**Escuela:** Sistemas de Información en línea

**Período académico:** Octubre – Marzo 2025

**Docente:** Marcelo Fabián Guato Burgos

**Asignatura:** Inteligencia Artificial 2-SIN-7A

**Actividad:** Aprendizaje Autónomo 1

**Tema: Manipulación de datos y herramientas para la IA**

**Nombre del estudiante:**

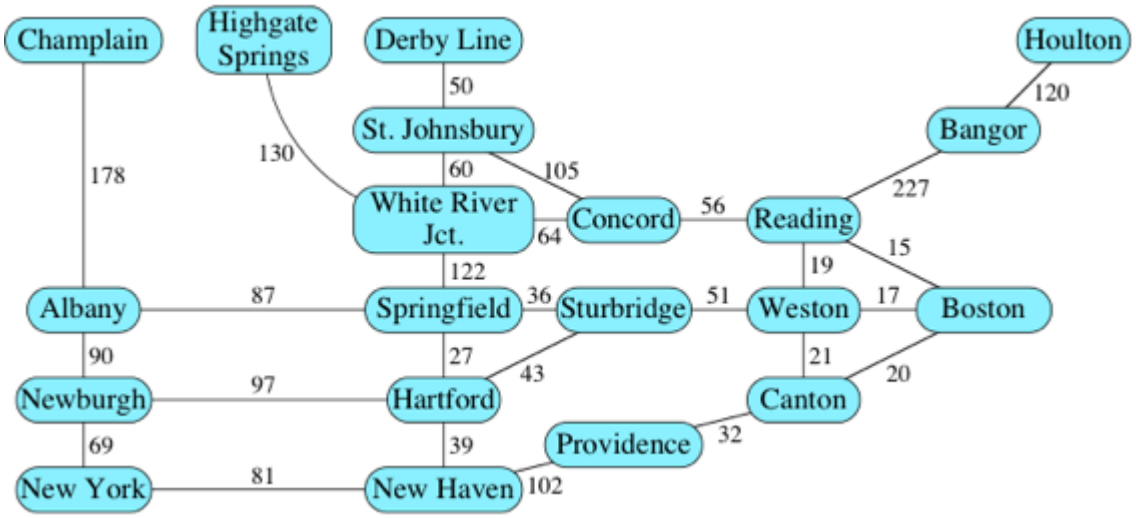
Sophia Monserrat Ibarra Jaramillo

UIDE
Powered by
Arizona State University

# Contenido

## Introducción

Este trabajo presenta un grafo ponderado que modela un mapa de carreteras del noreste de Estados Unidos. En este contexto, cada ciudad se trata como un nodo y cada carretera como una arista cuyo peso corresponde a la distancia entre ciudades. El enfoque toma como base los materiales de Khan Academy sobre descripción y representación de grafos, donde se expone el uso de pesos en aristas para reflejar distancias o costos en mapas de carreteras.

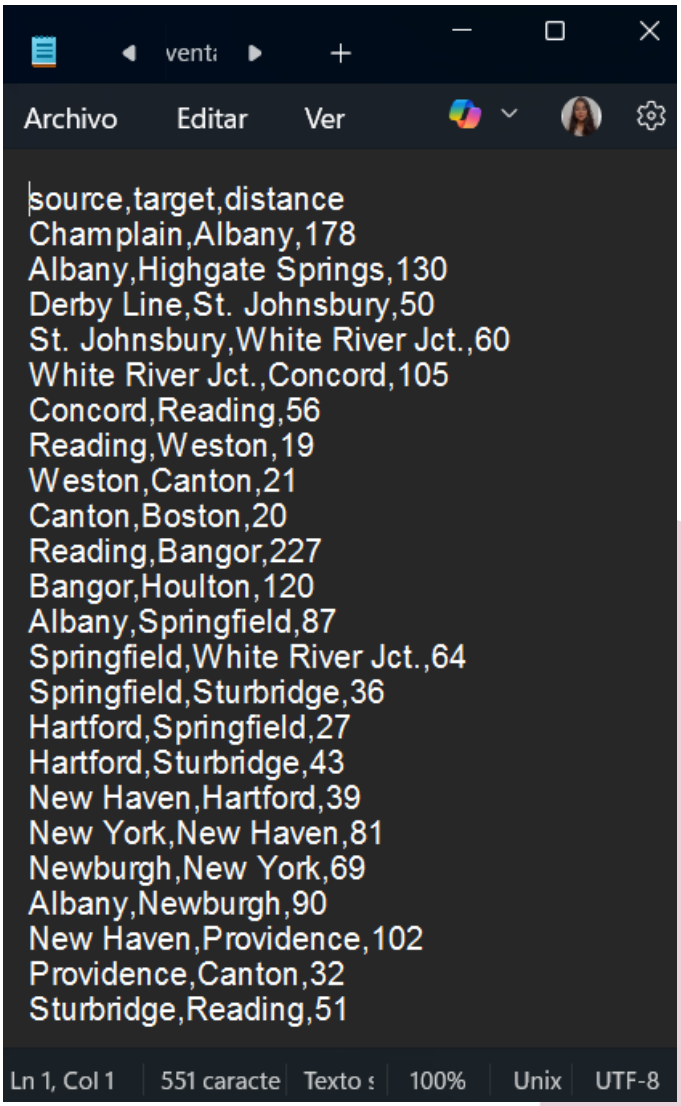



**Objetivos de la entrega**

- Construir un archivo CSV que represente el grafo de la Figura 1 (ciudades, conexiones y distancias).
- Desarrollar un script en Python que lea el CSV en un DataFrame, y dibuje el grafo con NetworkX, mostrando los nombres de las ciudades y los pesos de las aristas.
- Presentar una visualización clara y legible del grafo y dejar todo reproducible.

**Dataset** **(CSV)**

Como primer paso, se elaboró un archivo CSV que permite representar el grafo en formato textual. Para ello, se definió una estructura simple compuesta por tres columnas: source, target y distance. Cada fila del archivo describe una conexión entre dos ciudades, donde source indica la ciudad de origen, target la ciudad de destino y distance representa la distancia asociada a dicha conexión.



Esta estructura permite almacenar de forma clara la información del grafo y facilita su posterior procesamiento mediante herramientas de análisis de datos en Python.

**Programa en Python, DataFrame y visualización con NetworkX**

El programa desarrollado tiene como finalidad leer un archivo CSV que contiene información sobre rutas entre distintas ciudades y a partir de esos datos, construir y visualizar un grafo utilizando Python. Para lograrlo, se emplean las librerías Pandas, NetworkX y Matplotlib, las cuales permiten manipular datos, modelar grafos y generar representaciones gráficas.

Para empezar, se importan las librerías necesarias y se definen las rutas donde se encuentran el archivo CSV de entrada y la imagen de salida. Estas rutas se gestionan mediante la librería Pathlib, lo que facilita el trabajo con archivos y carpetas dentro del proyecto. También se define una variable que indica el tipo de distribución o diseño que se utilizará para dibujar el grafo, siendo el diseño manual el seleccionado para que la visualización sea similar a la imagen original propuesta en la actividad.

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
from pathlib import Path

DATA = Path(__file__).resolve().parents[1] / "data" / "rutas_khan.csv"
OUT = Path(__file__).resolve().parents[1] / "img" / "grafo_khan.png"
LAYOUT = "manual"  # 'manual' | 'kamada_kawai' | 'spring'
```

El programa lee el archivo CSV mediante la función read_csv() de Pandas. Durante este proceso, las columnas del archivo son convertidas a minúsculas para mantener uniformidad y se seleccionan únicamente las columnas necesarias: source, target y distance. De esta manera, se obtiene un DataFrame limpio que representa las conexiones entre ciudades y las distancias asociadas a cada una de ellas.

```
def cargar_csv(ruta: Path) -> pd.DataFrame:
    df = pd.read_csv(ruta)  # pandas.read_csv
    df = df.rename(columns=str.lower)
    return df[["source", "target", "distance"]]
```

El DataFrame es utilizado para construir el grafo. Se crea un grafo no dirigido en NetworkX y se recorren todas las filas del DataFrame. Por cada fila se añade una arista al grafo, donde las ciudades se representan como nodos y la distancia se guarda como un atributo de la arista. Esto permite que el grafo conserve la información cuantitativa presente en el dataset original.

```
def construir_grafo(df: pd.DataFrame) -> nx.Graph:
    G = nx.Graph()
    for _, r in df.iterrows():
        G.add_edge(r["source"], r["target"], weight=float(r["distance"]))
    return G
```

Una vez construido el grafo, se define la posición de los nodos. En el caso del diseño manual, se asignan coordenadas aproximadas a cada ciudad, basándose en la distribución observada en la imagen de referencia. Este paso es importante, ya que permite controlar la forma final del grafo y evitar una distribución automática que no respete la estructura visual original. Adicionalmente, el programa ofrece otras opciones de distribución automática como Kamada-Kawai y Spring Layout, aunque estas no se utilizan en la ejecución principal.

```
def posiciones(G: nx.Graph, layout: str) -> dict:
    layout = layout.lower()
    if layout == "manual":
        # Coordenadas aproximadas para parecerse a la imagen original (x, y):
        return {
```

```
        # bloque noreste (Maine)
        "Boston": (10.0, 0.0),
        "Reading": (8.6, 0.45),
        "Weston": (7.9, 0.2),
        "Canton": (7.8, -0.55),
        "Providence": (7.0, -0.9),
        "Sturbridge": (6.4, 0.45),

        # eje central
        "Springfield": (5.2, 0.3),
        "Hartford": (4.6, 0.0),
        "New Haven": (4.1, -0.65),
        "New York": (3.4, -1.1),
        "Newburgh": (3.7, -0.4),
        "Albany": (3.2, 0.45),

        # norte (Vermont/New Hampshire)
        "White River Jct.": (5.0, 1.0),
        "St. Johnsbury": (5.3, 1.6),
        "Derby Line": (5.45, 2.1),
        "Concord": (5.9, 0.85),

        # noroeste
        "Highgate Springs": (4.25, 1.45),
        "Champlain": (3.3, 2.0),

        # este lejano
        "Bangor": (9.1, 1.0),
        "Houlton": (10.6, 1.0),
    }

elif layout == "kamada_kawai":
```

```
        # Requiere SciPy. respetar longitudes relativas por peso.
        return nx.kamada_kawai_layout(G, weight="weight")
    elif layout == "spring":
        return nx.spring_layout(G, seed=7, weight="weight")
    else:
        raise ValueError("Layout no soportado.")


def dibujar(G: nx.Graph, pos: dict, salida: Path) -> None:
    plt.figure(figsize=(13, 6))
```

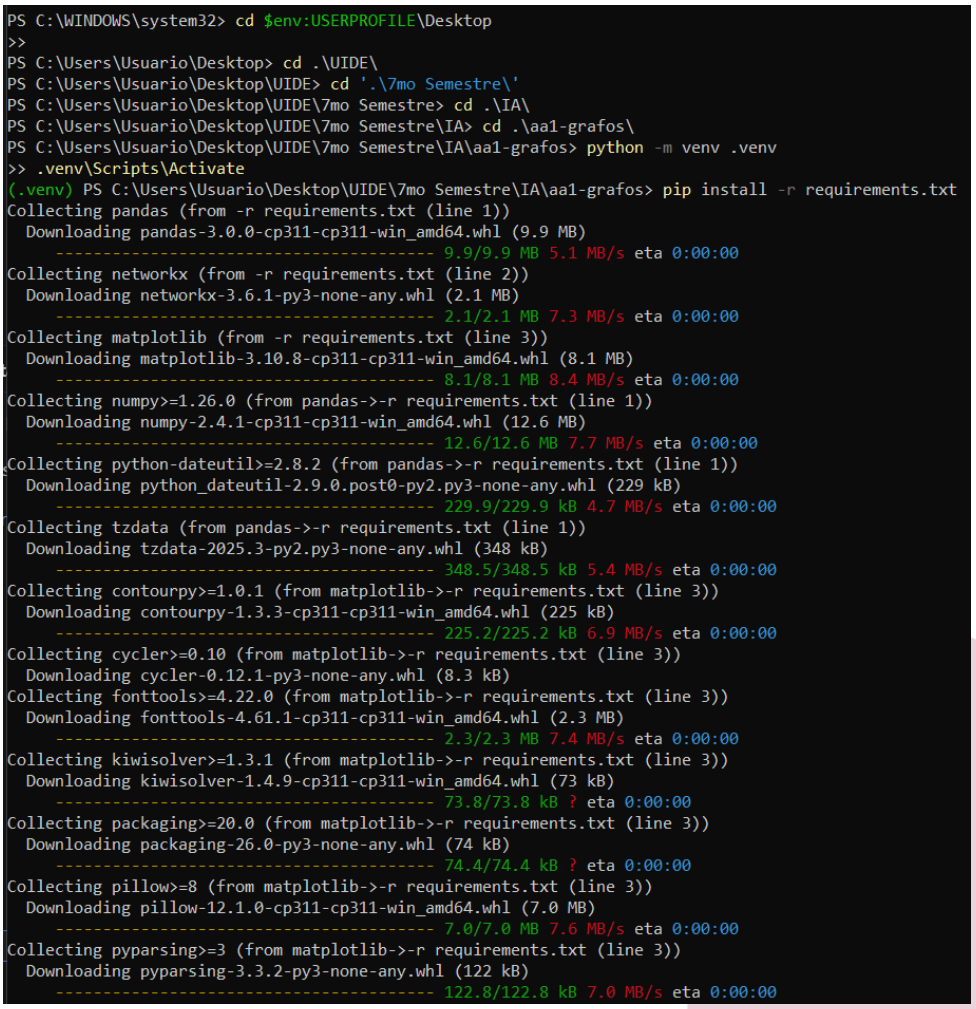
Finalmente, el grafo es dibujado utilizando Matplotlib. Primero se trazan las aristas para que no oculten los nodos, luego se dibujan los nodos con un color uniforme y bordes visibles, y después se agregan las etiquetas de cada ciudad. También se incluyen las etiquetas de las aristas, que corresponden a las distancias entre las ciudades. La imagen resultante se guarda automáticamente en la carpeta correspondiente y se muestra un mensaje indicando que el archivo fue creado correctamente.

```
    # Aristas primero para que no tapen los nodos
    nx.draw_networkx_edges(G, pos, width=1.6, edge_color="#444")

    # Nodos
    nx.draw_networkx_nodes(G, pos, node_color="#98e6f5", node_size=1550, edgecolors="#333")

    # Etiquetas de nodos (pequeños ajustes para el “cluster” Boston–Reading–Weston–Canton–Providence)
    pos_lab = pos.copy()
    for n, (dx, dy) in {
```

```
        "Reading": (0.0, 0.10),
        "Weston": (0.0, -0.07),
        "Canton": (0.0, -0.08),
        "Providence": (0.0, -0.08),
        "Sturbridge": (0.0, 0.08),
    }.items():
        if n in pos_lab:
            x, y = pos_lab[n]
            pos_lab[n] = (x + dx, y + dy)

    nx.draw_networkx_labels(G, pos_lab, font_size=9, font_weight="bold")

    # Etiquetas de pesos
    labels = nx.get_edge_attributes(G, "weight")
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels, font_size=8)

    plt.axis("off")
    salida.parent.mkdir(parents=True, exist_ok=True)
    plt.tight_layout()
    plt.savefig(salida, dpi=200)
    print(f"Imagen creada en: {salida}")


def main():
    df = cargar_csv(DATA)
    G = construir_grafo(df)
    pos = posiciones(G, LAYOUT)
    dibujar(G, pos, OUT)


if __name__ == "__main__":
    main()
```

**Ejecución del programa:**

Para ejecutar el programa, primero se verificó que las librerías necesarias estuvieran instaladas en el entorno de trabajo. Una vez comprobado esto, se ubicaron el archivo CSV y el script en las carpetas correspondientes del proyecto. Desde la terminal, se ejecutó el archivo Python, lo que permitió que el programa leyera automáticamente el archivo de datos, construyera el grafo y generara la imagen final.

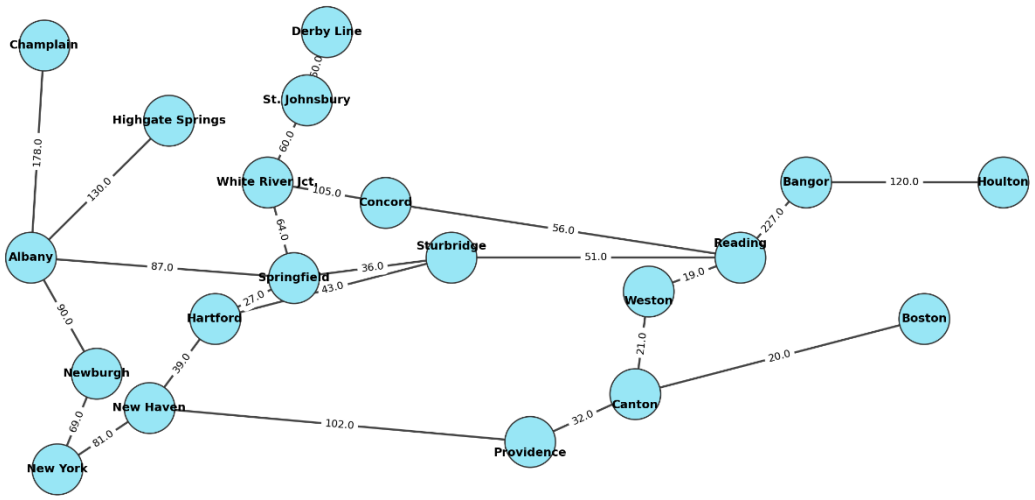



Al finalizar la ejecución, el programa guardó la imagen del grafo en la carpeta definida y mostró un mensaje de confirmación. Este proceso evidencia cómo un conjunto de datos estructurados

puede ser procesado y visualizado de manera eficiente usando Python, facilitando la comprensión de relaciones complejas como las representadas en un grafo.

```
(.venv) PS C:\Users\Usuario\Desktop\UIDE\7mo Semestre\IA\aa1-grafos> python src/dibujar_grafo.py
Imagen creada en: C:\Users\Usuario\Desktop\UIDE\7mo Semestre\IA\aa1-grafos\img\grafo_khan.png
```

**Resultados y análisis**




La imagen generada por el programa muestra una representación gráfica del grafo construido a partir del archivo CSV. Cada ciudad aparece como un nodo y cada ruta como una línea que conecta dos nodos. Los valores numéricos ubicados sobre las líneas representan las distancias entre las ciudades, permitiendo interpretar fácilmente el significado de cada conexión.

Aunque el diseño no es idéntico al de la imagen original, la distribución manual de los nodos permite mantener una estructura muy similar, donde se respetan los agrupamientos regionales y la orientación general del mapa. Esto demuestra que los datos almacenados en un archivo CSV pueden transformarse en una visualización clara y comprensible mediante el uso de herramientas adecuadas de programación.

## Conclusiones

Realizar este trabajo me permitió darme cuenta de que antes de hablar de inteligencia artificial es necesario entender bien cómo se organizan los datos. Transformar una imagen en un archivo CSV no fue solo copiar información, sino decidir qué elementos eran realmente importantes y cómo representarlos de manera clara. Esto hizo evidente que, si los datos no están bien estructurados desde el inicio, cualquier intento de análisis o visualización pierde sentido, sin importar las herramientas que se utilicen después.

Además, la visualización del grafo mostro que la forma en que se muestran los datos cambia la manera en que se entienden. Al ajustar manualmente la posición de los nodos, se observa que pequeños cambios pueden hacer que un grafo sea claro o confuso. Esto permitió reflexionar que en muchos casos la tecnología no falla, sino la forma en que las personas presentan y comunican la información.

## Referencias Bibliográficas:


Khan Academy. (s. f.). *Describir grafos*. Khan Academy. https://es.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/describing-graphs

NetworkX Developers. (s. f.). *Weighted graph drawing*. NetworkX Documentation. https://networkx.org/documentation/stable/auto_examples/drawing/plot_weighted_graph.html

NetworkX Developers. (s. f.). *draw_networkx_edge_labels*. NetworkX Documentation. https://networkx.org/documentation/stable/reference/generated/networkx.drawing.nx_pylab.draw_networkx_edge_labels.html

The pandas development team. (s. f.). *pandas.read_csv*. pandas Documentation. https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

Matplotlib Developers. (s. f.). *Pyplot summary*. Matplotlib Documentation. https://matplotlib.org/stable/api/pyplot_summary.html