Abril de 2020

Informe: Simulación de lanzamiento de dados

Universidad Politécnica Salesiana

Estudiante: Bryam David Vega Moreno

Maestro: Ing. Diego Quisi

Problema

Desarrollar una aplicación que simule el lanzamiento de dos dados en cualquier lenguaje que genere un histograma con el número de ocurrencias de la sumatoria,teniendo diferentes escenarios de lanzamiento:

- 10
- 100
- 1000
- 10000
- 100000
- 1000000

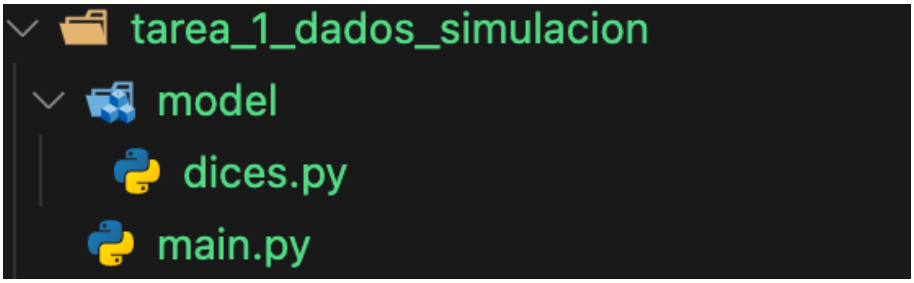
Desarrollo

En este caso utilizamos a **Python** como lenguaje de programación para poder realizar dicha aplicación. El objetivo de utilizar python es que dicho lenguaje tiene una gran potencialidad para calculos matemáticos. Ademas de ello cuenta con un conjunto de librerías que facilitan el desarrollo de la aplicación.

Para poder realizar esta aplicación utilizamos la siguiente lógica:

1. Generamos dos números randómicos que estan entre 1 y 6 puesto que esos son los valores que contiene un dado.
2. Procedemos a sumar dichos números randómicos y guardarlos en una lista
3. Graficamos la lista generada utilizando un histograma donde en el eje de **x** están las posibles sumatorias y en el eje **y** se tiene la frecuencia de las posibles sumatorias.

Si bien parece una aplicación sencilla de desarrollar debido a que los pasos a seguir son sencillos, lo que se desea hacer es **automatizar** el proceso de simulación con el objetivo de que pueda funcionar para distintos escenarios en los que el usuario desee probar y de esa manera se genere el conjunto de gráficas correspondientes. Ademas de ello buscamos una programación un poco más avanzada utilizando entidades para normalizar nuestro código y tenerlo de una manera más ordenada. Para este caso la lógica de nuestra aplicación esta distribuida de la siguiente manera:



Una desventaja de python es que no se tiene un orden fijo para programar, por lo que suele generar un desorden tedioso al momento de codificar, sin embargo, para evitar esto utilizamos un modelo similar al mvc con el fin de encontrar orden. Una vez entendido la estructura de nuestra aplicación procedemos a explicarla

dices.py

```python
import random

class Dice:

    def __init__(self):
        self.__value =  random.randint(1,6)

    def _throw_dice(self):
        self._set_value(random.randint(1,6))
        return self._get_value()

    def _set_value(self,value):
        self.__value = value

    def _get_value(self):
        return self.__value
```

Este archivo contiene una entidad denominado Dice (Dado) que nos va a permitir simular un dado y su valor, como podemos ver contiene el método **_trhow_dice** el cual permite lanzar el dado y obtener un valor randómico para nuestra aplicación.

main.py

```python
import argparse
import coloredlogs
import datetime
import logging
logging.basicConfig(level=logging.INFO)
import pandas as pd
import numpy as np
import os

from model.dices import Dice

logger =  logging.getLogger(__name__)
coloredlogs.install()

def main(list_simulate):
    logger.info("Starting simulate to {}".format(list_simulate))

    try:
        os.mkdir('figures')
    except:
        logger.error('Creation of the directory {} failed'.format('figures'))

    for i in list_simulate:
            df = pd.DataFrame({'sum':_simulate_sum(i)})
            _generate_histogram(df,'figure_size_{}'.format(i),i)


def _simulate_sum(size):
    logger.info("Simulate Dice sum with size of {}".format(size))

    dice_one = Dice()
    dice_two = Dice()
    frequency = [dice_one._throw_dice()+dice_two._throw_dice() for i in range(0,size)]
    return frequency


def _generate_histogram(df,name_figure,size):
    logger.info("Generating Histograme for {}".format(name_figure))

    now = datetime.datetime.now().strftime('%Y_%m_%d')
    hist = df.plot(kind='hist',figsize=(10,7),bins=11,
                   alpha=0.7,color='yellowgreen',grid=True,
                   title='Frequency of addition of two dice for a test of {} values'.format(size))
    fig = hist.get_figure()
    fig.savefig('figures/{name}_{date}_histogram.png'.format(name=name_figure,date=now))


if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    parser.add_argument('list_simulate',
                        help='List of values to simulate. For example : python main.py 10,100',
                        type=str)
    args = parser.parse_args()

    main([int(i) for i in args.list_simulate.split(',')])
```

Este archivo contiene las funciones que nos van a permitir obtener las sumatorias de los lanzamientos de datos y a su vez guardar los histogramas para los diferentes escenarios. Dicho archivo contiene las siguientes funciones:

- **main:** Esta función ejecuta la aplicación las funciones de sumatoria para los diferentes escenarios que el usuario escoja
- **_simulate_sum:** Función que simula la suma de los dados y retorna una lista con todos los lanzamientos realizados.
- **_generate_histogram:** Genera el histograma para el escenario propuesto con el fin de visualizar la ocurrencia de las sumatorias al realizar el lanzamiento de dados.

Una vez revisado el código, procedemos a realizar un análisis de los escenarios obtenidos a lo largo de las pruebas realizadas con la aplicación.

Análisis de resultados

Ejecutando la aplicación, realizamos las pruebas para los siguientes escenarios [10,100,1000,10000,100000,1000000], con el fin de ver como se modificada el histograma.

```
(base) bvegam@MacBook-Pro-de-bryam tarea_1_dados_simulacion % python main.py 10,100,1000,10000,100000,1000000
2021-04-07 20:32:40 MacBook-Pro-de-bryam.local __main__[22939] INFO Starting simulate to [10, 100, 1000, 10000, 100000, 1000000]
2021-04-07 20:32:40 MacBook-Pro-de-bryam.local __main__[22939] INFO Simulate Dice sum with size of 10
2021-04-07 20:32:40 MacBook-Pro-de-bryam.local __main__[22939] INFO Generating Histograme for figure_size_10
2021-04-07 20:32:40 MacBook-Pro-de-bryam.local __main__[22939] INFO Simulate Dice sum with size of 100
2021-04-07 20:32:40 MacBook-Pro-de-bryam.local __main__[22939] INFO Generating Histograme for figure_size_100
2021-04-07 20:32:40 MacBook-Pro-de-bryam.local __main__[22939] INFO Simulate Dice sum with size of 1000
2021-04-07 20:32:40 MacBook-Pro-de-bryam.local __main__[22939] INFO Generating Histograme for figure_size_1000
2021-04-07 20:32:40 MacBook-Pro-de-bryam.local __main__[22939] INFO Simulate Dice sum with size of 10000
2021-04-07 20:32:40 MacBook-Pro-de-bryam.local __main__[22939] INFO Generating Histograme for figure_size_10000
2021-04-07 20:32:41 MacBook-Pro-de-bryam.local __main__[22939] INFO Simulate Dice sum with size of 100000
2021-04-07 20:32:41 MacBook-Pro-de-bryam.local __main__[22939] INFO Generating Histograme for figure_size_100000
2021-04-07 20:32:41 MacBook-Pro-de-bryam.local __main__[22939] INFO Simulate Dice sum with size of 1000000
2021-04-07 20:32:43 MacBook-Pro-de-bryam.local __main__[22939] INFO Generating Histograme for figure_size_1000000
```

Como podemos darnos cuenta hemos corrido nuestra aplicación para los escenarios propuestos en donde guardamos cada uno de los histogramas con el fin de ver su cambio. A continuación mostramos los histogramas obtenidos.

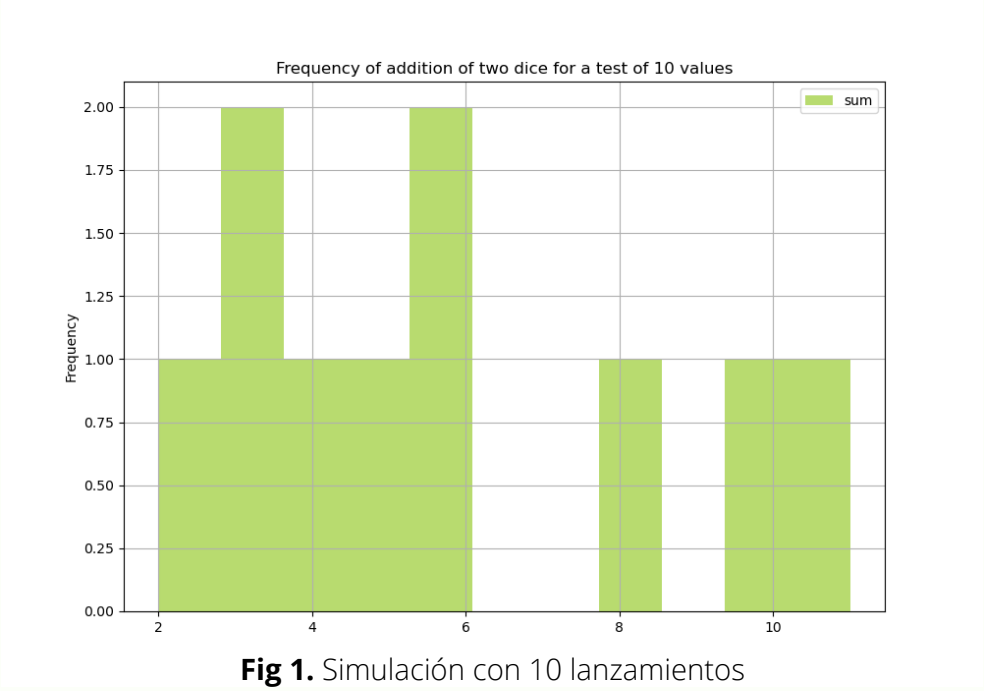


Fig 1. Simulación con 10 lanzamientos

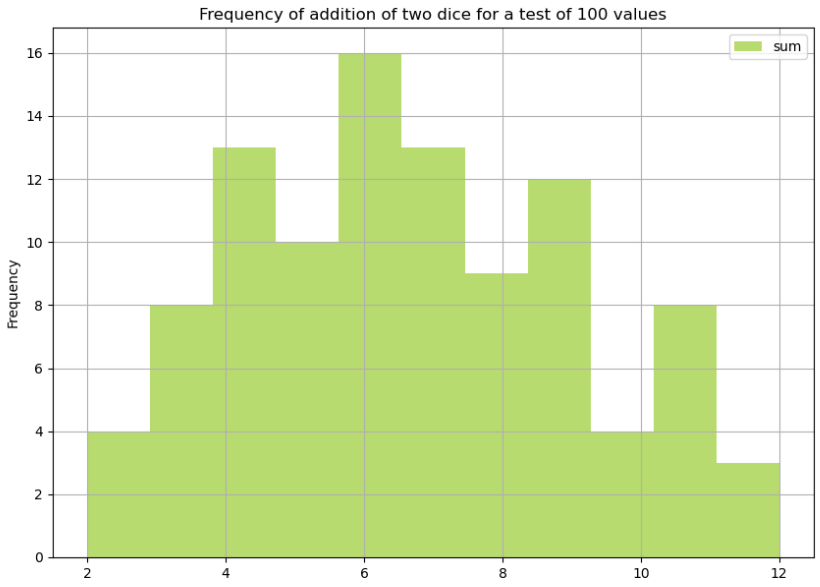


Fig 2. Simulación con 100 lanzamientos

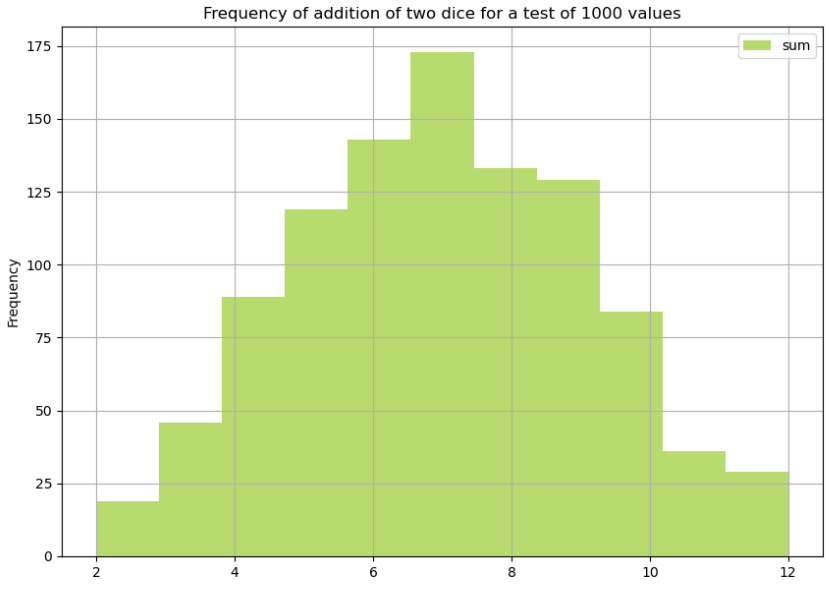


Fig 3. Simulación con 1000 lanzamientos

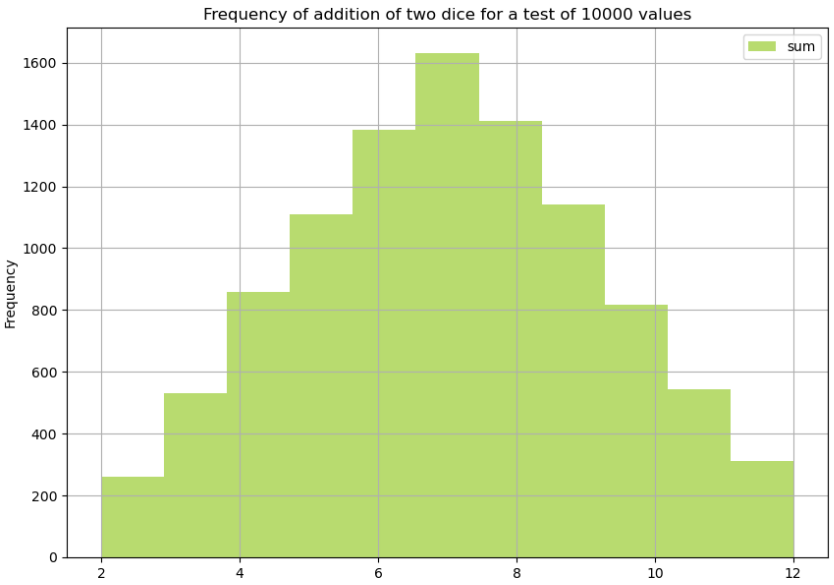


Fig 4. Simulación con 10000 lanzamientos

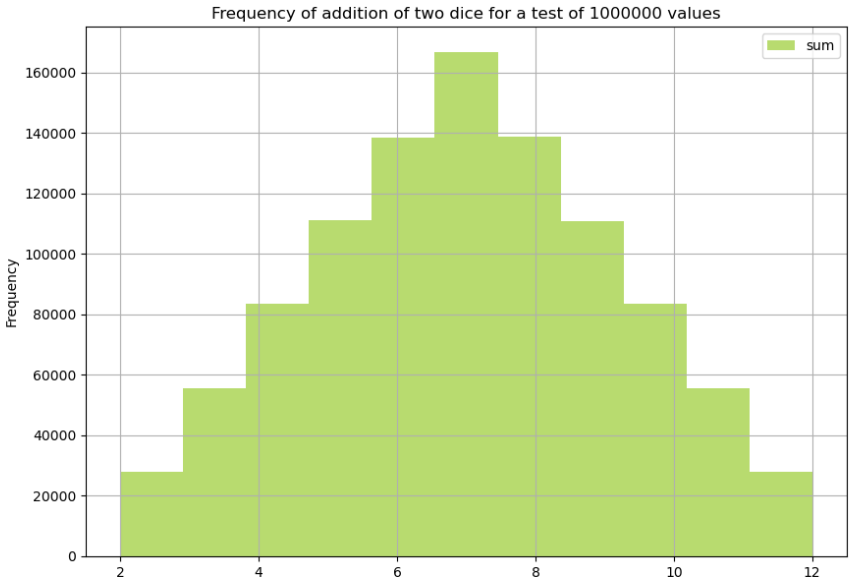


Fig 5. Simulación con 100000 lanzamientos

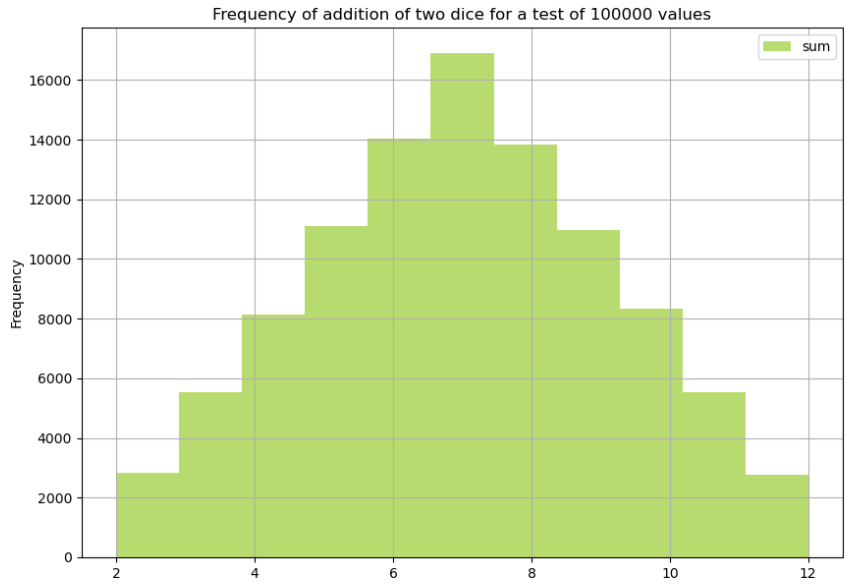


Fig 6. Simulación con 1000000 lanzamientos

Con todas las gráficas presentadas, podemos notar descubrimientos muy interesantes:

- Al utilizar un escenario de 10 lanzamientos podemos darnos cuenta que nuestro histograma es muy pobre en información y no nos da una idea clara de como se estan comportando los datos.
- Al usar un escenario de 100 lanzamientos empezamos a ver un pequeño comportamiento de los datos, sin embargo sigue siendo un poco confuso su comportamiento.
- Cuando realizamos los escenarios para 1000,10000,100000 y 10000000 podemos notar un claro comportamiento de los datos, ya que la gráfica nos presenta una distribución conocida como **"distribución normal".**

La distribuación normal es una de las distribuciones más frecuentes y esta no es más que un modelo teórico capaz de aproximar satisfactoriamente el valor de una variable aleatoria a una situación ideal. Podemos notar que la probabilidad de que la sumatoria sea 2 es la misma probabilidad de que nos salga un 12 y así sucesivamente hasta llegar a un punto medio una campana mejor llamada en donde cae la mayoría de los casos que para este problema seria la sumatoria 7.

Conclusiones

Como nos hemos podido dar cuenta, una de los principales errores al momento de realizar simulaciones es probar con un cierto número de casos límitados, generando que no se pueda apreciar de buena forma el comportamiento de nuestros datos y podamos tomar decisiones equivocadas, por lo que escoger un correcto número de escenarios nos permitirá obtener un mejor feedback del comportamiento de nuestro datos a fin de que podamos entender su comportamiento como lo habiamos comentado como también tomar decisiones a partir de dicho comportamiento. En el ejemplo planteado del lanzamiento de dados es muy interesante puesto que nos permite ver el comportamiento de ladata y así poder escoger la mejor opción, enfocándonos en el ámbito de negocios y suponiendo que tenemos una distribución similar a este problema, podemos tomar una decisión de realizar la importación del producto que esta en la campana de la gráfica, mientras que los productos que están alejados de dicha campana no seran tan importantes para nuestro negocio, ademas de ello podemos tomar un umbral de productos importantes para nuestro negocio y darles mucho más valor.

A continuación dejo el link del repositorio digital en donde se encuentra nuestro código fuente:

https://github.com/bvegaM/simulacion_vega_bryam/tree/master/tarea_1_dados_simulacion