Análisis de Simulación de Ruleta en Python

Autor 1, Autor 2

April 6, 2024

1 Introducción

En este documento, nos adentraremos en el análisis de un código Python diseñado para simular un juego de ruleta y para visualizar diversas métricas estadísticas utilizando la biblioteca matplotlib. La ruleta, un ícono en el ámbito de los juegos de azar, ofrece una ventana fascinante para explorar conceptos fundamentales de probabilidad y estadística. El código objeto de nuestro estudio nos permitirá investigar detalladamente diferentes aspectos del juego, desde la frecuencia relativa de un número específico hasta el comportamiento del promedio, la varianza y el desvío en múltiples sesiones de juego. A través de gráficos dinámicos, podremos observar de manera clara y rigurosa cómo estas métricas evolucionan en diversos escenarios de juego.

2 Clase `Ruleta`

A continuación se presenta la estructura y los métodos de la clase `Ruleta`, que se utiliza para simular el juego de ruleta y realizar el análisis estadístico.

Listing 1: Estructura y métodos de la clase Ruleta

```
class Ruleta:
    def __init__(self, _numero_elegido, _cantidad_tiradas,
    _cantidad_corridas):
       # Constructor de la clase

    def simular_corrida(self):
        # Simula una corrida de la ruleta y calcula las medidas
        estadisticas

    def _simular_corridas(self):
        # Realiza todas las corridas de la simulacion

    def graficar_frecuencia_relativa(self):
        # Grafica la frecuencia relativa para cada corrida
```

```
def graficar_promedio(self):
    # Grafica el promedio para cada corrida

def graficar_varianza(self):
    # Grafica la varianza para cada corrida

def graficar_desvio(self):
    # Grafica el desv o para cada corrida
```

2.1 Explicación de la Clase y Métodos

A continuación se explica brevemente la estructura de la clase 'Ruleta' y sus métodos:

Este método es el constructor de la clase Ruleta, el cual inicializa los parámetros de la simulación, como el número elegido en la ruleta, la cantidad de tiradas por corrida y la cantidad de corridas a simular. El método también maneja posibles errores al simular las corridas.

```
def __init__(self, _numero_elegido, _cantidad_tiradas, _cantidad_corridas
    """
    Inicializa una instancia de la clase Ruleta.

    Par metros:
        _numero_elegido (int): N mero elegido en la ruleta.
        _cantidad_tiradas (int): Cantidad de tiradas por corrida.
        _cantidad_corridas (int): Cantidad de corridas a simular.
    """
    self.numero_elegido = _numero_elegido
    self.cantidad_tiradas = _cantidad_tiradas
    self.cantidad_corridas = _cantidad_corridas
    try:
        self.resultados = self._simular_corridas()
    except Exception as e:
        print("Error al simular las corridas:", e)
```

Este método simula una corrida de la ruleta y calcula las medidas estadísticas, como la frecuencia relativa, el promedio, la varianza y el desvío. Retorna una tupla con las listas de estas medidas.

```
def simular_corrida(self):
    """
    Simula una corrida de la ruleta y calcula las medidas estad sticas.

    Retorna:
        tuple: Tupla conteniendo las listas de frecuencia relativa,
        promedio, varianza y desv o.
    """
    resultados = [
    random.randint(0, 36) for _ in range(self.cantidad_tiradas)
    ]
    contador_numero_elegido = 0
    frecuencias_relativas = []
    promedios = []
    varianzas = []
    desvios = []
    for i, tirada in enumerate(resultados, start=1):
        if tirada == self.numero_elegido:
            contador_numero_elegido += 1
        frecuencia_relativa = contador_numero_elegido / i
        frecuencias_relativas.append(frecuencia_relativa)
        promedio = sum(resultados[:i]) / i
        promedios.append(promedio)
        varianza = sum((x - promedio) ** 2 for x in resultados[:i]) / i
        varianzas.append(varianza)
        desvio = np.sqrt(varianza)
        desvios.append(desvio)
    return frecuencias_relativas, promedios, varianzas, desvios
```

Método _simular_corridas: Este método realiza todas las corridas
de la simulación y retorna una lista de los resultados de cada corrida.

```
def _simular_corridas(self):
    """
    Realiza todas las corridas de la simulaci n.

    Retorna:
        list: Lista de resultados de cada corrida.
    """
    return [self.simular_corrida()
    for _ in range(self.cantidad_corridas)]
```

Método graficar_frecuencia_relativa:
Este método grafica la frecuencia relativa para cada corrida del juego de la ruleta.

```
def graficar_frecuencia_relativa(self):
    """
    Grafica la frecuencia relativa para cada corrida.
    """
    try:
        self._graficar('Frecuencia Relativa')
    except Exception as e:
        print("Error al graficar la frecuencia relativa:", e)
```

Método graficar_promedio:
Este método grafica el promedio para cada corrida del juego de la ruleta.

```
def graficar_promedio(self):
    """
    Grafica el promedio para cada corrida.
    """
    try:
        self._graficar('Promedio')
    except Exception as e:
        print("Error al graficar el promedio:", e)
```

Método graficar_varianza:
Este método grafica la varianza para cada corrida del juego de la ruleta.

```
def graficar_varianza(self):
    """
    Grafica la varianza para cada corrida.
    """
    try:
        self._graficar('Varianza')
    except Exception as e:
        print("Error al graficar la varianza:", e)
```

Método graficar_desvio:
Este método grafica el desvío para cada corrida del juego de la ruleta.

```
def graficar_desvio(self):
    """
    Grafica el desvío para cada corrida.
    """
    try:
        self._graficar('Desvío')
    except Exception as e:
        print("Error al graficar el desvío:", e)
```

Método _graficar:
Este método interno se utiliza para graficar las diferentes
métricas estadísticas para cada corrida del juego de la ruleta.

```
def _graficar(self, ylabel):
    """
    Función interna para graficar.

    Parametros:
        ylabel (str): Etiqueta del eje y.
    """
    plt.figure(figsize=(20, 10))
    labels = {
        'Frecuencia Relativa': 0,
        'Promedio': 1,
        'Varianza': 2,
        'Desvío': 3
    }
    index = labels[ylabel]
    for i, corrida in enumerate(self.resultados, start=1):
        data = corrida[index]
        plt.plot(
            range(self.cantidad_tiradas),
            data,
            label=f'Corrida {i}'
        )
    plt.xlabel('Cantidad de tiradas')
    plt.ylabel(ylabel)
    plt.title(ylabel)
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.2),
        ncol=len(self.resultados), fancybox=True,shadow=True
        )
    plt.grid(True)
    plt.show()
```

3 Uso del Código

Para utilizar la clase 'Ruleta' y sus métodos, puedes seguir estos pasos:

1. Crear una instancia de la clase 'Ruleta' proporcionando los argumentos necesarios.

2. Llamar a los métodos correspondientes para realizar la simulación y generar las gráficas.

3. Ejemplo:

```
parser = argparse.ArgumentParser(description='Simulacion de ruleta')
parser.add_argument('-c', '--numero_corridas', type=int, default=-1,
help='Nmero de corridas (por defecto: 5)')
parser.add_argument('-n', '--numero_tiradas', type=int, default=-1,
help='Nmero de tiradas (por defecto: 100)')
parser.add_argument('-e', '--numero_eleguido', type=int, default=0,
help='Nmero elegido (por defecto: 0)')

args = parser.parse_args()
cantidad_corridas, cantidad_tiradas,
numero_elegido = args.numero_corridas, args.numero_tiradas,
args.numero_eleguido
try:
    if cantidad_corridas < 0:
        raise ValueError("-c debe ser un entero positivo")
    if cantidad_tiradas < 0:
        raise ValueError("-n debe ser un entero positivo")
    if not 0 <= numero_elegido <= 36:
        raise ValueError("-e debe ser un entero positivo entre 0 y 36")
except ValueError as ve:
    print("Error en los argumentos de entrada:", ve)
    exit()

ruleta = Ruleta(numero_elegido, cantidad_tiradas, cantidad_corridas)
ruleta.graficar_frecuencia_relativa()
ruleta.graficar_promedio()
ruleta.graficar_varianza()
ruleta.graficar_desvio()
```

github: https://github.com/pepeargent0/simulation_utn