Resumen – Defensa Trabajo Integrador de Seminario de Python

```python
import streamlit as st
import os
import sys
import pandas as pd
import matplotlib.pyplot as plt
import json
import folium
from streamlit_folium import st_folium
import io
```

`streamlit`: permite crear la interfaz web.

- `os` y `sys`: permiten trabajar con rutas de archivos y modificar el `sys.path` para importar módulos de otras carpetas.
- `pandas`: para manejo de archivos CSV y estructuras tipo DataFrame.
- `matplotlib.pyplot`: para generar gráficos.
- `json`, `folium`, `st_folium`: están importados pero no se usan en esta sección (seguro los usan otras páginas).
- `io`: se menciona para evitar modificar funciones originales (aunque acá tampoco se usa directamente).

```python
BASE_DIR = os.path.abspath(os.path.join(os.path.dirname(__file__), '..', '..'))
SRC_DIR = os.path.join(BASE_DIR, 'src')

if SRC_DIR not in sys.path:
    sys.path.append(SRC_DIR)
```

Esto permite que aunque estés ejecutando desde `/pages/`, puedas importar archivos desde la carpeta `/src`.

```python
archivo_ind = RESULTADOS_PATH / "individuos_completo.csv"
try:
    arch = pd.read_csv(archivo_ind, sep=";", encoding="utf-8")
except FileNotFoundError:
    st.error("No se encontró el archivo 'individuos_completo.csv'.")
    arch = None
except Exception as e:
    st.error(f"Ocurrió un error: {e}")
    arch = None
```

Carga el CSV principal. Si falla, muestra error con `st.error()` y pone `arch = None`.

PUNTO 1 – Distribución por grupos de edad y sexo

```python
if arch is not None:
    if "ANO4" not in arch or "TRIMESTRE" not in arch or "CH04" not in arch or "CH06" not in arch:
        st.error("Faltan columnas requeridas en el archivo CSV.")
    else:
        st.header("Distribución de la población por grupos de edad (cada 10 años) y sexo")
```

Control de calidad: Asegura que existan las columnas necesarias.

```
años_disponibles = sorted([int(a) for a in arch["ANO4"].dropna().unique()])
año = st.selectbox("Seleccione el año", años_disponibles)

trimestres_disponibles = sorted([int(t) for t in arch[arch["ANO4"] == año]["TRIMESTRE"].dropna
trimestre = st.selectbox("Seleccione el trimestre", trimestres_disponibles)
```

Filtro por año y trimestre con `selectbox`, usando valores únicos y ordenados. SelectBOX permite al usuario seleccionar **una única opción** de una **lista desplegable**

valor_elegido = st.selectbox("Texto que ve el usuario", lista_de_opciones)

"Texto que ve el usuario" → es el label que aparece encima del desplegable.

lista_de_opciones → es la lista de opciones que puede elegir el usuario.

valor_elegido → es la opción seleccionada por el usuario. Podés usarla después para filtrar o mostrar resultados.

```
arch_filtrado = arch[(arch["ANO4"] == año) & (arch["TRIMESTRE"] == trimestre)]
arch_filtrado = arch_filtrado[arch_filtrado["CH06"] >= 0]
```

Filtrado de datos para el año y trimestre seleccionados, y edades válidas. (>0)

```
arch_filtrado["Grupo Edad"] = ((arch_filtrado["CH06"] // 10) * 10).astype(int)
arch_filtrado = arch_filtrado[arch_filtrado["Grupo Edad"] <= 100]
```

Agrupamiento por décadas: edades 0–9, 10–19, ..., hasta 100.

```
distribucion = arch_filtrado.groupby(["Grupo Edad", "CH04"]).size().unstack(fill_value=0)
distribucion.columns = ["Varón", "Mujer"] if 1 in distribucion.columns else distribucion.colum
```

Agrupamiento final: por grupo de edad y sexo. Se renombran columnas si aparecen los valores 1 y 2 (normalmente 1 = varón, 2 = mujer).

```
fig, ax = plt.subplots(figsize=(10, 6))
distribucion.plot(kind="bar", ax=ax)
ax.set_title("Distribución por grupo de edad y sexo")
ax.set_xlabel("Grupo de edad")
ax.set_ylabel("Cantidad de personas")
ax.legend(title="Sexo")

st.pyplot(fig)
```

- Crea una figura (`fig`) y un conjunto de ejes (`ax`) usando `matplotlib (es una **librería de visualización de datos en Python**.)`
- `figsize=(10, 6)` define el **tamaño** del gráfico: 10 de ancho y 6 de alto (en pulgadas).
- Le pide al DataFrame `distribucion` que se grafique como un **gráfico de barras** (`kind="bar"`).
- Usa los ejes que creamos antes (`ax=ax`), en vez de crear otros nuevos.

- Este DataFrame debe tener como índice los grupos de edad, y como columnas los sexos (por ejemplo: "Varón", "Mujer").
- Asigna un **título al gráfico**
- Etiqueta el **eje X** con "Grupo de edad".
- Etiqueta el **eje Y** con "Cantidad de personas".
- Muestra una **leyenda** (cuadro de colores que identifica a cada sexo) con el título "Sexo".
- Muestra el gráfico `fig` en la aplicación Streamlit.

PUNTO 2 – Edad promedio de personas por aglomerado (ultimo TRI/AÑO)

```
ult_anio = arch["ANO4"].max()
ult_trimestre = arch[arch["ANO4"] == ult_anio]["TRIMESTRE"].max()
```

Se obtiene el **último año y trimestre disponibles**.

```
arch_con_filtro = arch[(arch["ANO4"] == ult_anio) & (arch["TRIMESTRE"] == ult_trimestre)]

edad_promedio = arch_con_filtro.groupby("AGLOMERADO")["CH06"].mean().sort_index()
edad_promedio.index = edad_promedio.index.astype(int).astype(str)
```

Filtra los datos del último período y **calcula la edad promedio por aglomerado**.

`arch_con_filtro.groupby("AGLOMERADO")`:

- Agrupa los datos por aglomerado (una especie de región o área urbana que define la EPH).
- Esto permite calcular cosas por cada grupo, en este caso, **por cada aglomerado**.
- Selecciona solo la columna `CH06`, que representa la **edad** de cada persona.
- El .mean calcula el promedio de la edad
- `.sort_index()`: Ordena los resultados por el número de aglomerado (que es el índice de la serie generada).
- El resultado es una **Serie de pandas** donde: Las claves (índice) son los códigos de los aglomerados y los valores son las **edades promedio**.
- edad_promedio.index = edad_promedio.index.astype(int).astype(str) ¿Qué hace? 👉 Convierte el índice de la Serie edad_promedio:

  .astype(int) fuerza a que los aglomerados estén en formato numérico (por si quedaron como strings).

  .astype(str) los vuelve a pasar a texto.

```
info, ax = plt.subplots(figsize=(10, 6))
edad_promedio.plot(kind="bar", ax=ax)
ax.set_title(f"Edad promedio por aglomerado - Año {int(ult_anio)}, Trimestre {int(ult_trimestr
ax.set_xlabel("Aglomerado")
ax.set_ylabel("Edad promedio")

st.pyplot(info)
```

Gráfico de barras: muestra edad promedio por aglomerado.

PUNTO 3 – Evolución de la dependencia demográfica

```
st.header("Evolución de la dependencia demográfica por aglomerado")

aglomerados = sorted([int(a) for a in arch["AGLOMERADO"].dropna().unique()])
select_aglom = st.selectbox("Seleccione un aglomerado", sorted(aglomerados))
```

Se permite **elegir un aglomerado**. `arch["AGLOMERADO"]` → selecciona la columna que contiene los códigos de aglomerados (por ejemplo, 2, 8, 22, etc.).

1. `.dropna()` → elimina los valores que estén vacíos (NaN), por si hay registros incompletos.
2. `.unique()` → obtiene **solo los valores distintos**. No queremos repetir el mismo aglomerado varias veces.
3. `[int(a) for a in ...]` → convierte cada valor a `int`, por si estaban como strings o floats (ej. "2.0" → 2).
4. `sorted(...)` → ordena la lista de aglomerados de menor a mayor (por ejemplo, [2, 4, 6, 8, 12]).

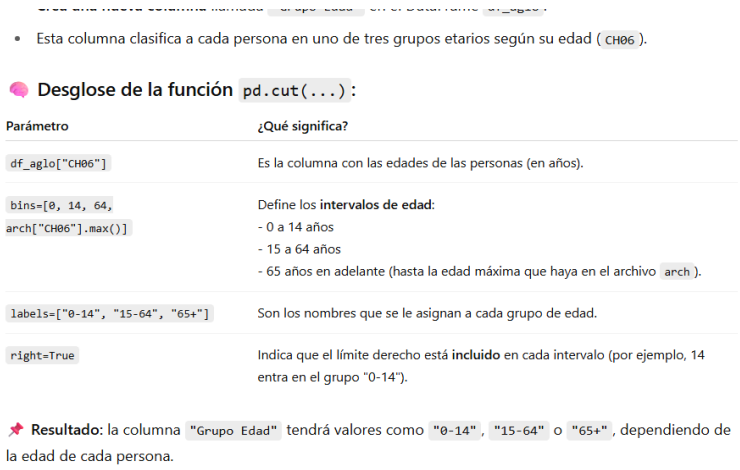
📌 **Resultado final:** una lista ordenada de aglomerados sin duplicados, lista para ser usada en el selector.

```
df_aglo = arch[arch["AGLOMERADO"] == select_aglom]
df_aglo["Grupo Edad"] = pd.cut(df_aglo["CH06"], bins=[0, 14, 64, arch["CH06"].max()], labels=[
```

Filtra por aglomerado y clasifica edades en tres grupos: menores, población activa y mayores.

Filtra el DataFrame `arch` (que contiene los datos completos de la EPH) para **quedarse solo con los registros del aglomerado que seleccionó el usuario** en el `selectbox`.

Resultado: `df_aglo` es un nuevo DataFrame que contiene **solo los datos del aglomerado seleccionado**.



- Esta columna clasifica a cada persona en uno de tres grupos etarios según su edad (`CH06`).

🧠 **Desglose de la función `pd.cut(...)`:**

Parámetro	¿Qué significa?
`df_aglo["CH06"]`	Es la columna con las edades de las personas (en años).
`bins=[0, 14, 64, arch["CH06"].max()]`	Define los **intervalos de edad**: - 0 a 14 años - 15 a 64 años - 65 años en adelante (hasta la edad máxima que haya en el archivo `arch`).
`labels=["0-14", "15-64", "65+"]`	Son los nombres que se le asignan a cada grupo de edad.
`right=True`	Indica que el límite derecho está **incluido** en cada intervalo (por ejemplo, 14 entra en el grupo "0-14").

📌 **Resultado:** la columna `"Grupo Edad"` tendrá valores como `"0-14"`, `"15-64"` o `"65+"`, dependiendo de la edad de cada persona.

```
grupo = df_aglo.groupby(["ANO4", "TRIMESTRE", "Grupo Edad"]).size().unstack(fill_value=0)
grupo["Dependencia Demográfica"] = ((grupo["0-14"] + grupo["65+"]) / grupo["15-64"]) * 100
```

Agrupa por año-trimestre y calcula el **índice de dependencia demográfica**.

Agrupa el DataFrame `df_aglo` (filtrado por aglomerado) según tres columnas:

- `ANO4`: el año
- `TRIMESTRE`: el trimestre (1, 2, 3 o 4)
- `Grupo Edad`: la categoría etaria de cada persona (ya creada con `pd.cut` como "0-14", "15-64", "65+")
- **`.size()`** cuenta cuántas personas hay en cada combinación de (año, trimestre, grupo de edad).
- **`.unstack()`** "desapila" la última columna del groupby (que es `"Grupo Edad"`) y la convierte en columnas:
- Se genera una tabla donde las filas representan combinaciones de año y trimestre, y las columnas son `"0-14"`, `"15-64"` y `"65+"`.
- **`fill_value=0`**: Si no hay personas en algún grupo etario para un trimestre, se pone 0 en vez de dejar `NaN`

LA SEGUNDA FILA:

Calcula la tasa de dependencia demográfica para cada fila (es decir, para cada trimestre).

1. La fórmula es:

Dependencia Demogra´fica=(Poblacio´n de 0-14 + Poblacio´n de 65+Poblacio´n de 15-64)×100\text{Dependencia Demográfica} = \left(\frac{\text{Población de 0-14 + Población de 65+}}{\text{Población de 15-64}} \right) \times 100Dependencia Demogra´fica=(Poblacio´n de 15-64Poblacio´n de 0-14 + Poblacio´n de 65+)×100

3. Esta tasa **indica cuántas personas "dependientes" (niños y adultos mayores)** hay por cada 100 personas en edad activa (15-64 años).

4. El resultado se guarda en una **nueva columna llamada `"Dependencia Demográfica"`**.

```
grupo["Periodo"] = grupo.index.map(lambda x: f"{int(x[0])}-T{int(x[1])}")

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(grupo["Periodo"], grupo["Dependencia Demográfica"], marker="o")
ax.set_title(f"Evolución de la dependencia demográfica para el aglomerado seleccionado --> {se
ax.set_xlabel("Periodo")
ax.set_ylabel("Dependencia demográfica")
plt.xticks(rotation=45)

st.pyplot(fig)
```

Gráfico de línea: muestra cómo cambia la dependencia a lo largo del tiempo en ese aglomerado.

PUNTO 4 – Media y mediana edad por año y trimestre

```
st.subheader("Media y Mediana de Edad por Año y Trimestre")

info_edad = arch.groupby(["ANO4", "TRIMESTRE"])["CH06"].agg(Media="mean", Mediana="median").r
st.dataframe(info_edad)
```

Agrupa por año-trimestre y calcula la **media y mediana** de edad. Luego la muestra en una **tabla interactiva**.

Agrupa el DataFrame `arch` por año (`ANO4`) y trimestre (`TRIMESTRE`).

- Sobre la columna `"CH06"` (que representa la edad de las personas), aplica dos funciones estadísticas:
- `"mean"` → calcula el **promedio (media)** de edad.
- `"median"` → calcula la **mediana** de edad.
- Con `.agg(Media="mean", Mediana="median")`:
- Se generan dos columnas nuevas: una llamada `"Media"` y otra `"Mediana"` (estos son nombres personalizados).
- `.reset_index()`:
- Convierte los índices de agrupación (`ANO4`, `TRIMESTRE`) en columnas normales para que sea más fácil mostrar el resultado como una tabla.

La línea st.dataframe (info_Edad) Muestra el DataFrame `info_edad` en pantalla usando el componente **`st.dataframe`**, que permite ver y explorar la tabla con scroll y ordenamiento por columnas.

```
info_edad["Periodo"] = info_edad.apply(lambda x: f"{int(x['ANO4'])}-T{int(x['TRIMESTRE'])}", a

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(info_edad["Periodo"], info_edad["Media"], marker='o', color='blue', label='Media')
ax.plot(info_edad["Periodo"], info_edad["Mediana"], marker='s', color='orange', label='Mediana
ax.set_title("Media y Mediana de Edad por Año y Trimestre")
ax.set_xlabel("Año y Trimestre")
ax.set_ylabel("Edad")
ax.legend()
plt.xticks(rotation=45)
st.pyplot(fig)
```

Gráfico de línea: muestra evolución de edad media y mediana en el tiempo.

Crea una nueva columna llamada `"Periodo"` que junta el año (`ANO4`) y el trimestre (`TRIMESTRE`) en un solo string con formato `"AÑO-Ttrimestre"`, por ejemplo: `"2022-T3"`.

- Se usa `.apply(..., axis=1)` para aplicar la función **fila por fila** (`axis=1` significa que la función se aplica sobre filas, no sobre columnas).

Dibuja una **línea azul** con puntos (`marker='o'`) que representa cómo varía la **edad promedio (media)** a lo largo del tiempo (por período).

Dibuja otra **línea naranja** con cuadrados (`marker='s'`) para representar la evolución de la **mediana** de edad.

plt.xticks(rotation=45): Rota 45 grados las etiquetas del eje X para que los períodos tipo `"2022-T3"` no se superpongan.

🎯 ¿Qué significa cada uno?

- `marker='o'` → círculo
- `marker='s'` → cuadrado (de *square* en inglés)

📚 ¿Qué es pandas?

`pandas` es una librería de Python especializada en **manipulación y análisis de datos**. La estructura principal que usa es el `DataFrame`, que permite manejar datos tabulares (como una tabla de Excel).

✅ Utilizaciones de `pandas` en tu código:

1. Filtrar datos con condiciones

```
python
CopiarEditar
arch_con_filtro = arch[(arch["ANO4"] == ult_anio) & (arch["TRIMESTRE"] ==
ult_trimestre)]
```

👉 Usás `pandas` para **filtrar filas** del DataFrame `arch` según el año y trimestre más reciente.

---

2. Agrupar y calcular promedio (mean)

```
python
CopiarEditar
edad_promedio = arch_con_filtro.groupby("AGLOMERADO")["CH06"].mean()
```

👉 Agrupás por `AGLOMERADO` y calculás el **promedio de edad (CH06)** para cada grupo.

---

3. Modificar el índice

```
python
CopiarEditar
edad_promedio.index = edad_promedio.index.astype(int).astype(str)
```

👉 Convertís el índice (que es el código de aglomerado) a `int` y luego a `str` para poder mostrarlo mejor (por ejemplo, en un gráfico o selectbox).

---

4. Eliminar nulos y obtener valores únicos

```
python
CopiarEditar
arch["AGLOMERADO"].dropna().unique()
```

👉 Eliminás los valores nulos de la columna `AGLOMERADO` y obtenés los **aglomerados únicos** (sin repetir).

---

5. Filtrar por aglomerado

```
python
CopiarEditar
df_aglo = arch[arch["AGLOMERADO"] == select_aglom]
```

👉 Filtrás el DataFrame para quedarte solo con los datos del aglomerado seleccionado.

---

6. Crear columna por rango de edad

```
python
CopiarEditar
```

```
df_aglo["Grupo Edad"] = pd.cut(df_aglo["CH06"], bins=[0, 14, 64,
arch["CH06"].max()], labels=["0-14", "15-64", "65+"])
```

👉 Usás `pandas.cut()` para **clasificar edades en rangos** (grupos etarios), una operación muy típica en análisis demográfico.

---

7. Agrupar por múltiples columnas y contar

```
python
CopiarEditar
grupo = df_aglo.groupby(["ANO4", "TRIMESTRE", "Grupo
Edad"]).size().unstack(fill_value=0)
```

👉 Agrupás por año, trimestre y grupo de edad, contás cuántos casos hay en cada uno, y con `unstack()` convertís eso en tabla.

---

8. Crear columna derivada (dependencia demográfica)

```
python
CopiarEditar
grupo["Dependencia Demográfica"] = ((grupo["0-14"] + grupo["65+"]) / grupo["15-
64"]) * 100
```

👉 Calculás un nuevo indicador combinando columnas existentes.

---

9. Agregar resumen estadístico (mean y median)

```
python
CopiarEditar
info_edad = arch.groupby(["ANO4", "TRIMESTRE"])["CH06"].agg(Media="mean",
Mediana="median")
```

👉 Usás `agg()` para sacar varias estadísticas en una sola operación (media y mediana de edad por periodo).

---

10. Resetear índice para convertirlo en columnas

```
python
CopiarEditar
info_edad = info_edad.reset_index()
```

👉 Dejás de usar `ANO4` y `TRIMESTRE` como índice, y los pasás a columnas para que sea más fácil mostrarlo o graficarlo.

---

11. Crear columna nueva combinando otras

```
python
CopiarEditar
info_edad["Periodo"] = info_edad.apply(lambda x: f"{int(x['ANO4'])}-
T{int(x['TRIMESTRE'])}", axis=1)
```

👉 Usás `apply()` para construir una columna con formato tipo "2024-T2", útil para ejes de tiempo.