1. Informe:

   El código proporcionado es una herramienta eficiente para generar y calcular distancias entre puntos en múltiples espacios dimensionales. Está diseñado para ser utilizado en experimentos que requieran la generación de datos aleatorios en diferentes dimensiones y la posterior medición de distancias entre estos datos.

   El código consta de las siguientes partes clave:

   Cálculo de Distancias: El código implementa una función llamada distancia_formula que calcula la distancia euclidiana entre dos puntos en un espacio n-dimensional. Utiliza la fórmula estándar para calcular la distancia euclidiana.

   Generación de Datos Aleatorios: Mediante la función crear_datos, el código genera datos aleatorios en espacios n-dimensionales. Utiliza un generador de números aleatorios para crear puntos con coordenadas en el rango [0.0, 1.0].

   Almacenamiento de Resultados: El código guarda las distancias calculadas en archivos de texto separados. Utiliza la función calcularYGuardar para realizar estos cálculos y guardar los resultados. El nombre de cada archivo de salida contiene información sobre la dimensión correspondiente.

   Configuración Flexible: El código permite configurar el número de puntos (p) y una lista de dimensiones en las que se desean realizar los cálculos. Esto hace que sea fácil adaptar el código a diferentes necesidades de experimentación.

   En resumen, este código es una herramienta versátil para generar y analizar datos en diferentes espacios dimensionales. Facilita la realización de experimentos y la recopilación de datos para su posterior procesamiento y análisis.

2. Código:

2.1. C++:

- Función para calcular la distancia entre dos puntos en n distancias:

```
double distancia_formula(const vector<double>& punto1, const vector<double>& punto2) {
    double sumaCuadrados = 0.0;

    for (size_t i = 0; i < punto1.size(); ++i) {
        double diferencia = punto1[i] - punto2[i];
        sumaCuadrados += diferencia * diferencia;
    }
    return sqrt( X: sumaCuadrados);
}
```

- Generar valores aleatorias paras las coordenadas de los puntos:

```
auto crear_datos(int p,int dimensiones) -> vector<vector<double>> {
    random_device rd;
    mt19937 gen( sd: rd());
    uniform_real_distribution<> dis( a: 0.0,  b: 1.0);
    vector<vector<double>>datos;
    for(int i=0;i<p;i++){
        vector<double> punto;
        for(int j=0;j<dimensiones;j++){
            punto.push_back(dis( &: gen));
        }
        datos.push_back(punto);
    }
    return datos;
```

- Procesar los datos para su evaluación y guardado posterior.

```
void calcularYGuardar(const vector<vector<double>>& puntos, const string& dimension) {
    ofstream archivo( s: "datos_dimension_"+dimension+".txt");
    if (!archivo.is_open()) {
        std::cerr << "No se pudo abrir el archivo para escritura." << std::endl;
        return;
    }
    for (size_t i = 0; i < puntos.size(); ++i) {
        for (size_t j = i + 1; j < puntos.size(); ++j) {
            double distancia = distancia_formula( punto1: puntos[i],  punto2: puntos[j]);
            archivo<<distancia<<endl;
        }
    }
    archivo.close();
}
```

- main:

```cpp
int main() {
    int p=100;
    vector<int>dimensiones={10, 50, 100, 500, 1000, 2000, 5000};
    vector<string> demensiones_cadena;
    for (int dimension : dimensiones) {
        string cadena = to_string( val: dimension);
        demensiones_cadena.push_back(cadena);
    }
    for(int i=0;i<dimensiones.size();i++){
        vector<vector<double>> puntos = crear_datos(p, dimensiones: dimensiones[i]);
        calcularYGuardar(puntos,  dimension: demensiones_cadena[i]);
    }

    return 0;
}
```

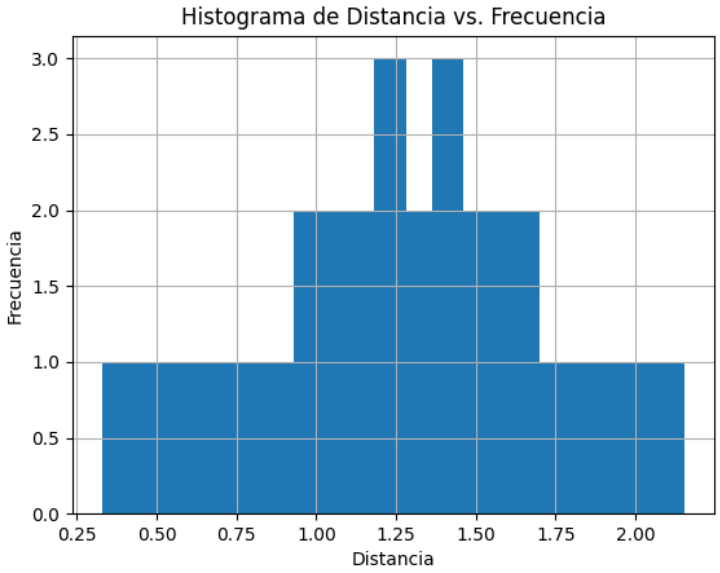
2.2. Python:

- Graficar los datos obtenidos previamente generados en un C++, almacenados en archivos de texto:

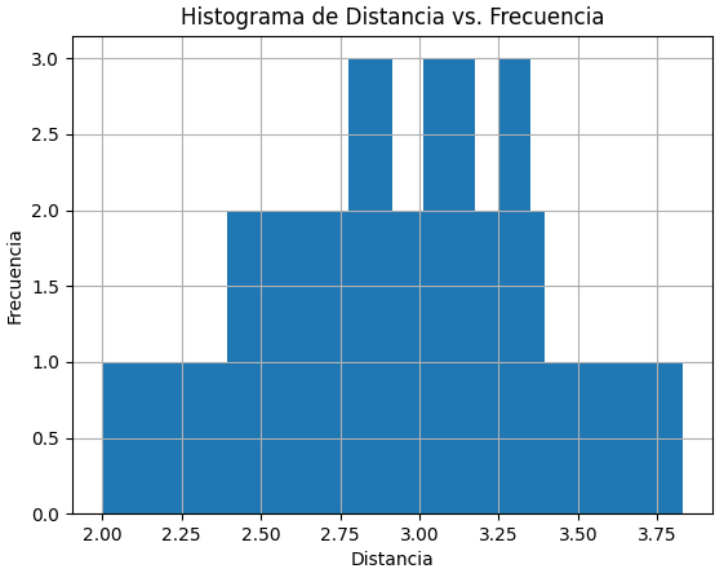
```python
import matplotlib.pyplot as plt

frecuencias = {}

with open("datos_dimension_5000.txt", "r") as archivo:
    for linea in archivo:
        distancia = float(linea.strip())
        if distancia in frecuencias:
            frecuencias[distancia] += 1
        else:
            frecuencias[distancia] = 1

distancias = list(frecuencias.keys())
frecuencias = list(frecuencias.values())

plt.bar(distancias, frecuencias, width=0.1)
plt.xlabel("Distancia")
plt.ylabel("Frecuencia")
plt.title("Histograma de Distancia vs. Frecuencia")
plt.grid(True)


plt.savefig("histograma5000.png")

plt.show()
```

3. Resultados:

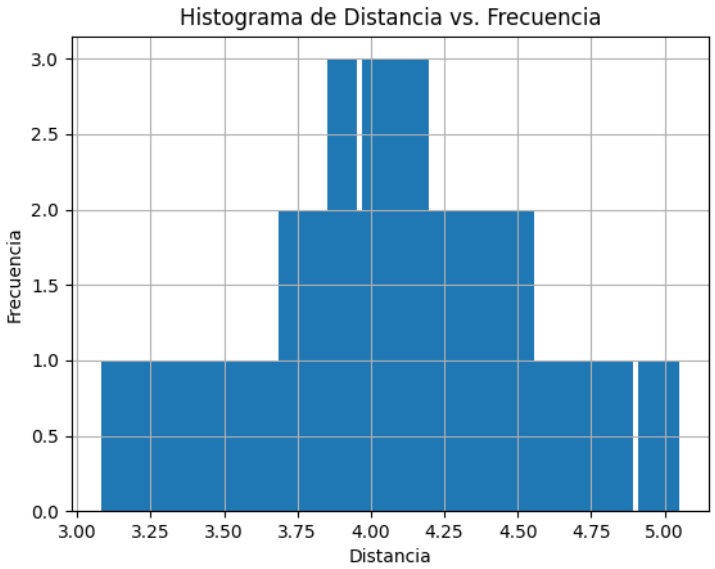
3.1. Dimension 10:



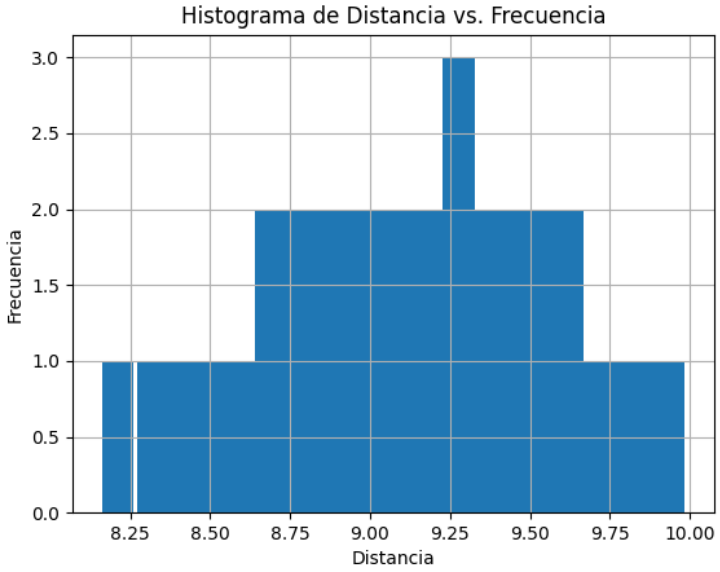
3.2. Dimension 50:



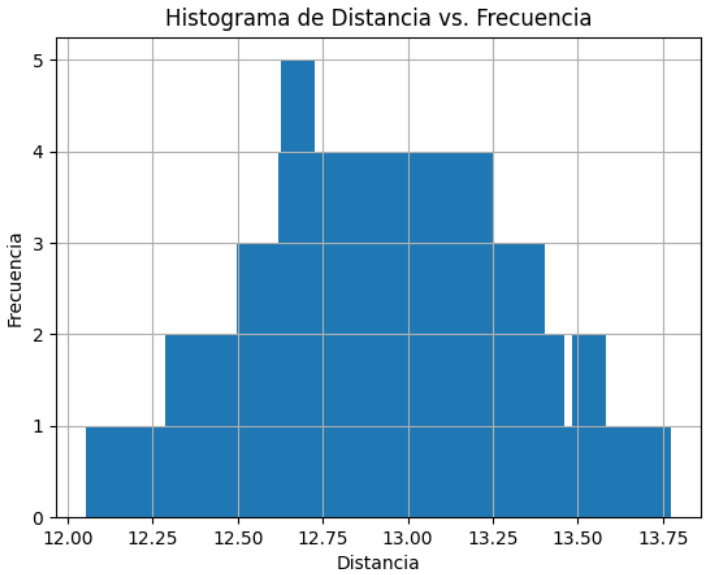
3.3. Dimension 100:



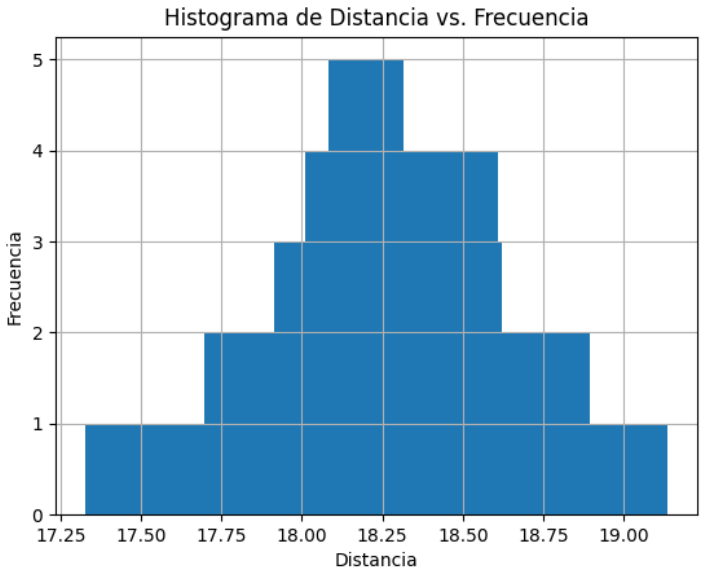
3.4. Dimension 500:



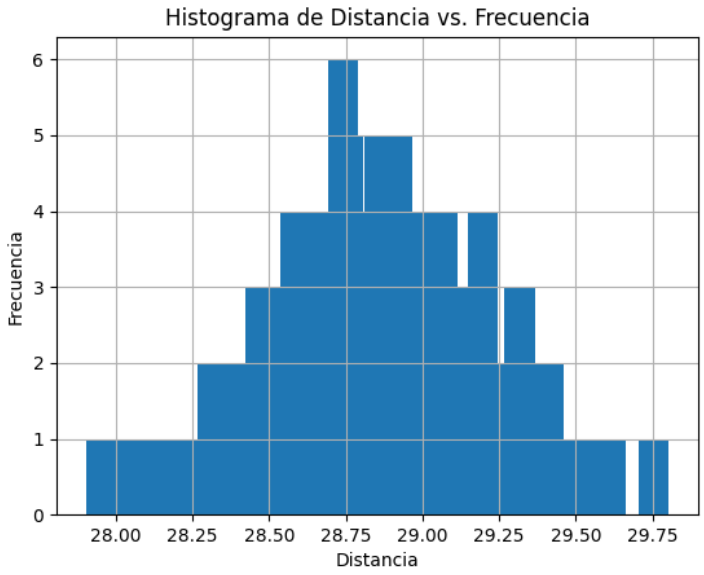
3.5. Dimension 1000:



3.6. Dimension 2000:



3.7. Dimension 5000:



4. Conclusiones:

El estudio y análisis de datos en espacios de diferentes dimensiones son fundamentales en una amplia gama de disciplinas, desde la estadística hasta la ciencia de datos y la inteligencia artificial. En este contexto, el código proporcionado ofrece una valiosa herramienta para explorar y comprender cómo las dimensiones afectan la distribución y las relaciones entre los datos.

La elección de la dimensionalidad en análisis de datos es un compromiso entre la capacidad de representación y la complejidad computacional. Cada nivel de dimensionalidad tiene sus propias ventajas y desafíos, y la elección adecuada depende del contexto específico del problema y de los recursos disponibles. La comprensión de cómo las dimensiones impactan en los resultados es esencial para tomar decisiones informadas en análisis de datos y machine learning.

A medida que la dimensionalidad aumenta, la capacidad de comprender las relaciones entre las características se vuelve más compleja. La visualización directa de datos en espacios de alta dimensionalidad se vuelve prácticamente imposible.

El procesamiento y análisis de datos en dimensiones más altas requieren una capacidad computacional significativamente mayor. Los algoritmos pueden volverse más costosos en términos de tiempo y recursos.

5. GitHub:

- EDA_LAB/lab_01 at main · AntonyAroni/EDA_LAB (github.com)