Clase 6

Contenido

Análisis Exploratorio de Datos con Pandas.

Pandas nos permite cargar datos, modelarlo, realizar la preparación, manipulación y análisis de los mismos, para ello nos presenta un concepto fundamental para visualizar las estructuras de datos, los DataFrames. Son estructuras de datos etiquetados bidimensionales, es decir, que contienen filas y columnas.

Para saber más acerca de esta librería recomiendo consultar en la documentación oficial: https://pandas.pydata.org/. En esta página podrá encontrar como instalar Pandas y una guía de usuario con ejemplos cortos sobre cómo usarla.

1.- Importar pandas

Como estándar se importa pandas como *pd*,

```
import pandas as pd
```

Clase 6

2.- Leer y guardar archivos CSV

Pandas nos permite leer archivos en formato CSV (archivos separados por comas) y convertirlos a una estructura de datos como DataFrame para trabajar con ellos.

```
df = pd.read_csv("<dirección donde se encuentra el archivo>")
```

Asimismo, pandas nos permite almacenar un DataFrame que hayamos creado o que esté modificado y listo para ingresar a un modelo de Machine Learning o implementar visualizaciones.

```
df.to_csv("<ruta donde almacenar el archivo con el nombre finalizando en .csv>")
```

Pandas también nos permite leer archivos HTML, JSON, SQL y Excel.

```
pd.read_html("<ruta>")
pd.read_json("<ruta>")
pd.read_sql("<ruta>")
pd.read_excel("<ruta>")
```

3.- Crear un DataFrame

Se pueden crear DataFrame de dos formas:

1. A partir de diccionarios de listas.

Para este caso vamos a crear un DataFrame con datos de un supermercado. La clave del diccionario es el nombre de la columna y el valor es una lista donde cada elemento contiene la información de la columna en el DataFrame. Cada valor de las columnas debe estar ingresado en el orden que corresponda para visualizar de manera correcta las filas del DataFrame.

Primero, definimos el diccionario:

```
dic_to_df = {'Categoria':['Verduras','Lacteos','Carnes frias'],
             'Producto':['Pepino','Leche Entera','Salchichas'],
             'Precio':[600,2300,4650],
             'Fecha':['2021-01-02','2019-01-31','2020-05-25']}
```

Luego, el diccionario se convierte a DataFrame:

```
df = pd.DataFrame(dic_to_df)
```

En le Figura 1 puede verse el dataframe creado.

	Categoria	Producto	Precio	Fecha
0	Verduras	Pepino	600	2021-01-02
1	Lacteos	Leche Entera	2300	2019-01-31
2	Carnes frias	Salchichas	4650	2020-05-25

Figura1. DataFrame a partir de diccionarios de listas.

2. A partir de listas de diccionarios.

En este caso cada elemento de la lista es un diccionario que contiene la información de una fila del DataFrame. Las claves serán los nombres de las columnas y los valores tendrán la información que corresponde al producto ingresado.

Primero, creamos la lista de diccionarios con la información de los productos:

```
list_to_df =[{'Categoria':'Verduras','Producto':'Pepino','Precio':600,'Fecha':'2021-01-02'},
{'Categoria':'Lacteos','Producto':'Leche Entera','Precio':2300, 'Fecha':'2019-01-31'},
{'Categoria':'Carnes Frias','Producto':'Salchichas','Precio':4650, 'Fecha':'2020-05-25'}]
```

Luego, creamos el DataFrame:

```
df = pd.DataFrame(list_to_df)
```

En la Figura 2 puede verse el dataframe creado.

	Categoria	Producto	Precio	Fecha
0	Verduras	Pepino	600	2021-01-02
1	Lacteos	Leche Entera	2300	2019-01-31
2	Carnes Frias	Salchichas	4650	2020-05-25

Figura 2. DataFrame a partir de listas de diccionarios.

Clase 6

4.- Visualizar un DataFrame

Para visualizar un DataFrame podemos simplemente poner el nombre que le hemos asignado al DataFrame y visualizarlo como en las imágenes anteriores. Esta es una buena opción cuando tenemos pocos datos, pero cuando la cantidad de datos aumenta se hace tediosa la visualización. Pandas nos presenta una opción para visualizar algunas filas del DataFrame y con ello hacernos una idea de los datos que vamos a preparar.

```
df.head()
```

Con este método de Pandas podemos visualizar las cinco primeras filas del DataFrame. También podemos indicar dentro del paréntesis cuantas filas del DataFrame queremos ver:

```
df.head(10)
```

Con la línea de código anterior podremos visualizar las 10 primeras filas del DataFrame.

```
df.tail()
```

El método anterior nos permite visualizar las últimas filas del DataFrame. Al igual que ***.head()*** le podemos indicar el número de filas que queremos visualizar.

5.- Explorar el DataFrame

Para explorar el DataFrame e identificar datos importantes del DataFrame con el que vamos a trabajar podemos usar los siguientes métodos,

Información de DataFrame por cada columna, con cantidad de valores nulos y tipos de datos:

```
df.info()
```

Algunos datos estadísticos del DataFrame para las columnas numéricas: Total de datos, media, valor mínimo y máximo, cuartiles.

```
df.describe()
```

Clase 6

6.- Clasificación y subconjuntos de datos

Podemos ordenar los valores de un DataFrame por orden alfabético, de manera descendente y ascendente.

```
df.sort_values('<nombreColumna>', ascending = False)
```

En este método, ascending viene por defecto con el valor True, es decir que ordena los datos de forma ascendente.

Se pueden ordenar por más de una columna, para ello se ingresan los nombres de las columnas como una lista, al igual que los valores que tomará *ascending*.

```
df.sort_values(['<nombreColumna1, nombreColumna2 >'], ascending = [False, True])
```

Para un subconjunto de datos de un DataFrame podemos indicar el nombre de la columna o columnas que queremos visualizar.

```
df['<nombreColumna>']
df[['<nombreColumna1>','<nombreColumna1>']]
```

También se pueden visualizar los datos que cumplen una condición definida.

```
df[df['<nombreColumna>']<Condición>]
```

En la condición se puede indicar que el valor de esa columna sea mayor, igual, menor o diferente a un valor definido. Incluso se pueden definir varias condiciones haciendo uso de operadores lógicos (*and, or*).

Finalmente, si queremos columnas que tengan un valor especifico podemos usar el método *.isin()*

```
condicion = df['<nombreColumna>'].isin([<Valor o valores>])
df[condicion]
```

Clase 6

7.- Resumen estadístico

```
df['<nombreColumna>'].mean() #Media
df['<nombreColumna>'].median() #Mediana
df['<nombreColumna>'].mode() #Moda
df['<nombreColumna>'].min() #Valor mínimo
df['<nombreColumna>'].max() #Valor máximo
df['<nombreColumna>'].var() #Varianza
df['<nombreColumna>'].std() #Desviación estándar
df['<nombreColumna>'].sum() #Suma
df['<nombreColumna>'].quantile() #Cuartiles
```

Estadísticas acumulativas:

```
df['<nombreColumna>'].cumsum() #Suma acumulativa
df['<nombreColumna>'].cummin() #Mínimo acumulativo
df['<nombreColumna>'].cummax() #Máximo acumulativo
df['<nombreColumna>'].cumprod() #Producto acumulativo
```

También podemos usar el metodo *.agg()* para uno o mas métodos o para aplicar una función previamente definida a los datos.

```
df['<nombreColumna>'].agg(<funcion>)
```

8.- Eliminar valores duplicados

Pandas nos permite identificar valores duplicados en el DataFrame y eliminarlos.

```
df.drop_duplicates(subset="<nombreColumna>",inplace=True)
```

El parámetro inplace nos permite indicar si queremos que los valores duplicados se eliminen directamente en el DataFrame.

9.- Contar valores del DataFrame

Es útil contar los valores que tenemos en el DataFrame, esta es otra forma de identificar valores duplicados para luego eliminarlos.

```
df['<nombreColumna>'].value_counts(sort=True)
```

10.- Valores NaN

Los datos difícilmente están perfectos y listos para realizar su visualización o crear modelos de ML, debemos realizar una limpieza previa. Algo que se suele hacer es identificar los valores NaN (Not a Number) y eliminarlos o usar técnicas para reemplazarlos.

Identificar si nuestro DataFrame tiene valores NaN:

```
df.isna()#Indica valores booleanos (True o False) por cada elemento
df.isna().any() #Indica valores booleanos por cada columna
df.isna().sum() #Indica valores numéricos por cada columna
```

Para eliminar los valores NaN:

```
df.dropna(inplace=True)
```

Para reemplazarlos por un valor especifico:

```
df.fillna(<valor>)
```

11.- Selección de datos

En esta ocasión veremos *.iloc[]* y *.loc[]*. Técnicas para seleccionar partes de datos.

.iloc[]: Está basado en posición. El numero indicado al final es excluyente, es decir, si usamos *.iloc[1:5]* nos muestra los valores desde la posición 1 hasta la 4.

.loc[]: Está basado en etiquetas. Ambos valores indicados están incluidos.

```
df.iloc[<numeroFila>,<numeroColumna>]
df.loc[<indiceInicio>:<indiceFin>]
```

numeroFila y *numeroColumna* pueden ser un único número o se usa la notación *inicio:fin* (al igual que en .loc).

Clase 6

12.- Group By

Este método nos permite agrupar datos por determinados valores en común para calcular operaciones.

```
df.groupby("<nombreColumna1>")["<nombreColumna2>"].agg(<[funcion]>)
```

nombreColumna1: Determina la columna por la cual se van a agrupar los datos

nombreColumna2: Este valor es opcional e indica los valores de otra columna del DataFrame al que le vamos a aplicar las funciones indicadas con *.agg()*.