Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

alkemy

Clase grabada

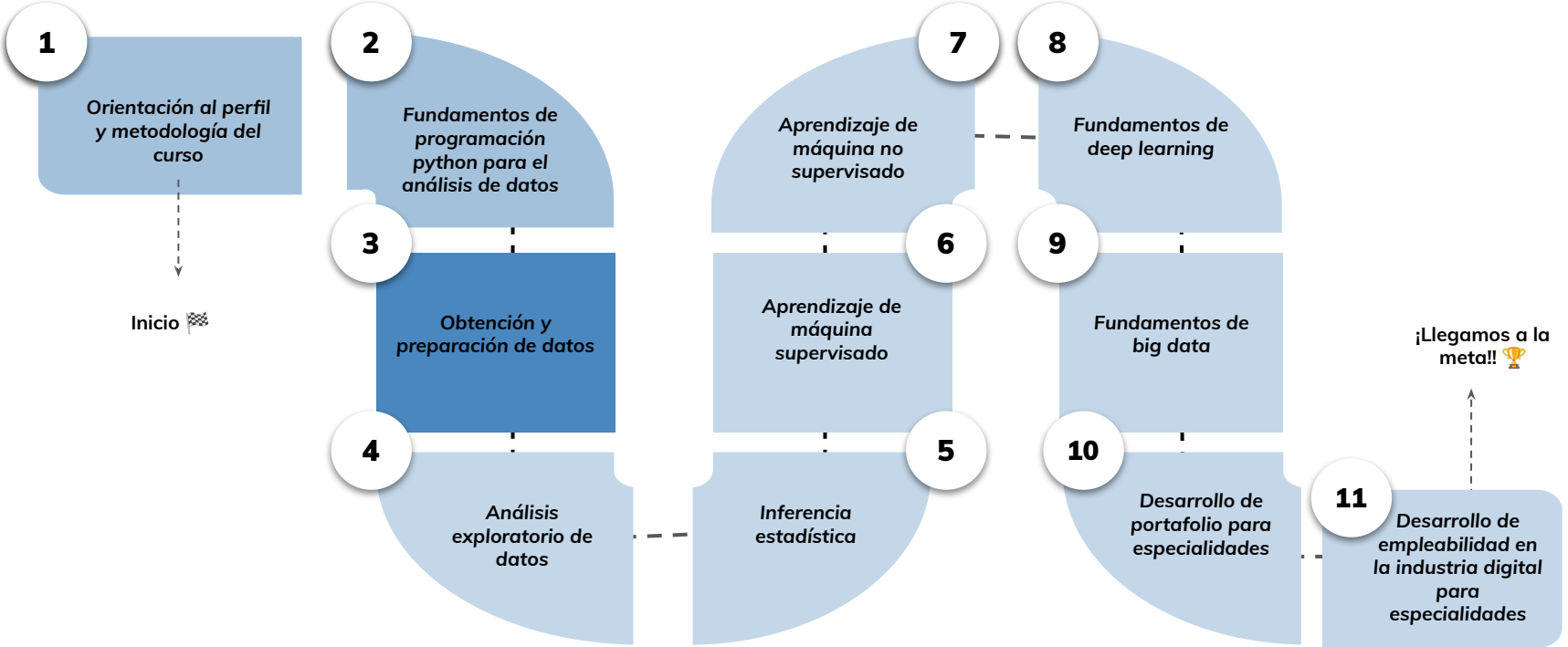
La librería pandas - Parte II

Aprendizaje Esperado 2: Utilizar métodos básicos de exploración en un set de datos utilizando estructuras de tipo serie y dataframe de la librería pandas para la selección, filtrado y sumarización de los datos para la resolución de un problema.

alkemy

Hoja de ruta

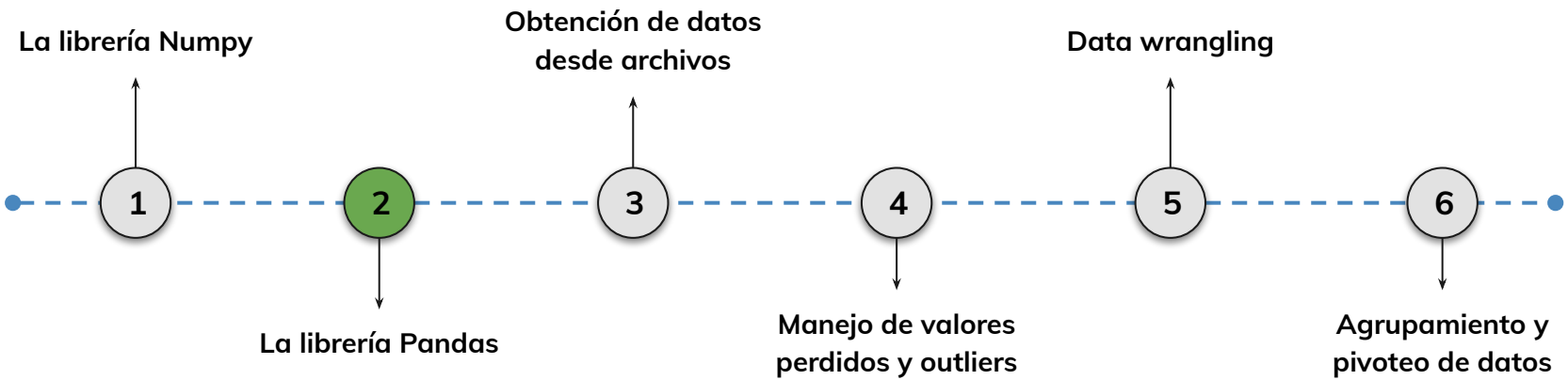
¿Cuáles skills conforman el programa? **Fundamentos de Ciencia de Datos**



alkemy

Roadmap de lecciones

*¿Cuáles **lecciones** estaremos estudiando en este módulo?*



alkemy

Learning Path

¿Cuáles temas trabajaremos hoy?

alkemy

Objetivos de aprendizaje

¿Qué aprenderás?

- Seleccionar datos específicos de un DataFrame usando índices y condiciones.
- Aplicar métodos de exploración para comprender la estructura de los datos.
- Resumir datos mediante funciones estadísticas y análisis de frecuencias.
- Integrar operaciones de agrupación y combinación de DataFrames.
- Preparar datos para su análisis y visualización.

Repaso clase anterior

¿Quedó alguna duda?

En la clase anterior trabajamos 📚:

- → *Repasamos la estructura y creación de Series y DataFrames en Pandas.*
- → *Aprendimos a seleccionar filas y columnas mediante índices y nombres.*
- → *Cargamos datos desde archivos externos y realizamos operaciones básicas.*
- → *Exploramos métodos de acceso .loc[] y .iloc[] para trabajar con subconjuntos.*

La librería pandas

Selección de filas o columnas en un DATAFRAME

Para seleccionar una columna específica, se puede utilizar su nombre como clave:

```
# Seleccionar la columna 'Nombre'nombres = df['Nombre']
```

Para seleccionar varias columnas:

```
# Seleccionar las columnas 'Nombre' y 'Edad'datos_personales = df[['Nombre', 'Edad']]
```

Selección de Filas en un DataFrame

Las filas pueden seleccionarse usando loc[] para etiquetas o iloc[] para posiciones numéricas:

Este tipo de selección es útil cuando se trabaja con grandes volúmenes de datos y se requiere acceder a subconjuntos específicos para su análisis o visualización.

```
print(df.loc[1])  # Devuelve la fila con índice 1
print(df.iloc[2])  # Devuelve la tercera fila (posición 2)
```

Selección de elementos en un DATAFRAME

Para acceder a un elemento específico, se combinan loc[] o iloc[] con nombres de columnas:

```
# Acceder al valor en la primera fila, columna 'Edad'edad_primera_persona = df.loc[0, 'Edad']# Acceder al valor en la segunda fila, tercera columnavalor = df.iloc[1, 2]
```

Selección de Subconjuntos de Datos

También se pueden seleccionar subconjuntos de datos, lo que facilita el análisis y la manipulación de información relevante:

```
# Seleccionar un bloque de datos (filas 0-1, columnas 'Nombre' y 'Ciudad')subconjunto =
df.loc[0:1, ['Nombre', 'Ciudad']]# Seleccionar las primeras 2 filas y las primeras 2
columnasbloque = df.iloc[0:2, 0:2]
```

Selección condicional en un DATAFRAME

Los DataFrames permiten filtrar datos con condiciones lógicas, lo que resulta muy útil para análisis exploratorio y segmentación de datos.

```
# Filtrar personas mayores de 25 añosmayores_25 = df[df['Edad'] > 25]# Filtrar personas de Madridde_madrid = df[df['Ciudad'] == 'Madrid']
```

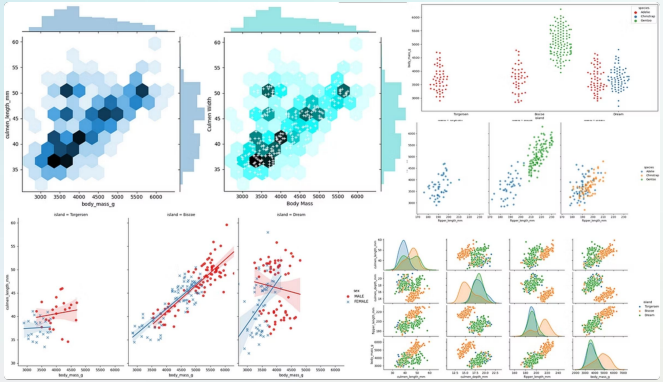
Combinación de Condiciones

También se pueden combinar condiciones con operadores & y |, lo que permite realizar consultas más complejas:

```
# Personas mayores de 25 años que viven en Barcelonafiltro = df[(df['Edad'] > 25) &
(df['Ciudad'] == 'Barcelona')]# Personas de Madrid o Barcelonaciudades = df[(df['Ciudad'] ==
'Madrid') | (df['Ciudad'] == 'Barcelona')]
```

La selección condicional es una de las funcionalidades más utilizadas en Pandas, ya que permite extraer información relevante de grandes conjuntos de datos.

MÉTODOS BÁSICOS DE EXPLORACIÓN



Métodos básicos de exploración

(head, tail, info, describe)

Los DataFrames incluyen métodos para obtener información rápida sobre los datos:

head()

Muestra las primeras filas del DataFrame (por defecto 5).

```
df.head()
```

tail()

Muestra las últimas filas del DataFrame (por defecto 5).

```
df.tail(3)
```

info()

Proporciona un resumen conciso del DataFrame, incluyendo tipos de datos y valores no nulos.

```
df.info()
```

describe()

Genera estadísticas descriptivas para columnas numéricas.

```
df.describe()
```

Importancia de los Métodos de Exploración

Estos métodos permiten entender la estructura y contenido de los datos antes de realizar análisis más detallados. info() proporciona información sobre los tipos de datos y valores nulos, mientras que describe() devuelve estadísticas generales sobre las columnas numéricas.

Métodos básicos de sumarización

MIN, MAX, COUNT, MEAN, MEDIAN, SUM)

Para obtener resúmenes estadísticos de las columnas numéricas, Pandas ofrece métodos que facilitan la exploración y comprensión de los datos:

min() y max()

Devuelven los valores mínimos y máximos de cada columna numérica.

```
df['Edad'].min()  # Edad
mínimadf['Edad'].max()  # Edad máxima
```

mean() y median()

Calculan la media y la mediana de las columnas numéricas.

```
df['Edad'].mean()    # Media de
edadesdf['Edad'].median()  # Mediana de
edades
```

count()

Cuenta el número de valores no nulos en cada columna.

```
df.count()  # Número de valores no nulos
```

sum()

Suma los valores de cada columna numérica.

```
df['Edad'].sum() # Suma total de edades
```

Importancia de los Métodos de Sumarización

Estos métodos son clave en el análisis de datos, ya que permiten identificar tendencias y patrones de manera rápida y eficiente.

Métodos unique, nunique, value_counts

Para analizar valores únicos y sus frecuencias, Pandas proporciona métodos que ayudan a explorar la distribución de los datos:

unique()

Devuelve un array con los valores únicos de una columna.

```
df['Ciudad'].unique()  # Ciudades únicas
```

nunique()

Cuenta el número de valores únicos en una columna.

```
df['Ciudad'].nunique()  # Número de ciudades diferentes
```

value_counts()

Cuenta la frecuencia de cada valor único en una columna.

```
df['Ciudad'].value_counts() # Frecuencia de cada ciudad
```

Aplicaciones de los Métodos de Análisis de Valores Únicos

Estos métodos son útiles para comprender la diversidad y composición de los datos, lo que facilita la toma de decisiones basada en información estructurada.

Ejemplo de Análisis con value_counts()

El método value_counts() es particularmente útil para analizar la distribución de categorías en un conjunto de datos:

```
# Contar la frecuencia de cada ciudadciudades = df['Ciudad'].value_counts()# Normalizar para obtener porcentajesporcentajes = df['Ciudad'].value_counts(normalize=True)# Ordenar por frecuencia descendenteordenado = df['Ciudad'].value_counts().sort_values(ascending=False)
```

Este tipo de análisis permite identificar rápidamente las categorías más comunes y su proporción en el conjunto de datos.

Métodos Adicionales para Exploración de Datos

dtypes

Muestra los tipos de datos de cada columna.

```
df.dtypes
```

shape

Devuelve una tupla con las dimensiones del DataFrame (filas, columnas).

```
df.shape
```

columns

Lista los nombres de las columnas del DataFrame.

```
df.columns
```

isnull().sum()

Cuenta los valores nulos en cada columna.

```
df.isnull().sum()
```

Métodos de Agrupación

Pandas permite agrupar datos según valores de una o más columnas para realizar análisis agregados:

```
# Agrupar por ciudad y calcular la media de edaddf.groupby('Ciudad')['Edad'].mean()# Agrupar por ciudad y contar registrosdf.groupby('Ciudad').size()# Agrupar por ciudad y obtener múltiples estadísticasdf.groupby('Ciudad').agg({'Edad': ['min', 'max', 'mean']})
```

La agrupación es fundamental para el análisis de datos, ya que permite identificar patrones y tendencias en diferentes segmentos del conjunto de datos.

Operaciones de Unión y Combinación

Pandas ofrece métodos para combinar diferentes DataFrames:

merge()

Combina DataFrames basándose en columnas comunes, similar a un JOIN en SQL.

```
pd.merge(df1, df2, on='id')
```

concat()

Concatena DataFrames a lo largo de un eje (filas o columnas).

```
pd.concat([df1, df2], axis=0)
```

join()

Une DataFrames utilizando sus índices.

```
df1.join(df2)
```

Manejo de Datos Faltantes

Pandas proporciona métodos para detectar y manejar valores nulos:

```
# Detectar valores nulosdf.isnull()# Eliminar filas con valores nulosdf_limpio = df.dropna()# Rellenar valores nulos con un valor específicodf_relleno = df.fillna(0)# Rellenar valores nulos con la media de la columnadf['Edad'] = df['Edad'].fillna(df['Edad'].mean())
```

El manejo adecuado de datos faltantes es crucial para evitar sesgos y errores en el análisis.

Transformación de Datos

Pandas facilita la transformación de datos para adaptarlos a diferentes necesidades:

apply()

Aplica una función a cada elemento, fila o columna.

```
df['Edad'].apply(lambda x: x * 2)
```

map()

Mapea valores de una Serie según un diccionario o función.

```
df['Género'].map({'M': 'Masculino', 'F': 'Femenino'})
```

replace()

Reemplaza valores específicos en el DataFrame.

```
df.replace('Madrid', 'MAD')
```

astype()

Convierte el tipo de datos de una columna.

```
df['Edad'].astype(float)
```

Ordenación de Datos

Pandas permite ordenar DataFrames según los valores de una o más columnas:

```
# Ordenar por edad en orden ascendentedf_ordenado = df.sort_values('Edad')# Ordenar por
múltiples columnasdf_ordenado = df.sort_values(['Ciudad', 'Edad'], ascending=[True, False])#
Ordenar por índicedf_ordenado = df.sort_index()
```

La ordenación facilita la visualización y análisis de datos, permitiendo identificar rápidamente valores extremos y tendencias.

Exportación de Datos

Pandas permite guardar DataFrames en diferentes formatos:

to_csv()

Guarda el DataFrame como archivo CSV.

```
df.to_csv('datos.csv', index=False)
```

to_excel()

Exporta el DataFrame a un archivo Excel.

```
df.to_excel('datos.xlsx',
sheet_name='Hoja1')
```

to_json()

Convierte el DataFrame a formato JSON.

```
df.to_json('datos.json')
```

to_sql()

Guarda el DataFrame en una tabla de base de datos SQL.

```
df.to_sql('tabla', conexion_sql)
```

Visualización de Datos con Pandas

Pandas se integra con bibliotecas de visualización para crear gráficos directamente desde DataFrames:

```
# Histograma de edadesdf['Edad'].plot.hist()# Gráfico de barras para
ciudadesdf['Ciudad'].value_counts().plot.bar()# Diagrama de
dispersióndf.plot.scatter(x='Edad', y='Ingresos')# Gráfico de líneas para series
temporalesdf_tiempo.plot.line()
```

La visualización es esencial para comprender patrones y tendencias en los datos de manera intuitiva.

Operaciones Avanzadas con DataFrames

pivot_table()

Crea tablas dinámicas para análisis multidimensional.

```
pd.pivot_table(df, values='Ventas',
index='Región', columns='Producto')
```

melt()

Transforma datos de formato ancho a largo.

```
pd.melt(df, id_vars=['ID'],
value_vars=['2020', '2021'])
```

crosstab()

Crea tablas de contingencia para análisis de frecuencias.

```
pd.crosstab(df['Género'], df['Ciudad'])
```

cut()

Divide datos numéricos en categorías o intervalos.

```
pd.cut(df['Edad'], bins=[0, 18, 35, 65,
100])
```

Manejo de Series Temporales

Pandas ofrece funcionalidades específicas para trabajar con datos temporales:

```
# Crear un rango de fechasfechas = pd.date_range(start='2023-01-01', periods=10, freq='D')# Convertir columna a tipo datetimedf['Fecha'] = pd.to_datetime(df['Fecha'])# Extraer componentes de fechadf['Año'] = df['Fecha'].dt.yeardf['Mes'] = df['Fecha'].dt.month# Remuestreo de datos temporalesdf_mensual = df.resample('M', on='Fecha').mean()
```

El análisis de series temporales es fundamental en áreas como finanzas, meteorología y análisis de tendencias.

Optimización de Rendimiento

Para trabajar con grandes volúmenes de datos, Pandas ofrece opciones de optimización:

Tipos de Datos Eficientes

Usar tipos de datos apropiados reduce el consumo de memoria.

```
df['ID'] = df['ID'].astype('int32')
```

Procesamiento por Lotes

Leer archivos grandes en fragmentos evita problemas de memoria.

```
for chunk in pd.read_csv('datos.csv',
chunksize=10000):
```

Operaciones Vectorizadas

Evitar bucles explícitos y usar operaciones vectorizadas mejora el rendimiento.

Categorías

Convertir columnas de texto repetitivo a tipo categoría reduce el uso de memoria.

```
df['Ciudad'] =
df['Ciudad'].astype('category')
```

Integración con Otras Bibliotecas

Pandas se integra perfectamente con el ecosistema de ciencia de datos de Python:

NumPy

Conversión fluida entre arrays de NumPy y estructuras de Pandas.

Matplotlib/Seaborn

Creación de visualizaciones avanzadas a partir de DataFrames.

Scikit-learn

Preparación de datos para modelos de machine learning.

SQL/Bases de Datos

Lectura y escritura directa desde y hacia bases de datos.

Conclusión

Pandas es una de las herramientas más poderosas dentro del ecosistema de Python para la manipulación y análisis de datos. A lo largo de este manual, se han abordado sus características esenciales, incluyendo la creación de estructuras de datos como Series y DataFrames, la selección y filtrado de información, y la exploración y agregación de datos mediante distintos métodos. Su flexibilidad y facilidad de uso han hecho que se convierta en una herramienta indispensable para analistas, científicos de datos y desarrolladores.

El conocimiento adquirido sobre Pandas permite gestionar grandes volúmenes de datos de manera eficiente, facilitando la transformación, limpieza y análisis de información con un código conciso y optimizado. Su integración con bibliotecas como NumPy, Matplotlib, Seaborn y Scikit-learn amplía aún más sus aplicaciones, permitiendo el desarrollo de flujos de trabajo completos para la exploración y modelado de datos.

Live Coding

¿En qué consistirá la Demo?

Mostraremos cómo filtrar y explorar un DataFrame real, aplicando condiciones, estadísticas y agrupaciones para obtener información útil.

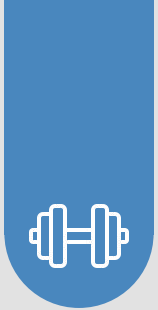
1. *Seleccionar columnas y filas específicas con .loc[] y .iloc[].*
2. *Filtrar datos usando operadores lógicos (&, |).*
3. *Aplicar métodos head(), tail(), info() y describe().*
4. *Calcular estadísticas básicas con mean(), median(), sum().*
5. *Analizar valores únicos con value_counts().*

Tiempo: 25 Minutos

Momento:

Time-out!

⏰5 -10 min.



Ejercicio N° 1

Exploración y resumen de datos

Exploración y resumen de datos

Contexto: 🙌

Utilizarás Pandas para explorar un conjunto de datos, aplicar filtros y generar estadísticas descriptivas que permitan comprender mejor la información.

Consigna: ✍️

1. Carga un dataset en un DataFrame.
2. Selecciona un subconjunto de columnas y filas.
3. Aplica filtros condicionales simples y combinados.
4. Obtén un resumen de las columnas con describe().
5. Calcula frecuencias con value_counts() y agrupa datos con groupby().

Paso a paso: ⚙️

1. Importar Pandas y cargar los datos.
2. Explorar con métodos head(), tail(), info().
3. Filtrar datos con condiciones.
4. Generar estadísticas y agrupaciones.
5. Guardar los resultados en un nuevo archivo CSV.

Tiempo 🕐**:** *45 Minutos*

¿Alguna consulta?

alkemy

Resumen

¿Qué logramos en esta clase?

- ✓ ***Seleccionamos datos de forma precisa con .loc[] y .iloc[].***
- ✓ ***Filtramos registros utilizando condiciones lógicas simples y múltiples.***
- ✓ ***Exploramos datasets con métodos de inspección rápida (head, tail, info).***
- ✓ ***Resumimos datos con estadísticas descriptivas y análisis de frecuencias.***
- ✓ ***Agrupamos información y combinamos DataFrames para análisis más complejos.***

¡Ponte a prueba!

Momento de ejercitación

Te invitamos a aprovechar esta última sección del espacio sincrónico para realizar de manera individual las ***actividades disponibles en la plataforma.*** *Estas propuestas son clave para afianzar lo trabajado y* ***forman parte obligatoria del recorrido de aprendizaje.***

💪 Análisis de caso ——— 💪 Selección Múltiple

💪 Comprensión lectora

Si al resolverlas surge alguna duda, compartela o tráela al próximo encuentro sincrónico.

alkemy

¡Muchas gracias!

alkemy