Universidad Autónoma de San Luis Potosí
Facultad de ingeniería
Inteligencia Artificial Aplicada
Practica: 3
Nombre Práctica: Conceptos Básicos de Pandas
Nombre del Alumno: Manuel Ramírez Galván

Fecha: 12/02/2025

Procedimiento

3.1.- Inicie Jupyter Notebooks y abra los notebooks "fundamentos", “agregación” e “indexado” proporcionados.
3.2.- Siga las instrucciones en los notebooks para explorar los conceptos básicos de Pandas.
3.3.- Resuelva los ejercicios proporcionados en el notebook “ejercicios”.

Resultados

Fundamentos

En esta parte se trabajara con el dataset ***homelessness***, este es un dataset que contiene estimaciones de personas sin hogar en cada estado de EE. UU. en 2018.

La columna "individual" es el número de personas sin hogar que no forman parte de una familia con niños.

La columna "family_members" es el número de personas sin hogar que forman parte de una familia con niños.

La columna "state_pop" es la población total del estado.

importar pandas como ***pd***

```
import pandas as pd    Import "pandas" could not be resolved from source
```
✓ 0.0s

Imagen 1.- Librería pandas

Importar el archivo *homelessness.csv*

```
df = pd.read_csv("C:/Users/HUAWEI/Desktop/L IA/Manual/Practica3_CONCEPTOS BASICOS DE PANDA/datasets/homelessness.csv")
```
✓ 0.0s

Imagen 2.- Importación del Archivo

Utilice las funciones ***head() y tail()*** para visualizar los datos

```
print("Primeras 5 filas:")
df.head()
```
✓ 0.0s

Primeras 5 filas:

	Unnamed: 0	region	state	individuals	family_members	state_pop
0	0	East South Central	Alabama	2570.0	864.0	4887681
1	1	Pacific	Alaska	1434.0	582.0	735139
2	2	Mountain	Arizona	7259.0	2606.0	7158024
3	3	West South Central	Arkansas	2280.0	432.0	3009733
4	4	Pacific	California	109008.0	20964.0	39461588

```
print("Ultimas 5 filas")
df.tail()
```
✓ 0.0s

Ultimas 5 filas

	Unnamed: 0	region	state	individuals	family_members	state_pop
46	46	South Atlantic	Virginia	3928.0	2047.0	8501286
47	47	Pacific	Washington	16424.0	5880.0	7523869
48	48	South Atlantic	West Virginia	1021.0	222.0	1804291
49	49	East North Central	Wisconsin	2740.0	2167.0	5807406
50	50	Mountain	Wyoming	434.0	205.0	577601

Imagen 3.- Visualización de los datos

Imprima las columnas y el tipo de dato de cada columna utilizando la función ***info()***

```
df.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Unnamed: 0      51 non-null     int64
 1   region          51 non-null     object
 2   state           51 non-null     object
 3   individuals     51 non-null     float64
 4   family_members  51 non-null     float64
 5   state_pop       51 non-null     int64
dtypes: float64(2), int64(2), object(2)
memory usage: 2.5+ KB
```

Imagen 4.- Tipos de datos de cada columna

Imprima las ***columnas*** del dataset

```
print("Columnas del DataFrame")
df.columns
```

✓ 0.0s

```
Columnas del DataFrame

Index(['Unnamed: 0', 'region', 'state', 'individuals', 'family_members',
       'state_pop'],
      dtype='object')
```

Imagen 5.- Nombre de las columnas

Ordenar y filtrar Datos

Ordene el dataset *homelessness* por *individuals*, de menor a mayor, guarde el sesultado en una variable llamada *homelessness_ind*.

Imprima los primeros elementos del resultado.

```
homelessness_ind = df.sort_values(by='individuals')
print("Primeros elementos ordenados por 'individuals':")
homelessness_ind.head()
```

✓ 0.0s

```
Primeros elementos ordenados por 'individuals':
```

	Unnamed: 0	region	state	individuals	family_members	state_pop
50	50	Mountain	Wyoming	434.0	205.0	577601
34	34	West North Central	North Dakota	467.0	75.0	758080
7	7	South Atlantic	Delaware	708.0	374.0	965479
39	39	New England	Rhode Island	747.0	354.0	1058287
45	45	New England	Vermont	780.0	511.0	624358

Imagen 6.- Datos de menor a mayor por “individuals”

Ordene el dataset *homelessness* primero por *region (ascendente)* y luego por *family_members (descendente)*, guarde el sesultado en una variable llamada *homelessness_reg_fam*.

Imprima los primeros elementos del resultado.

```
homelessness_reg_fam = df.sort_values(by=['region', 'family_members'], ascending=[True, False])
print("Primeros elementos ordenados por 'region' y 'family_members':")
homelessness_reg_fam.head()
```

✓ 0.0s

```
Primeros elementos ordenados por 'region' y 'family_members':
```

	Unnamed: 0	region	state	individuals	family_members	state_pop
13	13	East North Central	Illinois	6752.0	3891.0	12723071
35	35	East North Central	Ohio	6929.0	3320.0	11676341
22	22	East North Central	Michigan	5209.0	3142.0	9984072
49	49	East North Central	Wisconsin	2740.0	2167.0	5807406
14	14	East North Central	Indiana	3776.0	1482.0	6695497

Imagen 7.- Datos ordenados por "región" y "family_members"

Crea un DataFrame llamado *state_fam* que contenga únicamente la columna *state* y *family_members*.

```
state_fam = df[['state', 'family_members']]
state_fam.head()
```

✓ 0.0s

	state	family_members
0	Alabama	864.0
1	Alaska	582.0
2	Arizona	2606.0
3	Arkansas	432.0
4	California	20964.0

Imagen 8.- Creación de DataFrame

De *state_fam* Obtenga todos los renglones donde *family_members* sea mayor o igual a 2000.

```
f_family_members = state_fam[state_fam['family_members'] >= 2000]
f_family_members
```

✓ 0.0s

	state	family_members
2	Arizona	2606.0
4	California	20964.0
5	Colorado	3250.0
8	District of Columbia	3134.0
9	Florida	9587.0
10	Georgia	2556.0
11	Hawaii	2399.0
13	Illinois	3891.0
20	Maryland	2230.0
21	Massachusetts	13257.0
22	Michigan	3142.0
23	Minnesota	3250.0
25	Missouri	2107.0
30	New Jersey	3350.0
32	New York	52070.0
33	North Carolina	2817.0
35	Ohio	3320.0
37	Oregon	3337.0
38	Pennsylvania	5349.0
43	Texas	6111.0
46	Virginia	2047.0
47	Washington	5880.0
49	Wisconsin	2167.0

Imagen 9.- Obtención de datos con condición

De *state_fam* Obtenga todos los renglones donde *state* sea New York o Washington

```
f_state_fam = state_fam[state_fam['state'].isin(['New York', 'Washington'])]
f_state_fam
```

✓ 0.0s

	state	family_members
32	New York	52070.0
47	Washington	5880.0

Imagen 20.- Obtención de datos con segunda condición

Generar nuevas columnas

Agrega una nueva columna llamada ***total***, esta columna tendrá la suma de *individuals* y *family_members*.

```
df['total'] = df['individuals'] + df['family_members']
df.head()
```

✓ 0.0s

	Unnamed: 0	region	state	individuals	family_members	state_pop	total
0	0	East South Central	Alabama	2570.0	864.0	4887681	3434.0
1	1	Pacific	Alaska	1434.0	582.0	735139	2016.0
2	2	Mountain	Arizona	7259.0	2606.0	7158024	9865.0
3	3	West South Central	Arkansas	2280.0	432.0	3009733	2712.0
4	4	Pacific	California	109008.0	20964.0	39461588	129972.0

Imagen 31.- Creación de columna

Funciones de agregación

En esta sección se trabajará con el dataset de ventas de Walmart.

Este dataset contiene ventas semanales de varias tiendas de los Estados Unidos.

Funciones estadisticas

Importe el dataset ***sales_subset*** e imprima los primeros datos usando la función *head()*.

```
sales_subset = pd.read_csv("C:/Users/HUAWEI/Desktop/L IA/Manual/Practica3_CONCEPTOS BASICOS DE PANDA/datasets/sales_subse
sales_subset.head()
```

✓ 0.0s

	Unnamed: 0	store	type	department	date	weekly_sales	is_holiday	temperature_c	fuel_price_usd_per_l	unemployment
0	0	1	A	1	2010-02-05	24924.50	False	5.727778	0.679451	8.106
1	1	1	A	1	2010-03-05	21827.90	False	8.055556	0.693452	8.106
2	2	1	A	1	2010-04-02	57258.43	False	16.816667	0.718284	7.808
3	3	1	A	1	2010-05-07	17413.94	False	22.527778	0.748928	7.808
4	4	1	A	1	2010-06-04	17558.09	False	27.050000	0.714586	7.808

Imagen 42.- Importar y visualizar los datos del segundo archivo

Obtenga el promedio y la mediana de la columna *weekly_sales*

```
prom_weekly_sales = sales_subset['weekly_sales'].mean()
mediana_weekly_sales = sales_subset['weekly_sales'].median()
print("Promedio de ventas semanales: ", prom_weekly_sales)
print("Mediana de ventas semanales: ", mediana_weekly_sales)
```

✓ 0.0s

```
Promedio de ventas semanales:  23843.95014850566
Mediana de ventas semanales:  12049.064999999999
```

Imagen 53.- Obtención de mediana y media

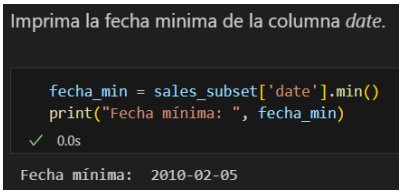

Imagen 64.- Obtención de fecha mínima

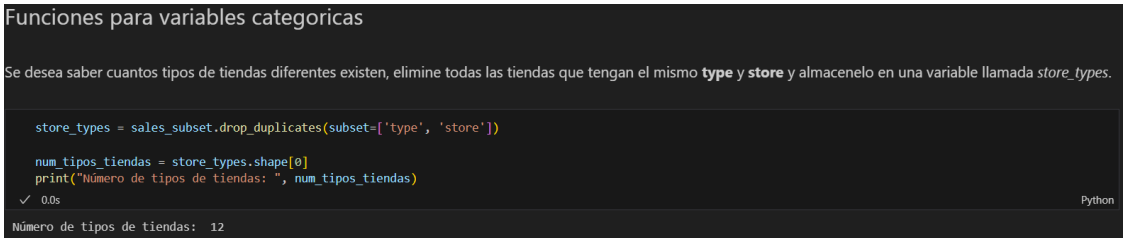

Imagen 75.- Obtención de número de tiendas

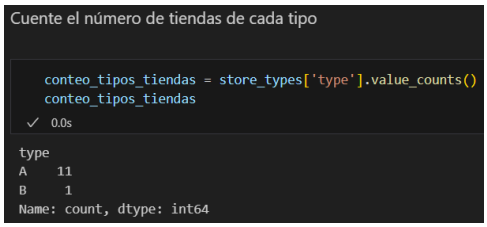

Imagen 86.- Obtención del número de cada tipo

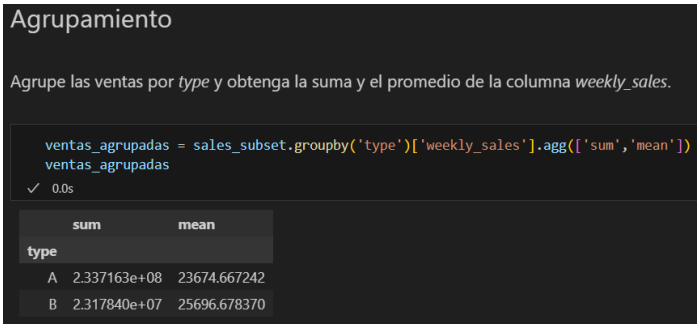

Imagen 97.- Suma y promedio de la agrupación

Imagen 108.- Suma y promedio por tabla pivote

Ahora obtenga el promedio de *weekly_sales* agrupando por *type* e *is_holiday*.

```
promedio_ventas = sales_subset.groupby(['type', 'is_holiday'])['weekly_sales'].mean()
promedio_ventas
```

✓ 0.0s

```
type  is_holiday
A     False         23768.583523
      True            590.045250
B     False         25751.980533
      True            810.705000
Name: weekly_sales, dtype: float64
```

```
promedio_ventas_pivote = pd.pivot_table(sales_subset, values='weekly_sales', index=['type', 'is_holiday'], aggfunc={'weekly_sales': ['sum', 'mean']})
promedio_ventas_pivote
```

✓ 0.0s

		mean	sum
type	is_holiday		
A	False	23768.583523	2.336927e+08
	True	590.045250	2.360181e+04
B	False	25751.980533	2.317678e+07
	True	810.705000	1.621410e+03

Imagen 119.- Promedio de la agrupación de dos tipos

Indexado

Subconjuntos con LOC y ILOC

En esta parte se trabajará con el dataset *temperatures*, este dataset la temperatura promedio mensual en distintas ciudades del mundo.

Importe el dataset ***temperatures***

```
temperatures_df = pd.read_csv("C:/Users/HUAWEI/Desktop/L IA/Manual/Practica3_CONCEPTOS BASICOS DE PANDA/datasets/temperatures.
temperatures_df.head()
```

✓ 0.0s

	Unnamed: 0	date	city	country	avg_temp_c
0	0	2000-01-01	Abidjan	Côte D'Ivoire	27.293
1	1	2000-02-01	Abidjan	Côte D'Ivoire	27.685
2	2	2000-03-01	Abidjan	Côte D'Ivoire	29.061
3	3	2000-04-01	Abidjan	Côte D'Ivoire	28.162
4	4	2000-05-01	Abidjan	Côte D'Ivoire	27.547

Imagen 20.- Importar y visualizar los datos del tercer archivo

Utilizando la función set_index(), asigne como índice la columna *date*, guarde el resultado en una variable llamada *temp_date*.

```
temp_date = temperatures_df.set_index('date')
temp_date.head()
```

✓ 0.0s

	Unnamed: 0	city	country	avg_temp_c
date				
2000-01-01	0	Abidjan	Côte D'Ivoire	27.293
2000-02-01	1	Abidjan	Côte D'Ivoire	27.685
2000-03-01	2	Abidjan	Côte D'Ivoire	29.061
2000-04-01	3	Abidjan	Côte D'Ivoire	28.162
2000-05-01	4	Abidjan	Côte D'Ivoire	27.547

Imagen 21.- Asignar columna

Utilizando el método loc, obtenga todos los renglones del 2010 (es importante ordenar el indice de fecha antes de usar el método loc).

```
temp_date = temp_date.sort_index()

temperatures_2010 = temp_date.loc['2010-01-01':'2010-12-31']
temperatures_2010
```

✓ 0.0s

date	Unnamed: 0	city	country	avg_temp_c
2010-01-01	4905	Faisalabad	Pakistan	11.810
2010-01-01	10185	Melbourne	Australia	20.016
2010-01-01	3750	Chongqing	China	7.921
2010-01-01	13155	São Paulo	Brazil	23.738
2010-01-01	5400	Guangzhou	China	14.136
...	...	...	...	...
2010-12-01	6896	Jakarta	Indonesia	26.602
2010-12-01	5246	Gizeh	Egypt	16.530
2010-12-01	11186	Nagpur	India	19.120
2010-12-01	14981	Sydney	Australia	19.559
2010-12-01	13496	Salvador	Brazil	26.265

1200 rows × 4 columns

Imagen 22.- Obtener valores de un rango de fechas

Ejercicio integrado

Utilizando el dataset *temperatures*, obtenga una nueva columna llamada *year* a partir de la columna *date*.

Almacene el resultado en una variable llamada *year_temp*

```
temperatures_df['year'] = pd.to_datetime(temperatures_df['date']).dt.year
year_temp = temperatures_df
```

✓ 0.0s

Imagen 23.- Obtener nueva columna a partir de otra columna

Asigne el año, el pais y la ciudad como tablas pivote en la variable year_temp para obtener la temperatura en grados Farenheit, el promedio, el valor minimo y maximo de temperaturas.

```
tabla_pivote = pd.pivot_table(temperatures_df,
                              values=['avg_temp_c'],
                              index=['year', 'country', 'city'],
                              aggfunc={'avg_temp_c': ['mean', 'min', 'max']})
tabla_pivote
```

✓ 0.0s Python

			avg_temp_c		
			max	mean	min
year	country	city			
2000	Afghanistan	Kabul	26.107	15.822667	3.326
	Angola	Luanda	26.496	24.410333	21.126
	Australia	Melbourne	22.095	14.320083	8.538
		Sydney	20.713	17.567417	14.454
	Bangladesh	Dhaka	29.253	25.905250	18.829
...	...	...	...	...	...
2013	United States	Chicago	22.230	11.586889	-0.509
		Los Angeles	25.090	18.120667	8.813
		New York	24.722	12.163889	-1.365
	Vietnam	Ho Chi Minh City	29.688	28.455000	26.597
	Zimbabwe	Harare	22.737	19.756500	16.299

1400 rows × 3 columns

Imagen 24.- Asignación de valores con tabla pivote

Comprensión

1. **¿Qué es una Serie de Pandas?**

Es una estructura de datos unidimensional, es parecido al arreglo de NumPy pero con la ventaja de tener etiquetas o índices personalizados. Además soporta diferentes tipos de datos (int, float, string, bool, etc.), funciona como un diccionario al acceder a los valores con etiquetas.

2. **¿Qué es un DataFrame de Pandas y como se diferencia de una Serie?**

Es una estructura de datos bidimensional que funciona similar a una hoja de calculo de Excel o a una base de datos. Es para manipular datos en filas y columnas.

Se diferencia al ser bidimensional, estar como tabla con filas y columnas y no como un vector con índices; tiene varias columnas y no solo una única columna de datos.

3. **Explique la diferencia entre los métodos loc y iloc de Pandas**

El acceso de loc está basado en etiquetas o nombres de índice, incluye el limite en el slicing, acepta listas de etiquetas o booleanos y se usa comúnmente en trabajos con índices personalizados.

Acceso de iloc está basado en índices numéricos de posición de fila y columna, no incluye el limite en el slicing, como el loc, acepta listas de etiquetas do booleanos y se usa comúnmente para acceder a datos por su posición.

4. **¿Qué es un índice en un DataFrame de Pandas y cuál es su propósito?**

Es una columna especial que identifica de manera única cada fila, funciona con una etiqueta para cada fila, facilitando el acceso y manipulación de los datos, puede ser numérico o personalizado.

Su propósito es acceder mas rápido a datos (loc[]), facilitación a la selección y filtrado de datos, permite ordenar datos fácilmente (df.soft_index()), útil en análisis de series temporales como fechas, además evita la duplicación de datos y mejora la organización.

5. **¿Cuál es la importancia de Pandas en el ámbito de análisis de datos y ciencia de datos?**

Permite cargar datos desde múltiples fuentes, tiene funciones avanzadas para manejar datos faltantes, también permite modificar estructuras de datos fácilmente, optimiza el rendimiento en el manejo de datos grandes, funciones grandes para análisis y estadística, integración con bibliotecas de Machine Learning, manejo eficiente de series temporales, compatibilidad con Matplotlib y Seaborn.

Conclusiones

Con Pandas tenemos gran manejo de datos con ayuda de la Serie y el DataFrame dependiendo de la necesidad que se requiera.

Ademas es muy útil porque facilita la manipulación, análisis y exploración de datos de manera eficiente.

Es útil al ser compatible con herramientas de Machine Learning, visualización de datos y por usarse en aplicaciones reales de análisis financiero, IA, big data, etc.