INSTITUTO TECNOLÓGICO DE CULIACÁN

INGENIERÍA EN SISTEMAS COMPUTACIONALES

ASIGNATURA:

INTELIGENCIA ARTIFICIAL

DOCENTE:

ZURIEL DATHAN MORA FELIX

DOCUMENTACION

TAREA:

SISTEMA DE RECOMENDACIÓN EN LA INDUSTRIA RESTAURANTERA

ALUMNOS:

FLORES BELTRAN JESUS GUADALUPE

JOSÉ MANUEL URÍAS ESCOBAR

FECHA DE ENTREGA:

12/10/2025

Introducción:

Para la documentación de este proyecto lo que hicimos fue ir describiendo el paso por paso, para así ir conociendo el tema, mientras vamos desarrollando, porque es algo nuevo para nosotros.

Las librerías que utilizamos son las siguientes:

- Pandas: para manejo de datos con dataframes(tablas), lo usamos para cargar los csv.
- Numpy: para utilización de arreglos, operaciones vectorizadas, etc.)
- Streamlit: framework para construir la UI web en Python pues nos pareció un poco mas sencillo su implementación con todo lo demas.
- PGMPY: libraría de modelos gráficos probabilísticos. La usamos para definir la red bayesiana. (nodos de gustos, rasgos, afinidades)

Paso 1:

En el restaurante modelo se manejan diferentes tipos de platillos que van desde opciones omnívoras hasta vegetarianas y veganas. Cada plato está hecho con varios ingredientes que determinan su sabor, si es picante, dulce o si lleva mariscos. También se considera si los ingredientes son de origen animal, si causan alguna alergia o si están disponibles en el momento.
Con esta información, el sistema puede saber qué platillos puede recomendar a cada cliente según sus gustos, alergias o tipo de dieta, adaptando las sugerencias de forma más personalizada y realista.

Paso 2:

Usaremos la red bayesiana por platillo, se realizara un filtro de (dietas, alergias y disponibilidad). Si el platillo falla en eso, no entra.

Se calcula la afinidad del usuario con el plato y devuelve una puntuación (alto, medio, bajo) o probable, que usaremos para posicionar.

Pero primero investigamos que es una red bayesiana, lo cual es lo siguiente: un modelo probabilístico que representa variables y sus dependencias con un grafo dirigido acíclico. Cada nodo es una variable Emm por ejemplo si le gusta lo picante, y las aristas indican dependencia.

La librería que usamos fue PGMPy

Variables del modelo (por plato)

- **Preferencias del usuario:** GustaPicante, GustaDulce, GustaMarisco (binarias: Sí/No).
- **Rasgos del plato:** RasgoPicante, RasgoDulce, RasgoMarisco (derivados del catálogo; binarios).
- **Afinidades intermedias:** AfinidadPicante, AfinidadDulce, AfinidadMarisco (valores: Alta/Media/Baja).

- **Afinidad total:** AfinidadTotal (Alta/Media/Baja), agregada a partir de las tres afinidades.
- **Restricciones como evidencia determinista:** PlatoCompatibleDieta (Sí/No), PlatoSeguroAlergenos (Sí/No), PlatoDisponible (Sí/No).
- **Salida del modelo:** Recomendable (Sí/No).

Y utilizamos las siguiente estructura de conexiones principales

- GustaPicante → AfinidadPicante ← RasgoPicante
- GustaDulce → AfinidadDulce ← RasgoDulce
- GustaMarisco → AfinidadMarisco ← RasgoMarisco
- AfinidadPicante, AfinidadDulce, AfinidadMarisco → AfinidadTotal
- PlatoCompatibleDieta, PlatoSeguroAlergenos, PlatoDisponible, AfinidadTotal → Recomendable

Las tablas de probabilidad son las siguientes:

Afinidades: si al usuario le gusta el rasgo y el plato lo tiene, la afinidad tiende a **Alta**; si no le gusta y el plato sí lo tiene, tiende a **Baja**.

AfinidadTotal: combina las tres afinidades con una CPD agregadora (si la mayoría son Altas, sube).

Recomendable: si alguno es compatible con su dieta, y es seguro de los alergenos y si esta disponible entonces la probabilidad de recomendación depende de afinidad o sea si al usuario le gusta el ingrediente del plato.

El proceso de razonamiento es el siguiente:

Entrada de datos del usuario: dieta, alergias, gustos (picante/dulce/marisco).

Clasificaciones en tiempo real: disponibilidad por plato, alergenos por ingrediente del plato.

Filtro: genera datos elegibles.

Luego la evidencia de la red bayesiana.

```
evidencia = {
    "GustaPicante": "Si" if perfil["gusta_picante"]==1 else "No",
    "RasgoPicante": "Si" if int(p["picante"])==1 else "No",
    "GustaDulce": "Si" if perfil["gusta_dulce"]==1 else "No",
    "RasgoDulce": "Si" if int(p["dulce"])==1 else "No",
    "GustaMarisco": "Si" if perfil["gusta_marisco"]==1 else "No",
```

"RasgoMarisco": "Si" if int(p["marisco"])==1 else "No",

"PlatoCompatibleDieta": comp_dieta,

"PlatoSeguroAlergenos": "Si" if not (

Luego la rankea dependiendo de si el plato está disponible o no.

Paso 3:

Incorporamos el razonamiento no monotono, en ese las recomendaciones pueden cambiar cuanto entra nueva información. A diferencia del monotono, aquí no es todo fijo: si cambian el inventario, las alergias o dietas del cliente, el sistema no muestra los platos que antes parecían ser correctos y añade otros nuevos o quita.

¿Como se implementa?

Aplicamos una serie de filtros antes del ranking:

1. Dieta (Vegana/vegetariana/omnívora).
2. Alergias (gluten, lácteos, nueces) detectadas por los alergenos de cada ingrediente.
3. Y la disponibilidad: el plato esta disponible solo si todos sus ingredientes están disponibles.
4. Después de el filtro, la red bayesiana calcula la afinidad (picante, dulce, marisco) y da la probabilidad de que el usuario guste de la recomendación.
5. Si llegan nuevos datos (por ejemplo si se agota algún ingrediente o el usuario marca alergia al gluten), se recalcula.

Ejemplo: si el tomate y pasta de trigo se marca como no disponible, la pasta al pomodoro queda con plato_disponible=0 y desaparece de las recomendaciones.

Si el usuario activa la alergia al gluten, cualquier plato que use ingrediente alergeno “gluten” se excluye.

Si cambia la dieta a “vegana”, solo pasan platos etiquetados como veganos.

En nuestro proyecto, el comportamiento no monotono (las recomendaciones cambian cuando hay nueva evidencia) se implementa directamente en la clase app_streamlit.py y en la función filtrar_platos de la clase filtro.py.

Se recalcula la disponibilidad por plato (and de ingredientes), Si un solo ingrediente pasa a no disponible, el plato completo se marca como no disponible.

ReCÁLCULO EN TIEMPO REAL de 'plato_disponible' (AND lógico)

disp_rt = (

  cargar_relacion()

  .merge(inventario_sim, on="id_ingrediente", how="left")

```
    .assign(disponible=lambda df: pd.to_numeric(df["disponible"],
errors="coerce").fillna(0).astype(int))
    .groupby("id_plato")["disponible"].min()   # AND sobre los ingredientes del plato
    .astype(int)
    .reset_index()
    .rename(columns={"disponible": "plato_disponible_rt"})
)
platos = (
  platos.drop(columns=["plato_disponible"], errors="ignore")
      .merge(disp_rt, on="id_plato", how="left")
)
platos["plato_disponible"] = platos["plato_disponible_rt"].fillna(0).astype(int)
platos = platos.drop(columns=["plato_disponible_rt"])
```

Recalculo de alergenos (gluten, lácteos, nueces)

A partir de la relación plato-ingrediente, se vuelve a derivar si un plato contiene alergenos.

```
pi_rt = cargar_relacion().merge(
  ingredientes[["id_ingrediente", "alergenos"]], on="id_ingrediente", how="left"
)
def _to_list(x):
  if pd.isna(x) or str(x).strip() == "":
    return []
  return [s.strip().lower() for s in str(x).split("|")]

pi_rt["alergenos_list_rt"] = pi_rt["alergenos"].apply(_to_list)
pi_rt["has_gluten_rt"]  = pi_rt["alergenos_list_rt"].apply(lambda L: "gluten"  in L)
pi_rt["has_lacteos_rt"] = pi_rt["alergenos_list_rt"].apply(lambda L: "lacteos" in L)
pi_rt["has_nueces_rt"]  = pi_rt["alergenos_list_rt"].apply(lambda L: "nueces"  in L)
```

```
aler_rt = pi_rt.groupby("id_plato").agg(
  plato_contiene_gluten=("has_gluten_rt", "max"),
  plato_contiene_lacteos=("has_lacteos_rt", "max"),
  plato_contiene_nueces=("has_nueces_rt", "max"),
).reset_index()
platos = (
  platos.drop(columns=["plato_contiene_gluten","plato_contiene_lacteos","plato_contiene_nueces"], errors="ignore")
      .merge(aler_rt, on="id_plato", how="left")
)
for col in ["plato_contiene_gluten","plato_contiene_lacteos","plato_contiene_nueces"]:
  platos[col] = platos[col].fillna(False).astype(int)
```

los filtros que expulsan platos según dieta/alergias/disponibilidad se implementa en la clase filtro.py, antes de rankear aplicamos un filtro determinista

```
def filtrar_platos(platos, perfil, aplicar_disponibilidad=True):
  trazas = []
  elegibles = []
  for _, p in platos.iterrows():
    razones_out = []

    # Seguridad (alérgenos)
    if perfil.get("alergia_gluten",0)==1 and int(p.get("plato_contiene_gluten",0))==1:
      razones_out.append(("seguridad","Contiene gluten"))
    if perfil.get("alergia_lacteos",0)==1 and int(p.get("plato_contiene_lacteos",0))==1:
      razones_out.append(("seguridad","Contiene lácteos"))
    if perfil.get("alergia_nueces",0)==1 and int(p.get("plato_contiene_nueces",0))==1:
      razones_out.append(("seguridad","Contiene nueces"))

    # Dieta
    dieta = str(perfil.get("dieta","omnivora")).strip().lower()
    tipo = str(p.get("tipo_dieta","")).strip().lower()
```

```
    if dieta == "vegana" and tipo != "vegana":
      razones_out.append(("dieta","No compatible con dieta vegana"))
    elif dieta == "vegetariana" and tipo not in ("vegana","vegetariana"):
      razones_out.append(("dieta","No compatible con dieta vegetariana"))

    # Disponibilidad
    if aplicar_disponibilidad and int(p.get("plato_disponible",0)) != 1:
      razones_out.append(("disponibilidad","Ingrediente(s) no disponible(s)"))
    if len(razones_out)==0:
      elegibles.append(p.to_dict())
    else:
      trazas.append({"id_plato": p.get("id_plato"), "nombre": p.get("nombre"), "exclusion": razones_out})
  return pd.DataFrame(elegibles), trazas
```

Evidencias deterministas de la red bayesiana:

Esta toma como evidencias los nodos que ya vienen del filtro o derivaciones de compatibilidad de dieta, seguridad por alergenos y disponibilidad), y solo si todo es “si” la disponibilidad puede ser alta

```
evidencia = {
  "GustaPicante": "Si" if perfil["gusta_picante"]==1 else "No",
  "RasgoPicante": "Si" if int(p["picante"])==1 else "No",
  "GustaDulce": "Si" if perfil["gusta_dulce"]==1 else "No",
  "RasgoDulce": "Si" if int(p["dulce"])==1 else "No",
  "GustaMarisco": "Si" if perfil["gusta_marisco"]==1 else "No",
  "RasgoMarisco": "Si" if int(p["marisco"])==1 else "No",
  "PlatoCompatibleDieta": comp_dieta,              # ← determinista
  "PlatoSeguroAlergenos": "Si" if ... else "No",   # ← determinista
  "PlatoDisponible": "Si" if int(p["plato_disponible"])==1 else "No",  # ← determinista
}
p_rec = inferir_p_recomendable(infer, evidencia)
```

Y se aplica un rerun automático por cambios en los widgets, cada cambio del usuario (inventario, alergias, dieta, gustos) aplica un rerun que vuelve a cargar el script con el nuevo estado, haciendo que lo anterior se recalcule:

```
st.multiselect(
    "Ingredientes NO disponibles",
    options=opciones_ids,
    format_func=lambda iid: f"{iid} — {ingr_nombres.get(iid, iid)}",
    key="k_ids_no_disp"
)  # al cambiar, Streamlit rerun -> se recalculan banderas y filtro
```

Paso 4, ejemplo de uso:

	id_plato	nombre	tipo_dieta	categoria	picante	dulce	marisco	plato_contiene
0	P01	Pasta al pomodoro	vegetariana	fuerte	0	0	0	
1	P02	Camarones al ajillo	omnivora	fuerte	1	0	1	
2	P03	Hummus con verduras	vegana	entrada	0	0	0	
3	P04	Mousse de chocolate	vegetariana	postre	0	1	0	
4	P05	Tacos de soya estilo pastor	vegana	fuerte	1	0	0	
5	P06	Ensalada mediterránea	vegana	entrada	0	0	0	
6	P07	Sopa de tomate	vegetariana	entrada	0	0	0	
7	P08	Filete de pescado a la plancha	omnivora	fuerte	0	0	1	
8	P09	Brownie vegano	vegana	postre	0	1	0	
9	P10	Curry de garbanzo	vegana	fuerte	1	0	0	

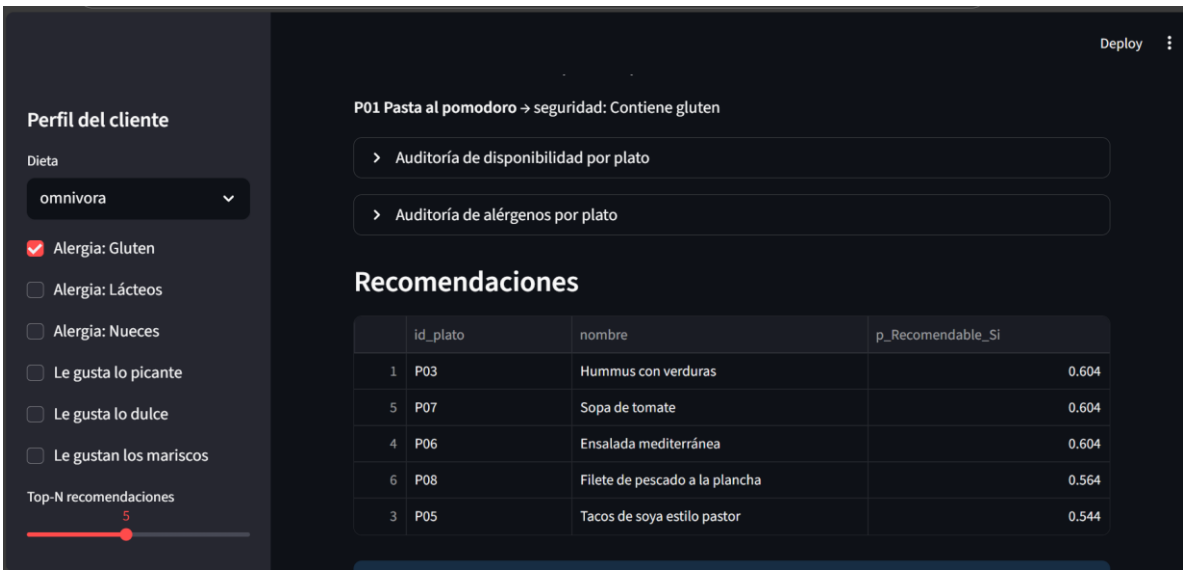
Esta es la interfaz de el programa, muestra el perfil del cliente, donde se puede elegir la dieta, alergias y si le gusta algo, y muestra un resumen del catalogo que hay.

	id_plato	nombre	p_Recomendable_Si
0	P01	Pasta al pomodoro	0.604
2	P03	Hummus con verduras	0.604
6	P07	Sopa de tomate	0.604
5	P06	Ensalada mediterránea	0.604
7	P08	Filete de pescado a la plancha	0.564

Compatibilidad de dieta, alérgenos y disponibilidad se usan como evidencias deterministas dentro de la Red Bayesiana. Cambios en inventario o perfil pueden retraer recomendaciones (razonamiento no monótono).

Y aquí muestra las recomendaciones, dependiendo de el perfil del usuario.

Ahora hice unos pequeños cambios para ver los resultados



Como se puede ver, se elimino la pasta, porque pues el perfil del cliente marco alergia al gluten.

Ya por ultimo Resumimos los siguientes puntos para explicar rápidamente que utilizamos y como planeamos el diseño de este proyecto:

Diseño conceptual

- **Objetivo:** recomendar platos según **perfil del cliente** (dieta, alergias, gustos) y **estado del inventario**.
- **Arquitectura general:**
    - **UI:** Streamlit (sidebar para entradas, tablas de resultados y expanders de auditoría).
    - **Capa de reglas:** verificación de **dieta**, **alérgenos** y **disponibilidad** por plato.
    - **Capa probabilística: Red Bayesiana** (PGMPy) para calcular la probabilidad de recomendación usando afinidades (picante, dulce, marisco) y evidencias deterministas.
    - **No monótono:** cualquier cambio en entradas recalcula y puede retirar/añadir platos.

Diseño técnico

- **Datos:** CSV de platos, ingredientes, platos_ingredientes, inventario.
- **Derivaciones en tiempo real:**
    - plato_disponible = AND de disponibilidad de sus ingredientes (si uno falta, el plato no está disponible).
    - plato_contiene_gluten/lacteos/nueces a partir de los alérgenos de los ingredientes.
- **Filtro:** función filtrar_platos(...) en filtro.py excluye por dieta/alérgenos/disponibilidad y genera **trazas**.
- **Red Bayesiana:** nodos de gustos/rasgos → afinidades → afinidad total → Recomendable. Evidencias deterministas: PlatoCompatibleDieta, PlatoSeguroAlergenos, PlatoDisponible.
- **Inferencia:** VariableElimination para obtener P(Recomendable=Sí) por plato.
- **Ordenamiento:** ranking por probabilidad.