

# Laboratorio 2 Introducción a la Inteligencia Artificial

David Alejandro Avendaño Rodríguez, Manuel Santiago Ibañez Gutierrez

2 de septiembre de 2025

## 1. Introducción

En la actualidad, el análisis de datos y su representación gráfica se han convertido en pilares fundamentales para la investigación científica, el desarrollo tecnológico y la toma de decisiones estratégicas. El volumen y la complejidad de la información que se genera día a día requieren no solo de herramientas que permitan su almacenamiento y procesamiento, sino también de entornos que faciliten su interpretación a través de visualizaciones claras, interactivas y escalables.

En este contexto, el ecosistema de librerías en Python orientado al análisis y visualización de datos ofrece un flujo de trabajo robusto y flexible. Dicho ecosistema está compuesto por herramientas especializadas en la manipulación de grandes volúmenes de información —como **pandas**, **dask** o **xarray**— que se complementan con librerías diseñadas para la representación gráfica —tales como **hvPlot**, **HoloViews**, **Datashader**, **Bokeh** y **Matplotlib**. Esta integración no solo agiliza el proceso de exploración y entendimiento de datos, sino que también habilita el desarrollo de soluciones interactivas mediante plataformas como **Streamlit**, orientadas a la comunicación efectiva de resultados.

El presente documento tiene como propósito describir y analizar este flujo de trabajo, detallando la función de cada una de las librerías involucradas y resaltando la importancia de su interoperabilidad. De esta manera, se busca ofrecer una visión global sobre cómo la combinación de estas herramientas constituye un marco eficiente para transformar datos en conocimiento útil y aplicable.

## Objetivos

### Objetivo general

Diseñar y documentar un flujo de trabajo en Python que integre librerías de análisis, visualización e interacción de datos, con el fin de demostrar cómo estas herramientas permiten transformar información cruda en resultados gráficos claros y útiles para la toma de decisiones.

### Objetivos específicos

1. Implementar la lectura, manipulación y preprocesamiento de datos mediante librerías de análisis como **pandas**, garantizando la correcta estructuración de la información.
2. Desarrollar visualizaciones estáticas e interactivas utilizando librerías como **hvPlot**, **HoloViews** y **Bokeh**, con el fin de facilitar la interpretación de patrones y tendencias.
3. Integrar las visualizaciones obtenidas en un entorno interactivo a través de **Streamlit**, permitiendo la exploración dinámica de los resultados por parte de los usuarios.

## 2. Primer Punto: Resumen de librerías de visualización y análisis de datos

Para tener una contextualización, se tiene en cuenta un ecosistema de librerías de Python orientadas al análisis y visualización de datos, organizadas en un flujo que integra fuentes de datos, representación intermedia y salidas gráficas interactivas. A continuación, siendo las principales herramientas:

- **pandas**: Biblioteca fundamental para la manipulación y análisis de datos estructurados en Python, basada en estructuras como **DataFrame**.
- **dask**: Extiende las capacidades de **pandas** y NumPy para el procesamiento distribuido y paralelo de grandes volúmenes de datos.
- **GeoPandas**: Especializada en el manejo y análisis de datos espaciales, integrando geometrías y operaciones geoespaciales sobre **DataFrames**.
- **RAPIDS**: Conjunto de bibliotecas aceleradas por GPU, diseñadas para análisis de datos y aprendizaje automático en gran escala.
- **xarray**: Orientada al manejo de arreglos multidimensionales (n-dimensiones), con soporte para datos etiquetados, común en ciencias del clima y datos geoespaciales.
- **ibis**: Proporciona una capa de abstracción para consultas analíticas sobre distintos motores de bases de datos, facilitando la portabilidad de código.
- **NetworkX**: Herramienta especializada en el análisis y modelado de grafos y redes complejas.
- **DuckDB**: Motor de base de datos embebido, optimizado para consultas analíticas directamente sobre archivos de datos.
- **polars**: Alternativa moderna a **pandas**, optimizada en rendimiento mediante paralelización y uso intensivo de memoria eficiente.

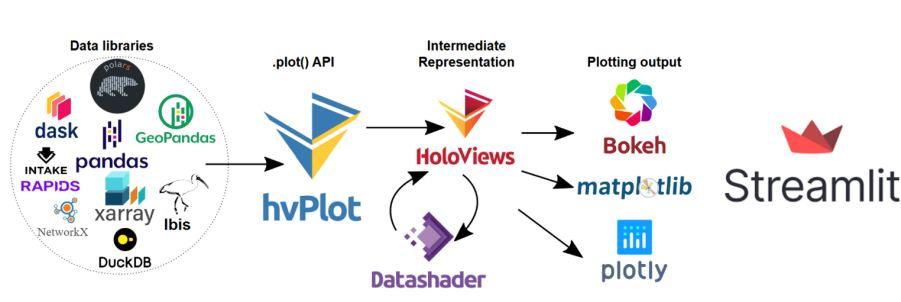


Figura 1: Interpretación ramificada de las librerías participantes.

Todas estas bibliotecas pueden integrarse mediante la API unificada **hvPlot**, la cual convierte los datos en representaciones intermedias a través de **HoloViews** y **Datashader**. Estas representaciones permiten generar salidas gráficas en librerías de visualización como **Bokeh**, **Matplotlib**, **Plotly** e incluso desplegarse en **Streamlit** para aplicaciones interactivas.

### 3. Segundo punto: Resumen de tesis acerca de agentes inteligentes

Un agente inteligente es una entidad, ya sea software o hardware, que tiene la capacidad de percibir su entorno mediante sensores y actuar sobre él a través de actuadores. Su principal objetivo es alcanzar metas específicas de manera autónoma. Para ello, debe tomar decisiones racionales, adaptarse a cambios en su entorno, interactuar con otros agentes o con usuarios, y ejecutar acciones que maximicen sus posibilidades de éxito.

El campo de potencial artificial es una técnica empleada en la planificación de trayectorias para robots móviles y agentes autónomos. Se basa en la idea de representar el espacio de movimiento como un campo de fuerzas. El objetivo ejerce una fuerza de atracción, que guía al agente hacia la meta. Los obstáculos generan una fuerza de repulsión, que evita colisiones. El movimiento del agente surge de la combinación de estas fuerzas, logrando que avance hacia su objetivo mientras esquivo los obstáculos. Este método es eficiente y sencillo de implementar, aunque puede presentar limitaciones como la aparición de mínimos locales que impiden llegar a la meta.

El modelo BDI es un enfoque de razonamiento para agentes inteligentes inspirado en la forma en que los humanos toman decisiones, ya sea la información que el agente posee sobre su entorno y su estado interno. Deseos, los objetivos o estados que el agente pretende alcanzar. Las intenciones, como los planes y acciones que el agente se compromete a ejecutar para cumplir sus deseos, considerando sus creencias.

Este enfoque permite que el agente no solo reaccione a estímulos externos, sino que también planifique, seleccione y ejecute acciones de manera deliberada. Por ello, se utiliza ampliamente en planificación de trayectorias, sistemas multiagente y robótica autónoma.

## 4. Tercer punto: Solución al algoritmo BDI

El script implementa un agente inteligente basado en el modelo BDI (Beliefs–Desires–Intentions) para la planificación de trayectorias en un entorno con obstáculos. El objetivo es que el agente navegue desde una posición inicial hasta un punto meta, evitando colisiones y resolviendo situaciones de atasco mediante mecanismos adaptativos. El entorno se modela utilizando un campo de potencial artificial, compuesto por dos componentes, una es el potencial atractivo, este atrae al agente hacia el objetivo, aumentando en función de la distancia al mismo y el potencial repulsivo, que repele al agente de los obstáculos, disminuyendo su influencia conforme aumenta la distancia. La combinación de estas fuerzas define la dirección de movimiento en condiciones normales.

El ciclo de razonamiento del agente se ejecuta, donde en cada iteración: Actualiza sus creencias, genera un deseo según su estado, planifica las intenciones a partir de ese deseo, propone un nuevo movimiento, verificando colisiones con obstáculos. En caso de colisión, aplica un pequeño desvío aleatorio para evitar bloqueos.

La simulación utiliza matplotlib para representar la trayectoria del agente (línea azul discontinua), su posición actual (círculo rojo), el objetivo (cruz verde). los obstáculos (cuadrados negros). los puntos de atasco detectados (estrellas naranjas) y una gráfica en paralelo que muestra la evolución de la energía potencial, lo cual permite evaluar si el agente avanza efectivamente hacia un mínimo global (la meta).

Gracias a esta combinación de campo de potencial artificial y modelo BDI, el agente logra desplazarse de forma autónoma hacia el objetivo, es capaz de detectar y registrar situaciones de atasco, implementa estrategias de escape y evitación basadas en su memoria de entornos problemáticos, mantiene un ciclo deliberativo que le permite alternar entre navegación, escape y evasión según el contexto.

En conjunto, el script demuestra un sistema de planificación de trayectorias inspirado en razonamiento humano, donde el agente no solo reacciona al entorno, sino que también planifica y adapta sus acciones para alcanzar de forma robusta el objetivo.

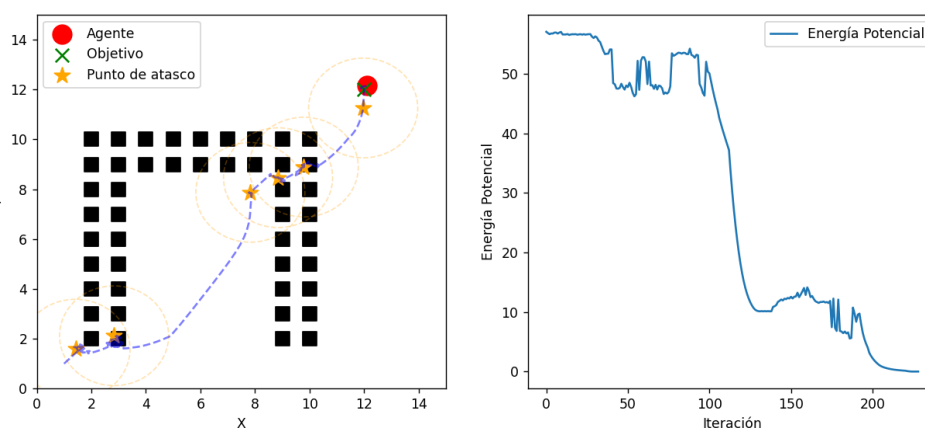


Figura 2: Interpretación gráfica del algoritmo BDI.

El gráfico de la izquierda representa el recorrido seguido por el agente dentro del entorno con obstáculos en forma de herradura. El agente (círculo rojo) partió desde la esquina inferior izquierda y avanzó hacia el objetivo (cruz verde), ubicado en la parte superior derecha. Durante el trayecto, se identificaron múltiples puntos de atasco (estrellas naranjas), los cuales corresponden a zonas donde

el agente experimentó dificultades para avanzar debido a la configuración de los campos de potencial artificial. En dichos puntos, el modelo BDI aplicó mecanismos de escape que le permitieron superar los bloqueos locales y retomar la trayectoria hacia la meta.

El gráfico de la derecha muestra la evolución de la energía potencial a lo largo de las iteraciones. Se observa un comportamiento inicial con oscilaciones relativamente altas, reflejando la presencia de atascos y movimientos de corrección en la trayectoria. Posteriormente, la energía desciende de forma significativa, evidenciando que el agente logra escapar de las trampas locales y aproximarse al objetivo de manera más estable. Finalmente, el valor de la energía converge hacia cero, indicando que el agente alcanza con éxito la posición final deseada.

En conjunto, ambos gráficos confirman que el algoritmo BDI, en combinación con campos de potencial artificial, es capaz de guiar al agente hasta el objetivo, aun cuando este enfrenta zonas de estancamiento que, de no contar con dicho mecanismo, podrían impedir la convergencia de la trayectoria.

## Conclusión

El ecosistema representado en la figura muestra cómo la interacción entre librerías de análisis de datos y herramientas de visualización ofrece un flujo de trabajo integral y altamente eficiente. En la etapa inicial, bibliotecas como **pandas**, **dask**, **xarray**, **GeoPandas**, entre otras, proporcionan estructuras y operaciones avanzadas para la manipulación de grandes volúmenes de información, incluyendo datos distribuidos, geospaciales y en distintos formatos. Posteriormente, mediante la API unificada de **hvPlot**, es posible generar representaciones intermedias que permiten abstraer la complejidad técnica, garantizando una transición fluida hacia **HoloViews** y **Datashader**. Estas últimas no solo potencian la escalabilidad de los gráficos, sino que también aseguran una adecuada representación de conjuntos masivos de datos.

La versatilidad del ecosistema se hace evidente en la etapa final, donde se integran librerías de visualización consolidadas como **Bokeh**, **Matplotlib** y **Plotly**, cada una con fortalezas particulares en gráficos interactivos, personalización y compatibilidad multiplataforma. Finalmente, herramientas de despliegue como **Streamlit** permiten transformar estos análisis en aplicaciones accesibles e interactivas, cerrando el ciclo desde la manipulación de datos hasta su presentación en entornos colaborativos y de fácil acceso.

En conjunto, este flujo refleja la importancia de la interoperabilidad entre librerías, mostrando cómo la integración de diferentes niveles —desde la adquisición y procesamiento de datos hasta su visualización y distribución— contribuye al desarrollo de soluciones más ágiles, escalables y orientadas a la toma de decisiones basada en datos.

## 5. Script

Listing 1: Algoritmo BDI

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib.animation import FuncAnimation
4
5 K_atractivo = 0.5
6 K_repulsivo = 3.0
7 radio_repulsion = 3.0
8
9 # Función para calcular el campo de potencial
10 def calcular_potencial(posicion_agente, objetivo, obstaculos, epsilon=0.25):
11     # Potencial de atracción hacia el objetivo
12     potencial_atractivo = 0.5 * K_atractivo * np.linalg.norm(objetivo -
13                                                                posicion_agente)**2
14
15     # Potencial de repulsión de los obstáculos
16     potencial_repulsivo = 0
17     for obstaculo in obstaculos:
```

```

17     distancia = np.linalg.norm(posicion_agente - obstaculo)
18     if 1 <= distancia < radio_repulsion:
19         potencial_repulsivo += 0.5 * K_repulsivo * (1 / distancia - 1 /
20             radio_repulsion)**2
21     elif epsilon < distancia < 1:
22         potencial_repulsivo += 0.5 * K_repulsivo * (1 / distancia - 1 /
23             radio_repulsion)
24     else:
25         potencial_repulsivo += 0
26
27     # Potencial total
28     return potencial_atractivo + potencial_repulsivo
29
30 # Funci n para calcular gradiente del campo de potencial
31 def calcular_gradiente(posicion_agente, objetivo, obstaculos, epsilon=0.15):
32     gradiente = np.zeros(2)
33     for i in range(2):
34         delta_pos = np.zeros(2)
35         delta_pos[i] = epsilon
36         potencial_pos = calcular_potencial(posicion_agente + delta_pos,
37             objetivo, obstaculos)
38         potencial_neg = calcular_potencial(posicion_agente - delta_pos,
39             objetivo, obstaculos)
40         gradiente[i] = (potencial_pos - potencial_neg) / (2 * epsilon)
41     return gradiente
42
43 # Clase para el agente BDI
44 class AgenteBDI:
45     def __init__(self, posicion_inicial, objetivo, obstaculos):
46         self.posicion = posicion_inicial
47         self.objetivo = objetivo
48         self.obstaculos = obstaculos
49         self.estado = "navegando"
50         self.contador_atascado = 0
51         self.historial_posiciones = [posicion_inicial.copy()]
52         self.direccion_escape = None
53         self.contador_escape = 0
54         self.puntos_atasco = []
55         self.contador_evitacion = 0
56         self.direccion_evitacion = None
57
58     def actualizar_creencias(self):
59         if len(self.historial_posiciones) > 10:
60             desplazamiento = np.linalg.norm(self.historial_posiciones[-1] -
61                 self.historial_posiciones[-10])
62             if desplazamiento < 0.5:
63                 self.estado = "atascado"
64                 self.contador_atascado += 1
65                 es_nuevo_atasco = True
66                 for punto in self.puntos_atasco:
67                     if np.linalg.norm(self.posicion - punto) < 1.0:
68                         es_nuevo_atasco = False
69                         break
70                 if es_nuevo_atasco:
71                     self.puntos_atasco.append(self.posicion.copy())
72             else:
73                 self.estado = "navegando"
74                 self.contador_atascado = 0
75
76         if np.linalg.norm(self.posicion - self.objetivo) < 0.5:
77             self.estado = "completado"

```

```

74     for punto in self.puntos_atasco:
75         if np.linalg.norm(self.posicion - punto) < 2.0:
76             self.estado = "evitando"
77             break
78
79     def generar_deseos(self):
80         if self.estado == "completado":
81             return "permanecer"
82         elif self.estado == "atascado" and self.contador_atascado > 5:
83             return "escapar"
84         elif self.estado == "evitando":
85             return "evitar"
86         else:
87             return "navegar"
88
89     def planificar_intenciones(self, deseo):
90         if deseo == "permanecer":
91             return np.zeros(2)
92         elif deseo == "escapar":
93             punto_mas_cercano = None
94             distancia_minima = float('inf')
95             for punto in self.puntos_atasco:
96                 distancia = np.linalg.norm(self.posicion - punto)
97                 if distancia < distancia_minima:
98                     distancia_minima = distancia
99                     punto_mas_cercano = punto
100             if punto_mas_cercano is not None:
101                 direccion_alejamiento = self.posicion - punto_mas_cercano
102                 if np.linalg.norm(direccion_alejamiento) < 0.1:
103                     angulo = np.random.uniform(0, 2*np.pi)
104                     direccion_alejamiento = np.array([np.cos(angulo), np.sin(
105                         angulo)])
106                 else:
107                     direccion_alejamiento = direccion_alejamiento / np.linalg.
108                         norm(direccion_alejamiento)
109                 direccion_objetivo = self.objetivo - self.posicion
110                 if np.linalg.norm(direccion_objetivo) > 0:
111                     direccion_objetivo = direccion_objetivo / np.linalg.norm(
112                         direccion_objetivo)
113                 direccion_combinada = 0.6 * direccion_alejamiento + 0.4 *
114                     direccion_objetivo
115                 direccion_combinada = direccion_combinada / np.linalg.norm(
116                     direccion_combinada)
117                 self.direccion_escape = direccion_combinada
118                 self.contador_escape = 0
119             else:
120                 if self.direccion_escape is None or self.contador_escape > 20:
121                     angulo = np.random.uniform(0, 2*np.pi)
122                     self.direccion_escape = np.array([np.cos(angulo), np.sin(
123                         angulo)])
124                     self.contador_escape = 0
125                 else:
126                     self.contador_escape += 1
127             return self.direccion_escape * 0.3
128         elif deseo == "evitar":
129             puntos_cercanos = []
130             for punto in self.puntos_atasco:
131                 if np.linalg.norm(self.posicion - punto) < 2.0:
132                     puntos_cercanos.append(punto)
133             direccion_evitacion = np.zeros(2)
134             for punto in puntos_cercanos:
135                 direccion_punto = self.posicion - punto

```

```

130         if np.linalg.norm(direccion_punto) > 0:
131             direccion_punto = direccion_punto / np.linalg.norm(
                direccion_punto)
132             peso = 1.0 / (0.1 + np.linalg.norm(self.posicion - punto))
133             direccion_evitacion += peso * direccion_punto
134         if np.linalg.norm(direccion_evitacion) > 0:
135             direccion_evitacion = direccion_evitacion / np.linalg.norm(
                direccion_evitacion)
136         direccion_objetivo = self.objetivo - self.posicion
137         if np.linalg.norm(direccion_objetivo) > 0:
138             direccion_objetivo = direccion_objetivo / np.linalg.norm(
                direccion_objetivo)
139         direccion_combinada = 0.5 * direccion_evitacion + 0.5 *
            direccion_objetivo
140         if np.linalg.norm(direccion_combinada) > 0:
141             direccion_combinada = direccion_combinada / np.linalg.norm(
                direccion_combinada)
142         return direccion_combinada * 0.15
143     else:
144         return -calcular_gradiente(self.posicion, self.objetivo, self.
            obstaculos) * 0.1
145
146     def ejecutar(self):
147         """Ejecuta un ciclo completo BDI"""
148         self.actualizar_creencias()
149         deseo = self.generar_deseos()
150         movimiento = self.planificar_intenciones(deseo)
151
152         # Proponer nueva posici n
153         nueva_posicion = self.posicion + movimiento
154
155         # Verificaci n de colisi n con obst culos
156         colision = False
157         for obstaculo in self.obstaculos:
158             if np.linalg.norm(nueva_posicion - obstaculo) < 0.5: # margen de
                seguridad
159                 colision = True
160                 break
161
162         if not colision:
163             self.posicion = nueva_posicion
164         else:
165             # Si choca, prueba un peque o desv o aleatorio
166             angulo = np.random.uniform(0, 2*np.pi)
167             self.posicion += 0.1 * np.array([np.cos(angulo), np.sin(angulo)])
168
169         self.historial_posiciones.append(self.posicion.copy())
170         return self.posicion, calcular_potencial(self.posicion, self.objetivo,
            self.obstaculos)
171
172
173     # Configuraci n inicial de la visualizaci n
174     fig, (ax, ax2) = plt.subplots(1, 2, figsize=(12, 5))
175
176     agente_posicion = np.array([1.0, 1.0])
177     objetivo = np.array([12, 12])
178     obstaculos = np.array([[2, 2],[2, 3], [2, 4],[2, 5],[2, 6],[2, 7],[2, 8],[2,
        9], [2, 10],[10, 10],
179                             [10, 9],[10, 8],[10, 7],[10, 6],[10, 5],[10, 4],[10,
        3],[10, 2],
180                             [9, 2], [9, 3],[9, 4],[9, 5],[9, 6],[9, 7],[9, 8],[9,
        9],

```

```

181         [3, 9],[3, 8],[3, 7],[3, 6],[3, 5],[3, 4],[3, 3], [3,
182             2],
183         [4,9],[5,9],[6,9],[7,9],[8,9],[3,10],[4,10],[5,10],[6,10],
184         [7,10],[8,10],[9,10]))
185 agente = AgenteBDI(agente_posicion, objetivo, obstaculos)
186 historial_energia_potencial = []
187
188 def update(frame):
189     global historial_energia_potencial
190     nueva_posicion, energia_potencial = agente.ejecutar()
191     historial_energia_potencial.append(energia_potencial)
192
193     ax.clear()
194     ax.scatter(nueva_posicion[0], nueva_posicion[1], color='red', marker='o',
195               s=200, label='Agente')
196     ax.scatter(objetivo[0], objetivo[1], color='green', marker='x', s=100,
197               label='Objetivo')
198
199     for obstaculo in obstaculos:
200         ax.scatter(obstaculo[0], obstaculo[1], color='black', marker='s', s
201                   =100)
202
203     for i, punto in enumerate(agente.puntos_atasco):
204         ax.scatter(punto[0], punto[1], color='orange', marker='*', s=150,
205                   label='Punto de atasco' if i == 0 else "")
206         circulo = plt.Circle((punto[0], punto[1]), 2.0, color='orange', fill=
207                               False, alpha=0.3, linestyle='--')
208         ax.add_patch(circulo)
209
210     historial = np.array(agente.historial_posiciones)
211     ax.plot(historial[:, 0], historial[:, 1], 'b--', alpha=0.5)
212
213     ax.set_xlim(0, 15)
214     ax.set_ylim(0, 15)
215     ax.set_title(f'Agente BDI - Estado: {agente.estado}')
216     ax.set_xlabel('X')
217     ax.set_ylabel('Y')
218     ax.legend()
219
220     ax2.clear()
221     ax2.plot(historial_energia_potencial, label='Energ a Potencial')
222     ax2.set_title('Evoluci n de la Energ a Potencial')
223     ax2.set_xlabel('Iteraci n')
224     ax2.set_ylabel('Energ a Potencial')
225     ax2.legend()
226
227 ani = FuncAnimation(fig, update, frames=500, interval=200)
228 plt.show()

```

## Referencias

- [1] S. Russell y P. Norvig, *Artificial Intelligence: A Modern Approach*, 4ta edici3n, Pearson, 2021.
- [2] I. Goodfellow, Y. Bengio, y A. Courville, *Deep Learning*, MIT Press, 2016. Disponible en: <https://www.deeplearningbook.org>
- [3] N. S. Nise, *Control Systems Engineering*, 7ma edici3n, Wiley, 2015.



- [4] Guido van Rossum y Python Software Foundation, *Python Language Reference*, versión 3.11, 2023.  
Disponible en: <https://www.python.org>
- [5] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python”, *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.