COLOMBIA POTENCIA DE LA VIDA

TIC

UNIVERSIDAD SERGIO ARBOLEDA

Análisis de Datos
Nivel Integrador

TALENTO TECH

UNIVERSIDAD SERGIO ARBOLEDA

PREPARACIÓN SEMANA 5

TRABAJO CON FECHAS Y SERIES TEMPORALES

Datetime: El módulo **datetime** de Python proporciona clases para trabajar con fechas y horas de manera eficiente. La clase principal es **datetime**, que representa un objeto combinado de fecha y hora.

```
from datetime import datetime

# Obtener la fecha y hora actual
fecha_actual = datetime.now()

# Crear una fecha específica
fecha_personalizada = datetime(2022, 10, 1, 12, 30, 0)
```

TRABAJO CON FECHAS Y SERIES TEMPORALES

Creación y Manipulación de Series Temporales en Pandas:

Pandas proporciona las clases **Timestamp** para representar puntos temporales y **DatetimeIndex** para índices temporales. Además, la manipulación de fechas y series temporales se realiza con facilidad en Pandas.

```python
import pandas as pd

# Crear una serie temporal con fechas
fechas = pd.date_range(start='2022-01-01', end='2022-01-10', freq='D')

# Crear una serie temporal con índice de fechas
datos = [10, 20, 15, 25, 30, 22, 18, 27, 29, 35]
serie_temporal = pd.Series(datos, index=fechas)

# Acceder a datos utilizando fechas
dato_1 = serie_temporal['2022-01-05']
rango_fechas = serie_temporal['2022-01-03':'2022-01-08']

# Resampling y frecuencias temporales
resample_semanal = serie_temporal.resample('W').mean()

# Operaciones con fechas
dias_transcurridos = serie_temporal.index.max() - serie_temporal.index.
```

TRABAJO CON FECHAS Y SERIES TEMPORALES

Operaciones Avanzadas con Fechas en Pandas:

Pandas proporciona métodos avanzados para trabajar con fechas, como desplazamientos temporales y cálculos de frecuencia.

```
# Desplazamiento temporal hacia adelante y hacia atrás
fechas_desplazadas = serie_temporal.index + pd.DateOffset(days=5)

# Cálculo de frecuencia mensual
conteo_mensual = serie_temporal.resample('M').count()
```

TRABAJO CON FECHAS Y SERIES TEMPORALES

Manejo de Zonas Horarias: Pandas es capaz de trabajar con zonas horarias, lo que es esencial para datos globales o aquellos que atraviesan diferentes husos horarios.

```
import pandas as pd
from datetime import datetime
import pytz

# Crear una serie temporal con zona horaria
fechas = pd.date_range(start='2022-01-01', end='2022-01-10', freq='D', t
serie_temporal = pd.Series(range(len(fechas)), index=fechas)

# Convertir a otra zona horaria
serie_temporal_nueva_zona = serie_temporal.tz_convert('America/New_York'
```

TRABAJO CON FECHAS Y SERIES TEMPORALES

Manejo de Periodos y Duraciones: Pandas permite trabajar con periodos y duraciones, útiles para datos financieros, por ejemplo.

```
# Crear un rango de periodos mensuales
periodos_mensuales = pd.period_range(start='2022-01', end='2022-12',

# Operaciones con periodos
periodo_actualizado = periodos_mensuales + 1

# Crear una duración de tiempo
duracion = pd.Timedelta(days=5, hours=3)
```

TRABAJO CON FECHAS Y SERIES TEMPORALES

Filtrado y Subconjuntos Basados en Fechas: Pandas permite realizar filtrado avanzado basado en fechas.

```
# Filtrar datos para un mes específico
datos_mes = serie_temporal['2022-03']

# Filtrar datos para un rango de fechas
datos_rango = serie_temporal['2022-02-15':'2022-03-15']
```

TRABAJO CON FECHAS Y SERIES TEMPORALES

Resampling y Rolling Windows: Resampling permite cambiar la frecuencia de los datos, y las rolling windows son útiles para realizar cálculos en ventanas móviles.

```
# Resamplear datos a frecuencia mensual
datos_mensuales = serie_temporal.resample('M').sum()

# Calcular la media móvil con una ventana de 3 días
media_movil = serie_temporal.rolling(window=3).mean()
```

TRABAJO CON FECHAS Y SERIES TEMPORALES

Comparación de Periodos: Pandas facilita la comparación y operaciones lógicas con periodos.

```
# Comparar si las fechas están en el mismo año
mismo_anio = serie_temporal.index.to_period('Y') == pd.Period('2022', freq='Y')
```

Manejo de Días Hábiles: Pandas permite trabajar con días hábiles y personalizar calendarios.

```
# Crear un rango de fechas hábiles
fechas_habiles = pd.date_range(start='2022-01-01', end='2022-01-10', freq='B')
```

TRABAJO CON FECHAS Y SERIES TEMPORALES

Frecuencias Temporales

En Pandas, las frecuencias temporales son una parte clave del trabajo con fechas y series temporales. La frecuencia temporal especifica la distancia entre los puntos temporales en una serie y afecta directamente a cómo se realizan operaciones de resampling y otras manipulaciones temporales. Veamos los conceptos fundamentales

1- Especificación de Frecuencias: Las frecuencias temporales se especifican mediante cadenas de texto que indican la frecuencia deseada. Algunos ejemplos comunes incluyen **'D'** para días, **'H'** para horas, **'M'** para meses y **'A'** para años.

```python
import pandas as pd

# Crear un rango de fechas diarias
fechas_diarias = pd.date_range(start='2022-01-01', end='2022-01-10', freq='D')
```

TRABAJO CON FECHAS Y SERIES TEMPORALES

2- **Aliasing de Frecuencias:** Pandas permite utilizar alias para frecuencias comunes. Por ejemplo, **'B'** para días hábiles, **'W'** para semanas, y **'Q'** para trimestres.

```
# Crear un rango de fechas mensuales
fechas_mensuales = pd.date_range(start='2022-01-01', end='2022-12-31', freq='M')
```

3- **Offset Aliasing:** Pandas también admite aliasing mediante "offsets", que son desplazamientos de frecuencia comunes. Por ejemplo, **B** para días hábiles, **H** para horas, y **T** para minutos.

```
# Crear un rango de fechas con frecuencia de 4 horas
fechas_cada_4_horas = pd.date_range(start='2022-01-01', end='2022-01-10', freq='4H')
```

TRABAJO CON FECHAS Y SERIES TEMPORALES

4- **Frecuencias Compuestas:** Se pueden combinar frecuencias para crear frecuencias compuestas. Por ejemplo, **'2D'** para cada 2 días, **'3M'** para cada 3 meses, y así sucesivamente.

```
# Crear un rango de fechas cada 2 semanas
fechas_cada_2_semanas = pd.date_range(start='2022-01-01', end='2022-12-31', freq='2W')
```

5- **Desplazamiento Temporal:** El desplazamiento temporal se refiere a cambiar las fechas basándose en la frecuencia. Esto es útil para calcular promedios móviles y otras operaciones temporales.

```
# Desplazar las fechas 3 días hacia adelante
fechas_desplazadas = fechas_diarias + pd.DateOffset(days=3)
```

TRABAJO CON FECHAS Y SERIES TEMPORALES

6- **Frecuencias Relativas:** Se pueden usar frecuencias relativas para especificar frecuencias basadas en el día de la semana, el mes o el año.

```
# Crear un rango de fechas con frecuencia mensual, pero solo los primeros lunes de cada mes
fechas_primeros_lunes = pd.date_range(start='2022-01-01', end='2022-12-31', freq='WOM-1MON')
```

7- **Frecuencias Irregulares:** Pandas permite manejar frecuencias irregulares mediante el uso de arrays de frecuencia.

```
# Crear un rango de fechas con frecuencia personalizada
fechas_irregulares = pd.date_range(start='2022-01-01', end='2022-01-10', freq=['2D', '3D', '5D'])
```

COMBINACIÓN Y FUSIÓN DE CONJUNTOS DE DATOS

Concatenación de Dataframes

La función **pd.concat()** se utiliza para concatenar DataFrames. La concatenación a lo largo de las filas (**axis=0**) es el comportamiento predeterminado. Si se desea concatenar a lo largo de las columnas, se utiliza **axis=1**.

```
# Concatenar a lo largo de las filas (por defecto)
df_concat_filas = pd.concat([df1, df2])

# Concatenar a lo largo de las columnas
df_concat_columnas = pd.concat([df1, df2], axis=1)
```

Merging de DataFrames:

El método **pd.merge()** realiza merging basándose en una o más columnas comunes. El parámetro **on** especifica la(s) columna(s) de fusión.

```
# Merging basado en la columna 'Key'
df_merge = pd.merge(df3, df4, on='Key')
```

COMBINACIÓN Y FUSIÓN DE CONJUNTOS DE DATOS

Otros parámetros importantes incluyen **how** (tipo de merging: 'inner', 'outer', 'left', 'right') y **suffixes** (sufijos para columnas duplicadas).

```
# Merging externo con sufijos
df_merge_outer = pd.merge(df3, df4, on='Key', how='outer', suffixes=('_df3', '_df4'))
```

Joining de DataFrames:

El método **join()** se utiliza para realizar joining basándose en los índices de los DataFrames. Se puede especificar el tipo de joining utilizando el parámetro **how**.

```
# Joining basado en los índices
df_join = df5.join(df6)
```

COMBINACIÓN Y FUSIÓN DE CONJUNTOS DE DATOS

Especificación de las Claves de Fusión:

Se pueden especificar múltiples columnas o índices para la fusión, permitiendo una combinación más precisa de los DataFrames.

```
# Fusionar DataFrames basándose en múltiples columnas
df_merge_multi = pd.merge(df3, df4, on=['Key'], how='inner')

# Unir DataFrames basándose en múltiples índices
df_join_multi = df5.join(df6, how='inner')
```

Tipos de Merging:

El parámetro **how** permite especificar el tipo de merging. Los tipos comunes son 'inner' (intersección), 'outer' (unión), 'left' (izquierda) y 'right' (derecha).

```
# Merging interno (por defecto)
df_merge_inner = pd.merge(df3, df4, on='Key', how='inner')

# Merging externo (incluye todos los registros)
df_merge_outer = pd.merge(df3, df4, on='Key', how='outer')
```

COMBINACIÓN Y FUSIÓN DE CONJUNTOS DE DATOS

Concatenación de Series:

Además de DataFrames, Pandas permite concatenar Series. La concatenación se realiza a lo largo de las filas (**axis=0**).

```
serie1 = pd.Series(['S0', 'S1', 'S2'], name='Serie1')
serie2 = pd.Series(['S3', 'S4', 'S5'], name='Serie2')

# Concatenar Series a lo largo de las filas
serie_concat_filas = pd.concat([serie1, serie2], axis=1)
```

COMBINACIÓN Y FUSIÓN DE CONJUNTOS DE DATOS

Ejemplo práctico

Supongamos que tenemos datos de ventas diarias de dos productos diferentes en dos ubicaciones diferentes. Queremos analizar y combinar estos conjuntos de datos para obtener una visión más completa.

```python
import numpy as np
import pandas as pd

# Crear datos de ventas diarias para dos productos en dos ubicaciones
fechas = pd.date_range(start='2022-01-01', end='2022-01-10', freq='D')
ubicacion1_ventas = np.random.randint(50, 100, size=len(fechas))
ubicacion2_ventas = np.random.randint(30, 80, size=len(fechas))

# Crear DataFrames para cada ubicación
df_ubicacion1 = pd.DataFrame({'Fecha': fechas, 'Producto_A': ubicacion1_ventas})
df_ubicacion2 = pd.DataFrame({'Fecha': fechas, 'Producto_B': ubicacion2_ventas})

# Manipulación avanzada de datos con Pandas
# Calcular la venta total diaria y agregar una columna de día de la semana
df_ubicacion1['Venta_Total'] = df_ubicacion1['Producto_A']
df_ubicacion2['Venta_Total'] = df_ubicacion2['Producto_B']
df_ubicacion1['Dia_Semana'] = df_ubicacion1['Fecha'].dt.day_name()

# Trabajo con Fechas y Series Temporales en Pandas
# Filtrar datos para el primer lunes de cada mes
df_ubicacion1_lunes = df_ubicacion1[df_ubicacion1['Fecha'].dt.is_month_start & (df_ubicacion1['Fecha'].dt.day_name() == 'Monday')]

# Combinación y Fusión de Conjuntos de Datos
# Concatenar DataFrames a lo largo de las filas
df_concatenado = pd.concat([df_ubicacion1, df_ubicacion2], ignore_index=True)

# Fusionar DataFrames basándose en la columna 'Fecha'
df_fusionado = pd.merge(df_concatenado, df_ubicacion1_lunes[['Fecha', 'Dia_Semana']], on='Fecha', how='left')

print("DataFrame de Ubicación 1:")
print(df_ubicacion1)
print("\nDataFrame de Ubicación 2:")
print(df_ubicacion2)
print("\nDataFrame Concatenado:")
print(df_concatenado)
print("\nDataFrame Fusionado:")
print(df_fusionado)
```

COMBINACIÓN Y FUSIÓN DE CONJUNTOS DE DATOS

Para ver el código completo de todo lo visto en esta clase y hacer seguimiento a las funciones y/o ejercicio proporcionado revisa aquí: https://acortar.link/JJu04O para mejor visualización abrirlo en colab.

¿Preguntas?