Instrucciones Pandas

Instrucciones Básicas

Importar y configuración inicial

```
import pandas as pd  # Importar pandas
pd.set_option('display.max_rows', 10)  # Limitar filas mostradas
pd.set_option('display.max_columns', None)  # Mostrar todas las columnas
```

Crear un DataFrame o Series

```
# Desde un diccionario
df = pd.DataFrame({
    'Nombre': ['Ana', 'Luis', 'Pedro'],
    'Edad': [23, 35, 30],
    'Ciudad': ['Madrid', 'Sevilla', 'Barcelona']
})

# Desde un archivo CSV
df = pd.read_csv('archivo.csv')

# Series (una sola columna)
s = pd.Series([10, 20, 30], name="Valores")
```

Explorar datos rápidamente

```
df.head(5)       # Primeras 5 filas
df.tail(5)       # Últimas 5 filas
df.info()        # Información de columnas y tipos
df.describe()    # Estadísticas numéricas básicas
df.shape         # (filas, columnas)
df.columns       # Lista de nombres de columnas
df.dtypes        # Tipos de datos por columna
```

Selección de datos

```
# Seleccionar columnas
df['Nombre']                # Serie
df[['Nombre', 'Edad']]      # DataFrame

# Seleccionar filas por índice
df.iloc[0]                  # Primera fila (por posición)
df.iloc[0:2]                # Filas de la 0 a la 1

# Seleccionar filas por condición
df[df['Edad'] > 25]
df[(df['Edad'] > 25) & (df['Ciudad'] == 'Madrid')]

# Selección por etiqueta
df.loc[0, 'Nombre']         # Celda
df.loc[:, ['Nombre', 'Ciudad']]
```

Filtrar y ordenar

```
df.sort_values(by='Edad')                # Ordenar ascendente
df.sort_values(by='Edad', ascending=False)  # Descendente
df['Ciudad'].unique()                    # Valores únicos
df['Ciudad'].nunique()                   # Número de valores únicos
df['Ciudad'].value_counts()              # Conteo de valores
```

Modificar datos

```
# Agregar columna
df['Edad+10'] = df['Edad'] + 10

# Eliminar columnas o filas
df.drop(columns=['Edad+10'], inplace=True)
df.drop(index=0, inplace=True)

# Renombrar columnas
```

```
df.rename(columns={'Nombre': 'Name'}, inplace=True)

# Reemplazar valores
df['Ciudad'] = df['Ciudad'].replace({'Madrid': 'MAD'})
```

Agrupación y operaciones

```
# Agrupar y calcular
df.groupby('Ciudad')['Edad'].mean()
df.groupby('Ciudad').agg({'Edad': ['mean', 'max']})

# Conteo rápido
df.groupby('Ciudad').size()
```

Trabajar con valores nulos

```
df.isnull().sum()          # Conteo de nulos por columna
df.dropna()                # Eliminar filas con nulos
df.fillna(0)               # Rellenar nulos con 0
df['Edad'].fillna(df['Edad'].mean(), inplace=True)  # Rellenar con media
```

Exportar datos

```
df.to_csv('salida.csv', index=False)    # CSV
df.to_excel('salida.xlsx', index=False) # Excel
```

Trucos

```
df.sample(5)                        # Muestra aleatoria de 5 filas
df.query('Edad > 25 and Ciudad == "MAD"')  # Filtrado tipo SQL
df['Edad'].between(20, 30)          # Rango de valores
df.memory_usage(deep=True)          # Uso de memoria
```

Instrucciones mas avanzadas

Optimizar carga de datos

```
# Cargar solo columnas necesarias
cols = ['Nombre', 'Edad', 'Ciudad']
df = pd.read_csv('archivo.csv', usecols=cols)

# Cargar en chunks (por partes) para datasets enormes
for chunk in pd.read_csv('archivo.csv', chunksize=50000):
    procesar(chunk)  # función personalizada

# Especificar tipos de datos para reducir memoria
df = pd.read_csv('archivo.csv', dtype={'Edad': 'int8', 'Ciudad': 'category'})
```

Convertir tipos para mejorar rendimiento

```
df['Ciudad'] = df['Ciudad'].astype('category')   # Categórico para strings re
petidos
df['Edad'] = df['Edad'].astype('int16')          # Numérico optimizado
df['Fecha'] = pd.to_datetime(df['Fecha'])        # Fechas eficientes
```

Filtrado y búsquedas rápidas

```
# Usar query para filtrado tipo SQL (más legible y rápido)
df_filtrado = df.query('Edad > 30 and Ciudad == "Madrid"')

# Filtrar usando isin (rápido para listas)
df[df['Ciudad'].isin(['Madrid', 'Sevilla'])]
```

Indexación inteligente

```
# Crear índice para búsquedas rápidas
df.set_index('ID', inplace=True)
```

```
# Buscar por índice (instantáneo)
fila = df.loc[12345]
```

Operaciones vectorizadas (evitar bucles)

```
# ❌ Lento (bucle)
df['Edad+10'] = [x + 10 for x in df['Edad']]

# ✅ Rápido (vectorizado)
df['Edad+10'] = df['Edad'] + 10
```

Agrupaciones avanzadas

```
# Varias funciones a la vez
df.groupby('Ciudad').agg(
    Edad_media=('Edad', 'mean'),
    Edad_max=('Edad', 'max'),
    Conteo=('Edad', 'size')
)

# Agrupación con orden
df.groupby('Ciudad', sort=False)['Edad'].mean()
```

Merge y Join eficientes

```
# Unir dos DataFrames
df_merged = df1.merge(df2, on='ID', how='inner')

# Join rápido por índice
df_joined = df1.join(df2, how='left')
```

Pivot y tablas dinámicas

```
# Pivot simple
df.pivot(index='Ciudad', columns='Sexo', values='Edad')
```

```
# Pivot_table (con agregación)
pd.pivot_table(df, values='Edad', index='Ciudad', columns='Sexo', aggfunc
='mean')
```

Trabajar con fechas y tiempos

```
df['Año'] = df['Fecha'].dt.year
df['Mes'] = df['Fecha'].dt.month
df['DiaSemana'] = df['Fecha'].dt.day_name()

# Agrupar por año y mes
df.groupby([df['Fecha'].dt.year, df['Fecha'].dt.month])['Ventas'].sum()
```

Aplicar funciones personalizadas de forma eficiente

```
# apply es más lento que vectorizar, pero útil para funciones personalizada
s
df['Categoria'] = df['Edad'].apply(lambda x: 'Adulto' if x >= 18 else 'Niño')

# apply por filas (axis=1)
df['Suma'] = df.apply(lambda row: row['Col1'] + row['Col2'], axis=1)
```