, 52,
 56, 52, 49, 49, 47, 64, 55, 54, 44, 47, 52, 48, 51, 46, 43, 47, 52,
 54, 44, 55, 51, 50, 47, 51, 49, 47, 45, 53, 51, 47, 56, 49, 49, 53,
 46, 55, 53, 54, 48, 48, 45, 42, 54, 48, 54, 60, 49, 57, 48, 50, 49,
 55, 46, 51, 46, 52, 51, 46, 45, 53, 35, 57, 55, 44, 49, 60, 44, 57,
 51, 45, 50, 45, 58, 46, 53, 46, 52, 47, 51, 51, 53, 45, 61, 50, 52,
 53, 49, 58, 56, 47, 47, 53, 46, 52, 46, 46, 53, 46, 63, 52, 47, 54,
 38, 43, 42, 51, 50, 42, 45, 53, 55, 48, 43, 57, 46, 43, 47, 43, 56,
 49, 47, 52, 44, 52, 45, 50, 56, 55, 51, 52, 56, 45, 49, 46, 54, 40,
 48, 46, 44, 43, 46, 48, 43, 56, 53, 52, 58, 46, 49, 51, 49, 47, 52,
 44, 51, 45, 50, 43, 51, 47, 52, 49, 55, 51, 41, 46, 53, 52, 52, 51,
 42, 59, 52, 52, 42, 56, 58, 50, 45, 55, 51, 45, 52, 53, 51, 48, 44,
 56, 47, 48, 46, 49, 52, 47, 60, 59, 50, 44, 45, 49, 47, 54, 56, 55,
 55, 53, 47, 48, 43, 49, 46, 58, 49, 58, 49, 43, 46, 59, 54, 46, 54,
 46, 56, 61, 52, 49, 52, 42, 54, 52, 61, 42, 46, 44, 49, 52, 51, 54,
 40, 56, 57, 57, 51, 42, 52, 46, 52, 59, 56, 44, 48, 48, 58, 53, 45,
 56, 50, 58, 51, 54, 58, 48, 42, 46, 54, 47, 55, 57, 51, 45, 49, 49,
 51, 45, 49, 58, 38, 48, 49, 49, 48, 47, 48, 55, 58, 53, 49, 47, 44,
 43, 43, 48, 54, 51, 44, 58, 58, 55, 46, 55, 49, 51, 54, 56, 47, 44,
 46, 46, 45, 47, 49, 48, 48, 55, 46, 50, 44, 55, 54, 51, 52, 60, 57,
 38, 49, 45, 54, 49, 50, 55, 45, 61, 47, 48, 52, 41, 59, 53, 46, 38,
 51, 44, 51, 57, 49, 48, 44, 50, 53, 50, 37, 53, 47, 49, 47, 57, 46,
 46, 48, 56, 52, 35, 45, 45, 50, 48, 41, 56, 46, 45, 52, 48, 47, 55,
 45, 59, 48, 48, 42, 52, 53, 54, 51, 57, 50, 48, 40, 53, 48, 52, 49,
 43, 56, 40, 45, 54, 48, 51, 52, 63, 39, 55, 48, 52, 54, 49, 46, 45,
 53, 50, 53, 45, 46, 48, 47, 54, 50, 43, 46, 54, 54, 54, 47, 50, 45,
 50, 47, 37, 53, 51, 45, 46, 50, 53, 50, 49, 49, 58, 47, 47, 48, 43,
 47, 51, 43, 52, 51, 52, 47, 48, 51, 51, 47, 55, 44, 47],
dtype=int64)
```

[6]:
```
# Visualisation styling code
sns.set(rc={'figure.figsize': (10, 5)})
sns.set_context('notebook')
sns.set_style("whitegrid")

# Generate the distplot
sns.distplot(x, kde=True, color='steelblue',
kde_kws={"label": "KDE",'color': '#66aa44','linewidth': 3, 'linestyle': '--',␣
↪'alpha': 0.8})
plt.xlabel("Distribution", labelpad=15)
plt.ylabel("Density", labelpad=15)
plt.title("Binomial Distribution", fontsize=15, y=1.012);
```



1.12 2. Explanation

A coin toss in mathematical terms can be thought of as a binomial experiment, were we have a coin with the probability of getting a head a success and tails a failure. We assume it's a "fair coin" for the task explained below. But what is a "fair coin"? In probability theory and statistics, a sequence of independent Bernoulli trials with probability of 1/2 of success on each trial is metaphorically called a "fair coin". One for which the probability is not 1/2 is called a biased coin or unfair coin.

Wait a minute, above I said a coin toss could be a thought of as a Binomial experiment where as later I mentioned a Bernoulli trial. Are these not different. This needs some explanation. "*The Bernoulli distribution represents the success or failure of a single Bernoulli trial. The Binomial Distribution represents the number of successes and failures in* $\boldsymbol{n}$ *independent Bernoulli trials for some given value of* $\boldsymbol{n}$*. For example, if a manufactured item is defective with probability* $\boldsymbol{p}$*, then*

the binomial distribution represents the number of successes and failures in a lot of ***n*** *items. In particular, sampling from this distribution gives a count of the number of defective items in a sample lot. Another example is the number of heads obtained in tossing a coin* ***n*** *times*".

2.1 Import required libraries This section of code covers importing the required libraries needed.

"'python # Import the required libraries import numpy as np import seaborn as sns import matplotlib.pyplot as plt

2 Graphics in retina format are more sharp and legible.

%config InlineBackend.figure_format = 'retina'

3 Sets the backend of matplotlib to the 'inline' backend.

%matplotlib inline"'

2.2 Random binomial In this section code I'm generating results of the binomial. Running the coin toss experiment 1000 times, where in each experiment we toss a fair coin 100 times asking how many heads we see in each of the 100 experiments. I have set the seed for the rng so that the results can be replicated.

```
# number of coin toss
n = 100

# set the probability
p = 0.5

# number of experiments
s = 1000

# generate the results
x = np.random.default_rng(1).binomial(n, p, s)
```

Additional information: If you are looking for more detail on random number generation in NumPy you can enter the following statement np.info(rng) statement in your own Jupyter notebook.

2.3 View the outcome Now we print the array of results.

```
python # view the outcome x
```

2.4 Visualize the results This section applies the visulation style and plots the outcome of the code.

```
# Visualisation styling code
sns.set(rc={'figure.figsize': (10, 5)})
sns.set_context('notebook')
sns.set_style("whitegrid")
```

```
# Generate the distplot
sns.distplot(x, kde=True, color='steelblue',
kde_kws={"label": "KDE",'color': '#66aa44','linewidth': 3, 'linestyle': '--', 'alpha': 0.8})
plt.xlabel("Distribution", labelpad=15)
plt.ylabel("Density", labelpad=15)
plt.title("Binomial Distribution", fontsize=15, y=1.012);
```

3.1 3. References

[1] Python and R Tips. 2018. *Simulating Coin Toss Experiment In Python With Numpy - Python And R Tips.* [online] Available at: https://cmdlinetips.com/2018/12/simulating-coin-toss-experiment-with-binomial-random-numbers-using-numpy/ [Accessed 17 November 2020].

[2] Medium. 2019. *How To Code A Fair Coin Flip In Python.* [online] Available at: https://towardsdatascience.com/how-to-code-a-fair-coin-flip-in-python-d54312f33da9 [Accessed 17 November 2020].

[3] Umayanga, T., 2020. *What Is Binomial Distribution And How To Plot It Using Python.* [online] Medium. Available at: https://medium.com/@tharakau/what-is-binomial-distribution-and-how-to-plot-it-using-python-e9d95bd341ee [Accessed 17 November 2020].

[4] Unf.edu. 2020. *Empirical Distributions.* [online] Available at: https://www.unf.edu/~cwinton/html/cop4300/s09/class.notes/DiscreteDist.pdf [Accessed 2 December 2020].

[5] Mimi Mathematics. 2020. *Fair Coin.* [online] Available at: https://en.mimi.hu/mathematics/fair_coin.html [Accessed 2 December 2020].

[6] Python and R Tips. 2020. *Simulating Coin Toss Experiment In Python With Numpy - Python And R Tips.* [online] Available at: https://cmdlinetips.com/2018/12/simulating-coin-toss-experiment-with-binomial-random-numbers-using-numpy/ [Accessed 20 November 2020].

3.2 Task 4 - Simpon's paradox

Simpson's paradox is a well-known statistical paradox where a trend evident in a number of groups reverses when the groups are combined into one big data set. Use numpy to create four data sets, each with an **x** array and a corresponding **y** array, to demonstrate Simpson's paradox. You might create your **x** arrays using numpy.linspace and create the **y** array for each **x** using notation like **y = a * x + b** where you choose the **a** and **b** for each **x, y** pair to demonstrate the paradox. You might see the Wikipedia page for Simpson's paradox for inspiration.

It occurs when the "*the correlation calculated for a dataset changes direction if the dataset is split into groups*". In effect a positive correlation when looking at a whole dataset not only disappears but ends up with a negative correlation when the data is split/partitioned. It illustrates "*the need to be skeptical of conclusions that rely on statistics that depend on how the data is aggregated*". It is a type of omitted variable bias , the result of which one variable overcomes the explanation of

a missing or lurking variable. This leads to a "*incorrect interpretation of an apparently obvious insight*". (Abousalh-Neto, N., 2020,(1))

3.3 1. Solution

1.1 Import required libraries

[7]:
```
# Import required libraries
import numpy as np
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt

# Graphics in retina format are more sharp and legible.
%config InlineBackend.figure_format = 'retina'

# Sets the backend of matplotlib to the 'inline' backend.
%matplotlib inline
```

1.2 Create first dataset This section of code creates the first dataset using NumPy linspace for the **x** values and the linear equation for the **y** values. I've added some noise to for aesthetics.

NumPy linspace: linspace is an in-built function in Python's NumPy library. It is used to create an evenly spaced sequence in a specified interval.

Linear equation: A linear equation is any equation that can be written in the form. ax+b=0. where a and b are real numbers and x is a variable. This form is sometimes called the standard form of a linear equation

[8]:
```
# create first dataset
x1 = np.linspace(0, 1, 24)
noise = np.random.normal(0, 0.2, x1.shape)
a, b = -1, 1
y = a*x1+b
y1 = y+noise
```

Lets have a look at the data.

[9]:
```
# view x1 array
x1
```

[9]:
```
array([0.        , 0.04347826, 0.08695652, 0.13043478, 0.17391304,
       0.2173913 , 0.26086957, 0.30434783, 0.34782609, 0.39130435,
       0.43478261, 0.47826087, 0.52173913, 0.56521739, 0.60869565,
       0.65217391, 0.69565217, 0.73913043, 0.7826087 , 0.82608696,
       0.86956522, 0.91304348, 0.95652174, 1.        ])
```

[10]:
```
# view y1 array
y1
```

[10]:
```
array([ 0.67951846,  0.89271986,  0.98031638,  1.00598405,  0.86163535,
        0.6124299 ,  0.51518619,  0.80052795,  0.38660513,  0.65979496,
        0.44865481,  0.26476429,  0.70977443,  0.4566235 ,  0.35088484,
        0.07668496,  0.69280803,  0.6279839 ,  0.44215756,  0.14625968,
        0.27658181,  0.17818195,  0.46759558, -0.0283186 ])
```

The below section of code displays the coefficient of **x1** and **y1**.

Coefficient: a value, in mathematics, that appears in front of and multiplies another value. For example in 2x + 4y = 7, 2 is the coefficient of x

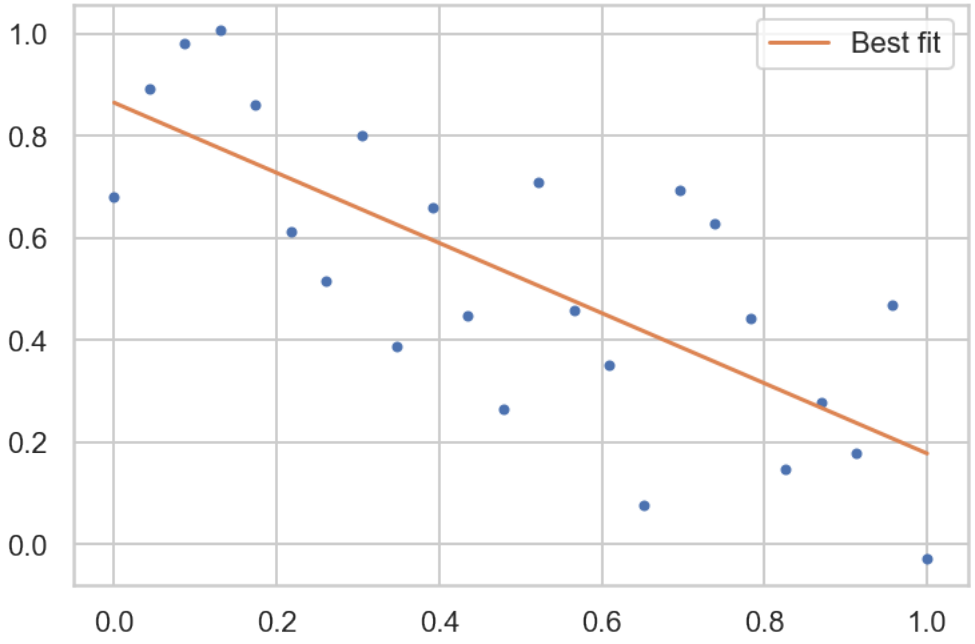
NumPy polyfit: This is a pretty general least squares polynomial fit function which accepts the data set and a polynomial function of any degree (specified by the user), and returns an array of coefficients that minimizes the squared error.

[11]:
```
# coefficient
coeffs1 = np.polyfit(x1, y1, 1)
coeffs1
```

[11]:
```
array([-0.6876621 ,  0.86488751])
```

Now lets plot the data.

[12]:
```
plt.plot(x1, y1, '.');
plt.plot(x1, coeffs1[0] * x1 + coeffs1[1], '-', label='Best fit')
plt.legend();
```



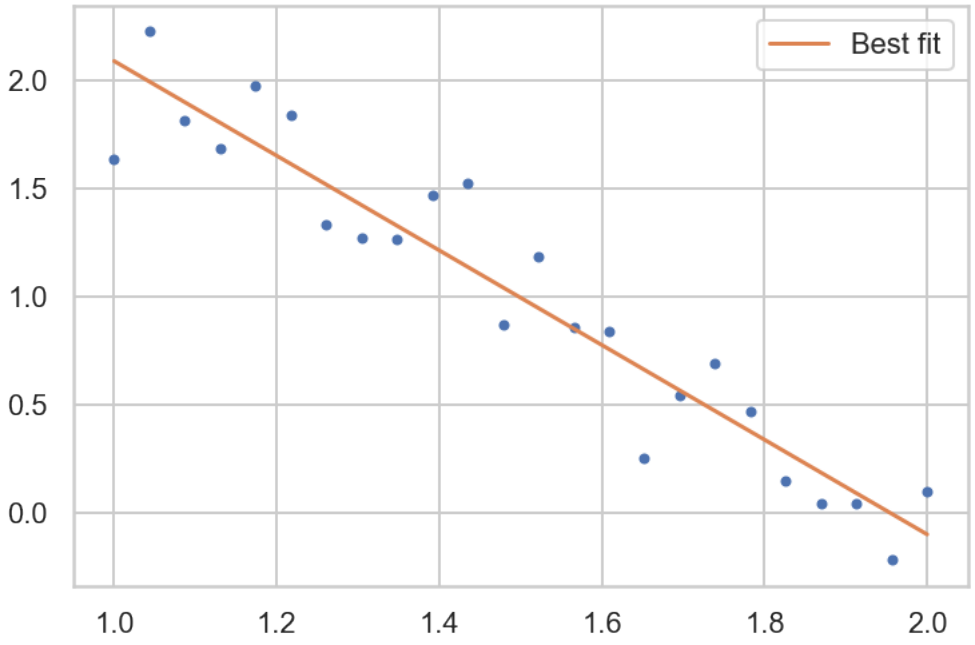
1.3 Create second dataset Note: For creating datasets 2, 3 and 4, I excluded the code that viewed the outputted arrays as It was only for illustrative purposes.

[13]:
```
# create second dataset
x2 = np.linspace(1, 2, 24)
noise = np.random.normal(0, 0.2, x2.shape)
a, b = -2, 4
y = a*x2+b
y2 = y+noise
```

[14]:
```
# coefficient
coeffs2 = np.polyfit(x2, y2, 1)
coeffs2
```

[14]:
```
array([-2.18794466,  4.27503146])
```

[15]:
```
plt.plot(x2, y2, '.');
plt.plot(x2, coeffs2[0] * x2 + coeffs2[1], '-', label='Best fit')
plt.legend();
```



1.4 Create third dataset

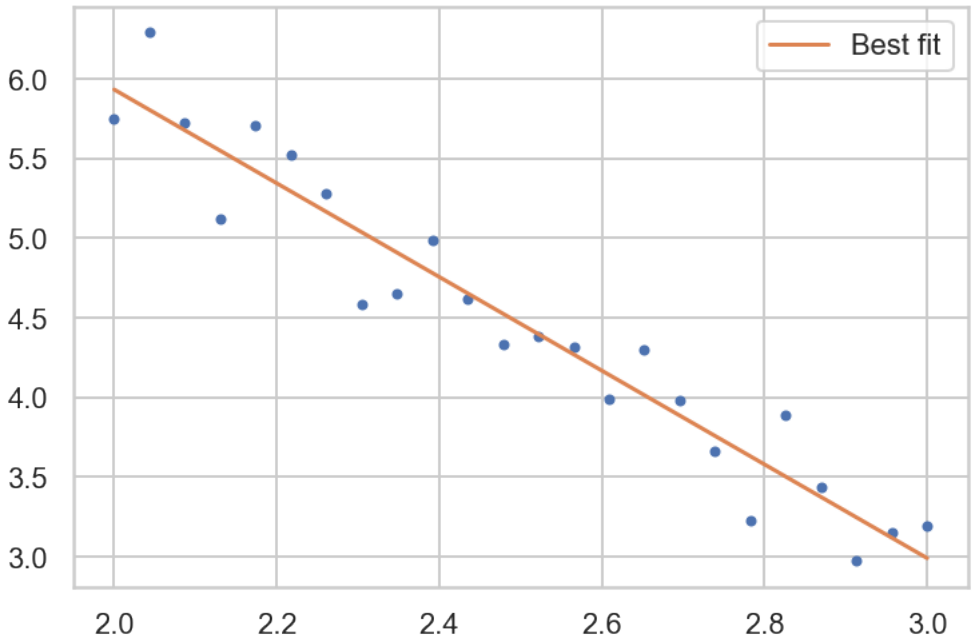
[16]:
```
# create third dataset
x3 = np.linspace(2, 3, 24)
noise = np.random.normal(0, 0.2, x3.shape)
a, b = -3, 12
y = a*x3+b
```

```
y3 = y+noise
```

[17]:
```
# coefficient
coeffs3 = np.polyfit(x3, y3, 1)
coeffs3
```

[17]:
```
array([-2.942202  , 11.81628692])
```

[18]:
```
plt.plot(x3, y3, '.');
plt.plot(x3, coeffs3[0] * x3 + coeffs3[1], '-', label='Best fit')
plt.legend();
```



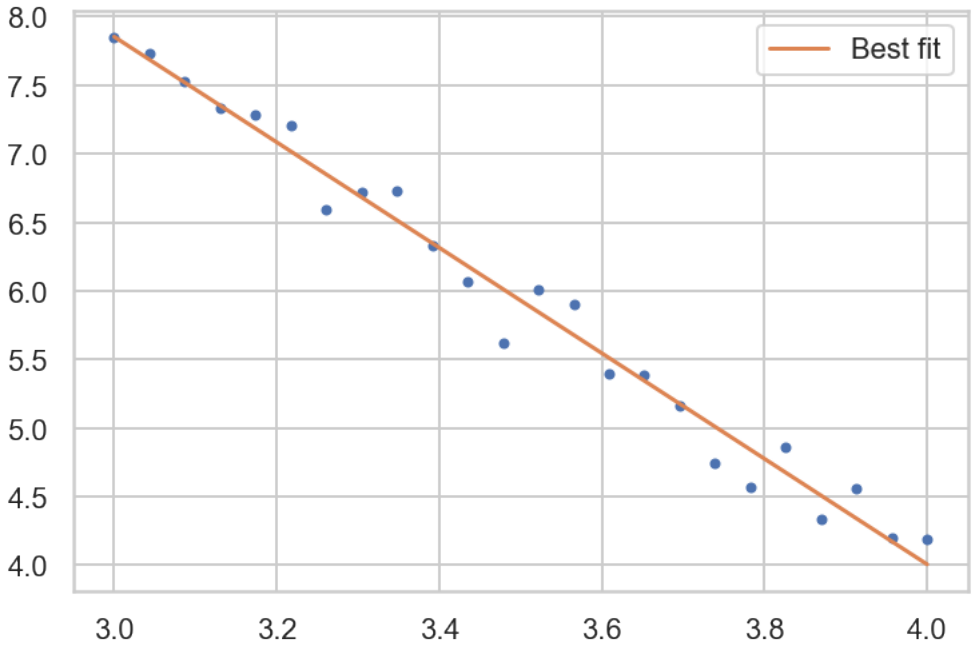
1.5 Create fourth dataset

[19]:
```
# create fourth dataset
x4 = np.linspace(3, 4, 24)
noise = np.random.normal(0, 0.2, x4.shape)
a, b = -4, 20
y = a*x4+b
y4 = y+noise
```

[20]:
```
# coefficient
coeffs4 = np.polyfit(x4, y4, 1)
coeffs4
```

[20]:
```
array([-3.8481155 , 19.39807952])
```

[21]:
```
plt.plot(x4, y4, '.');
plt.plot(x4, coeffs4[0] * x4 + coeffs4[1], '-', label='Best fit')
plt.legend();
```



1.6 Concatenate the datasets Now it's time to join the datasets together and see what happens.

[22]:
```
# concatenate the four datsets created earlier
x_array = np.concatenate((x1, x2, x3, x4))
y_array = np.concatenate((y1, y2, y3, y4))
```

[23]:
```
# coefficient
coeffs5 = np.polyfit(x_array, y_array, 1)
coeffs5
```

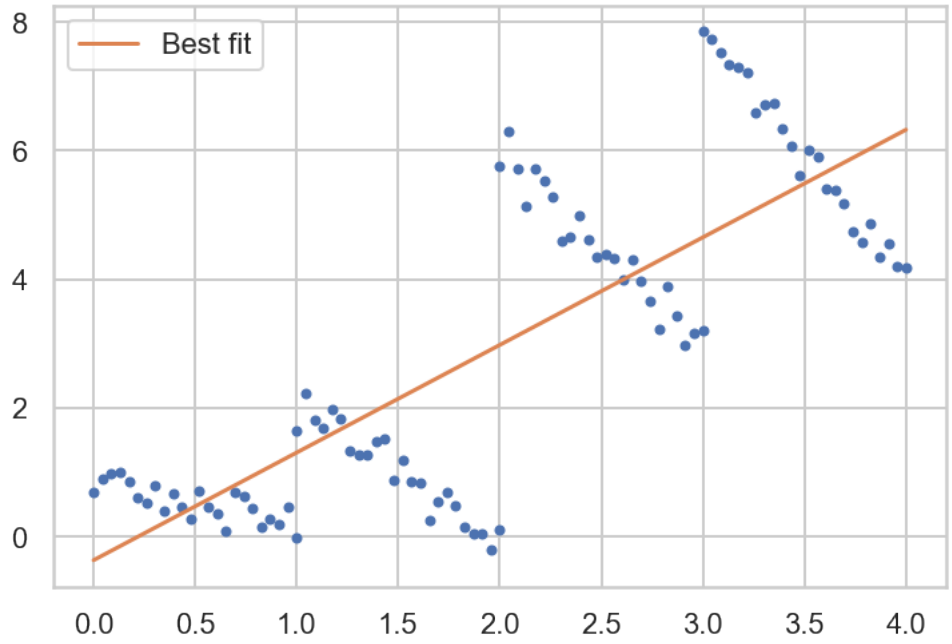
[23]:
```
array([ 1.67301229, -0.36986755])
```

Now lets plot the combined dataset and see what happens. According to Simpson's Paradox the correlation should change. In the four datasets we created there was a negative correlation between the x and y values by design.

Negative correlation: Negative correlation is a relationship between two variables in which one variable increases as the other decreases, and vice versa. In statistics, a perfect negative correlation is represented by the value -1, a 0 indicates no correlation, and a +1 indicates a perfect positive correlation. A perfect negative correlation means the relationship that exists between two variables is negative 100% of the time. (Investopedia. 2020.,(6))

1.7 Plotting

[24]:
```
plt.plot(x_array, y_array, '.');
plt.plot(x_array, coeffs5[0] * x_array + coeffs5[1], '-', label='Best fit')
plt.legend();
```



As expected the correlation is now positive.

Positive correlation: Positive correlation is a relationship between two variables in which both variables move in tandem—that is, in the same direction. A positive correlation exists when one variable decreases as the other variable decreases, or one variable increases while the other increases.

In statistics, a perfect positive correlation is represented by the correlation coefficient value +1.0, while 0 indicates no correlation, and -1.0 indicates a perfect inverse (negative) correlation. (Investopedia. 2020.,(7))

3.4 2. References

[1] Abousalh-Neto, N., 2020. *The Desert And The Dunes: Finding Oases And Avoiding Mirages With SAS Visual Analytics*. [online] Support.sas.com. Available at: https://support.sas.com/resources/papers/proceedings14/SAS252-2014.pdf [Accessed 3 December 2020].

[2] Long, R., 2016. *Can You Please Explain Simpson's Paradox With Equations, Instead Of Contingency Tables?*. [online] Cross Validated. Available at: https://stats.stackexchange.com/questions/221892/can-you-please-explain-simpsons-paradox-with-equations-instead-of-contingency [Accessed 4 December 2020].

[3] Atri, Y., 2020. *PYTHON: Solving Y=A+BX Where X Is A Predefined List**. [on-

line] Stack Overflow. Available at: https://stackoverflow.com/questions/60337609/python-solving-y-abx-where-x-is-a-predefined-list [Accessed 3 December 2020].

[4] Ebner, J., 2020. *How To Use The Numpy Linspace Function*. [online] Sharp Sight. Available at: https://www.sharpsightlabs.com/blog/numpy-linspace/ [Accessed 6 December 2018].

[5] utorial.math.lamar.edu. 2020. *Algebra - Linear Equations*. [online] Available at: https://tutorial.math.lamar.edu/classes/alg/solvelineareqns.aspx [Accessed 3 December 2020].

[6] Investopedia. 2020. *Negative Correlation Definition*. [online] Available at: https://www.investopedia.com/terms/n/negative-correlation.asp [Accessed 7 December 2020].

[7] Investopedia. 2020. *Understanding Positive Correlation*. [online] Available at: https://www.investopedia.com/terms/p/positive-correlation.asp [Accessed 7 December 2020].