COLOMBIA POTENCIA DE LA VIDA

TIC

# UNIVERSIDAD SERGIO ARBOLEDA

Análisis de Datos
Nivel Integrador

TALENTO TECH

UNIVERSIDAD SERGIO ARBOLEDA

# PREPARACIÓN SEMANA 1

# Empecemos con librerías importantes para el análisis de datos en Python.

- Veamos más a detalle todas las bondades de NumPy aquí: La cultura, como elemento de contexto general:

  https://gist.github.com/jofsanchezci/83719d8d77d453759e2ae7a712b118fc

- Y más al detalle Pandas aquí:

  https://gist.github.com/jofsanchezci/377498ace07ca49d470d99322fe1dee4

# LIMPIEZA AVANZADA DE DATOS

La limpieza de datos es crucial para garantizar la calidad de los análisis. Algunas técnicas avanzadas son:

## Transformación de Columnas

```python
# Aplicar una función a una columna
df['columna'] = df['columna'].apply(funcion)

# Crear una nueva columna basada en valores existentes
df['nueva_columna'] = df['columna'].map(lambda x: x * 2)
```

## Manejo de Outliers

```python
# Identificar y manejar outliers
limite_superior = df['columna'].quantile(0.95)
df_sin_outliers = df[df['columna'] < limite_superior]
```

# LIMPIEZA AVANZADA DE DATOS

La limpieza de datos es crucial para garantizar la calidad de los análisis. Algunas técnicas avanzadas son:

## Manejo de Valores nulos:

```
# Eliminar filas con valores nulos
df_sin_nulos = df.dropna()

# Rellenar valores nulos con un valor específico
df_con_relleno = df.fillna(valor_relleno)
```

## Detección y eliminación de duplicados

```
# Detectar duplicados basados en todas las columnas
df_sin_duplicados = df.drop_duplicates()

# Detectar duplicados basados en una columna específica
df_sin_duplicados_columna = df.drop_duplicates(subset='columna')
```

# LIMPIEZA AVANZADA DE DATOS

La limpieza de datos es crucial para garantizar la calidad de los análisis. Algunas técnicas avanzadas son:

## Operaciones de Merge y Join

```
# Combinar dos DataFrames por una columna común
df_combinado = pd.merge(df1, df2, on='columna_comun', how='inner')
```

## Operaciones de Pivot y Melt

```
# Pivotar datos
df_pivotado = df.pivot(index='fila', columns='columna', values='valor')

# Derretir datos
df_derretido = pd.melt(df_pivotado, id_vars='fila', value_vars=['col1', 'col2'])
```

## Operaciones de Ventana(Window Functions)

```
# Calcular promedio móvil
df['promedio_movil'] = df['columna'].rolling(window=3).mean()
```

# Ejemplo

- **Ejemplo 1:** Carga de Dato y limpieza

Acá imaginemos que tenemos un archivo CSV con datos de ventas, donde algunas filas contienen valores nulos, cargaremos os datos, realizaremos algunas operaciones de limpieza y mostraremos el DataFrame resultante.

```
import pandas as pd

# Cargar datos desde un archivo CSV
df_ventas = pd.read_csv('datos_ventas.csv')

# Visualizar el DataFrame original
print("DataFrame Original:")
print(df_ventas)

# Eliminar filas con valores nulos
df_limpiado = df_ventas.dropna()

# Visualizar el DataFrame después de la limpieza
print("\nDataFrame Después de la Limpieza:")
print(df_limpiado)
```

# Ejemplo

- **Ejemplo 2:** Manipulación y Análisis Complejo

Acá consideremos dos conjuntos de datos que representan información de empleados y departamentos. Realizaremos una operación de unión (merge) para combinar estos conjuntos y luego calcularemos el promedio de salarios por departamento

```
# Crear DataFrames de ejemplo
data_empleados = {'ID': [1, 2, 3, 4],
                  'Nombre': ['Alice', 'Bob', 'Charlie', 'David'],
                  'Departamento_ID': [101, 102, 101, 103],
                  'Salario': [60000, 70000, 80000, 75000]}

data_departamentos = {'Departamento_ID': [101, 102, 103],
                      'Departamento': ['Ventas', 'Marketing', 'Desarrollo']}

df_empleados = pd.DataFrame(data_empleados)
df_departamentos = pd.DataFrame(data_departamentos)

# Realizar una operación de unión por la columna 'Departamento_ID'
df_completo = pd.merge(df_empleados, df_departamentos, on='Departamento_ID')

# Calcular el promedio de salarios por departamento
promedio_salarios = df_completo.groupby('Departamento')['Salario'].mean()

# Mostrar el resultado
print("DataFrame Resultante:")
print(df_completo)
print("\nPromedio de Salarios por Departamento:")
print(promedio_salarios)
```

# BREVE REPASO DE PANDAS

**Recordemos…**

- ¿Qué es un DataFrame?
- ¿Cómo cargamos datos en un DataFrame?

# OPERACIONES BÁSICAS CON PANDAS

Las operaciones básicas son aquellas que nos permiten acceder, modificar y crear datos en un DataFrame de Pandas de manera eficiente, entre ellas tenemos:

- Acceder a una columna
- Acceder a varias columnas
- Acceder a una fila por índice
- Acceder a una fila por posición
- Modificar un valor especifico
- Modificar valores en una columna
- Crear una nueva columna basada en otras columnas
- Crear una nueva columna con condiciones
- Operaciones aritméticas básicas
- Uso de funciones integradas de Pandas
- Operaciones elemento a elemento
- Uso de NumPy para operaciones mas complejas

# OPERACIONES BÁSICAS CON PANDAS

Es decir:

```python
# Crear un DataFrame de ejemplo
data = {'columna_1': [1, 2, 3],
        'columna_2': [4, 5, 6],
        'columna': [-1, 0, 1]}

df = pd.DataFrame(data)

# 1. Acceder a Columnas y Filas
# Acceder a una columna
columna_1 = df['columna_1']

# Acceder a varias columnas
columnas_seleccionadas = df[['columna_1', 'columna_2']]

# Acceder a una fila por índice
fila_por_indice = df.loc[1]

# Acceder a una fila por posición
fila_por_posicion = df.iloc[1]

# 2. Modificar Valores
# Modificar un valor específico
df.at[1, 'columna_1'] = 10

# Modificar valores en una columna
df['columna_2'] = df['columna_2'] + 1

# 3. Crear Nuevas Columnas
# Crear una nueva columna basada en otras columnas
df['nueva_columna'] = df['columna_1'] + df['columna_2']

# Crear una nueva columna con condiciones
df['nueva_columna_condicional'] = np.where(df['columna'] > 0, 'positivo', 'negativo')
```

# OPERACIONES BÁSICAS CON PANDAS

Es decir:

```
# 4. Operaciones Aritméticas y Funciones sobre Columnas
# Operaciones aritméticas básicas
df['suma_columnas'] = df['columna_1'] + df['columna_2']

# Uso de funciones integradas de Pandas
df['columna_duplicada'] = df['columna'].apply(lambda x: x * 2)

# Operaciones elemento a elemento con NumPy
df['raiz_cuadrada_columna'] = np.sqrt(df['columna'])

# Imprimir el DataFrame resultante
print(df)
```

# FILTROS Y SELECCIÓN DE DATOS

- **Uso de condiciones para filtrar datos:**
  - Filtrar basado en una condición
  - Filtrar con varias condiciones(AND)
  - Filtrar con varias condiciones (OR)
- **Operadores Lógicos en la selección de datos**
  - Operador ‘isin()’ el cual se utiliza para filtrar filas donde los valores de una columna específica están presentes en una lista dada. Es útil cuando deseas seleccionar filas que contienen valores específicos en una columna.
  - Operador ‘between()’: su función es filtrar filas donde los valores de una columna están dentro de un rango especificado. Esta técnica es útil cuando se desea seleccionar filas basadas en valores numéricos dentro de un rango específico.
- **Uso de métodos ‘loc’ y ‘iloc’ para selección**
  - Selección por etiquetas con ‘loc’: Permite seleccionar filas y columnas basándose en etiquetas (nombres de columnas o índices). Puedes aplicar condiciones a las filas y especificar columnas específicas que deseas incluir en el resultado

# FILTROS Y SELECCIÓN DE DATOS

- **Uso de métodos ‘loc’ y ‘iloc’ para selección**
  - Selección por posición ‘iloc’: Es parecido a ‘loc’ pero utiliza índices de posición en lugar de etiquetas. Permite seleccionar filas y columnas basándose en posiciones numéricas en lugar de nombres de columnas o índices.
  - Selección especifica de celdas con ‘at’ y ‘iat’: se utilizan para seleccionar valores específicos en el DataFrame. at se utiliza con etiquetas, mientras que iat se utiliza con índices de posición. Estas funciones son útiles cuando solo necesitas acceder a un valor específico en el DataFrame.

# FILTROS Y SELECCIÓN DE DATOS

Es decir:

```python
# Crear un DataFrame de ejemplo
data = {'columna_1': [1, 2, 3],
        'columna_2': [4, 5, 6],
        'columna': [-1, 0, 1]}

df = pd.DataFrame(data)

# 1. Uso de Condiciones para Filtrar Datos
# Filtrar basado en una condición
df_filtrado_simple = df[df['columna'] > 0]

# Filtrar con múltiples condiciones (AND)
df_filtrado_and = df[(df['columna_1'] > 0) & (df['columna_2'] < 10)]

# Filtrar con múltiples condiciones (OR)
df_filtrado_or = df[(df['columna_1'] > 0) | (df['columna_2'] < 10)]

# 2. Operadores Lógicos en la Selección de Datos
# Operador isin()
valores_a_seleccionar = [1, 3, 5]
df_filtrado_isin = df[df['columna'].isin(valores_a_seleccionar)]

# Operador between()
df_filtrado_between = df[df['columna'].between(2, 5)]
```

# FILTROS Y SELECCIÓN DE DATOS

Es decir:

```python
# 3. Uso de Métodos loc e iloc para Selección
# Selección por etiquetas con loc
df_seleccionado_loc = df.loc[df['columna'] > 0, ['columna_1', 'columna_2']]

# Selección por posición con iloc
df_seleccionado_iloc = df.iloc[1:3, 0:2]

# Selección específica de celdas con at e iat
valor_celda_etiqueta = df.at[1, 'columna_1']
valor_celda_posicion = df.iat[1, 0]

# Imprimir el DataFrame resultante
print("DataFrame Original:")
print(df)
print("\nResultados de Filtrado y Selección:")
print("Filtrado Simple:")
print(df_filtrado_simple)
print("\nFiltrado AND:")
print(df_filtrado_and)
print("\nFiltrado OR:")
print(df_filtrado_or)
print("\nFiltrado con isin():")
print(df_filtrado_isin)
print("\nFiltrado con between():")
print(df_filtrado_between)
print("\nSelección con loc:")
print(df_seleccionado_loc)
print("\nSelección con iloc:")
print(df_seleccionado_iloc)
print("\nValores específicos con at e iat:")
print("Valor en la posición [1, 'columna_1']: ", valor_celda_etiqueta)
print("Valor en la posición [1, 0]: ", valor_celda_posicion)
```

# OPERACIONES DE AGRUPACIÓN Y CÁLCULOS BÁSICOS

Podemos realizar agrupación de datos usando el método ‘groupby’ de la siguiente manera:

```
grupos = df.groupby('columna')
```

**Agrupar por una columna**

```
grupos = df.groupby(['columna_1', 'columna_2'])
```

**Agrupar por múltiples columnas**

Posterior a dicha agrupación usando ‘groupby’ es común realizar operaciones agregadas, es decir, aplicar funciones que resumen o agregan información dentro de cada grupo. Algunas funciones agregadas comunes incluyen calcular la suma, el promedio, el mínimo, el máximo, la desviación estándar, etc.

# OPERACIONES DE AGRUPACIÓN Y CÁLCULOS BÁSICOS

Esto lo podríamos realizar así:

```python
#Agrupamos
grupos = df.groupby(['columna_1', 'columna_2'])

#Funcion para operacion agregada
def funcion_personalizada(x):
    return x.max() - x.min()

resultado_personalizado = grupos['columna_2'].agg(funcion_personalizada)
```

También se pueden realizar cálculos básicos como suma, promedio y conteo así:

```python
suma_por_grupo = grupos['columna_1'].sum()
```

**Suma por Grupo**

```python
promedio_por_grupo = grupos['columna_2'].mean()
```

**Promedio por Grupo**

```python
conteo_por_grupo = grupos.size()
```

**Conteo de Elementos por grupo**

# INTRODUCCIÓN A LA VISUALIZACIÓN DE DATOS CON PANDAS

Pandas proporciona métodos de visualización básicos que permiten crear gráficos directamente desde un DataFrame. Estos métodos son convenientes para realizar exploración visual de los datos antes de realizar análisis más detallados con bibliotecas de visualización más avanzadas como Matplotlib y Seaborn.

Entre los gráficos básicos tenemos:

- Gráficos de Líneas y Áreas

```
df['columna'].plot()
```

**Grafico de líneas para una columna**

```
df['columna'].plot.area()
```

**Grafico de área para una columna**

# INTRODUCCIÓN A LA VISUALIZACIÓN DE DATOS CON PANDAS

- Gráficos de Barras e Histogramas

```
df['columna'].plot.bar()
```

**Grafico de barras para una columna**

```
df['columna'].plot.hist()
```

**Histograma para una columna**

```
df.plot.scatter(x='columna_1', y='columna_2')
```

**Gráficos de Dispersión**

```
df['columna'].value_counts().plot.pie()
```

**Gráficos de Torta**

```
df['fecha_columna'] = pd.to_datetime(df['fecha_columna'])
df.set_index('fecha_columna')['columna'].plot()
```

**Gráficos Temporales**

# EJEMPLOS

- **Ejemplo 1:** Agrupación y funciones agregadas

```python
import pandas as pd

# Crear un DataFrame de ejemplo
data = {'Grupo': ['A', 'A', 'B', 'B', 'A', 'B'],
        'Valor_1': [10, 15, 20, 25, 30, 35],
        'Valor_2': [5, 8, 12, 18, 22, 28]}

df = pd.DataFrame(data)

# Agrupar por la columna 'Grupo'
grupos = df.groupby('Grupo')

# Calcular la suma y el promedio por grupo
suma_por_grupo = grupos['Valor_1'].sum()
promedio_por_grupo = grupos['Valor_2'].mean()

print("Suma por Grupo:")
print(suma_por_grupo)

print("\nPromedio por Grupo:")
print(promedio_por_grupo)
```

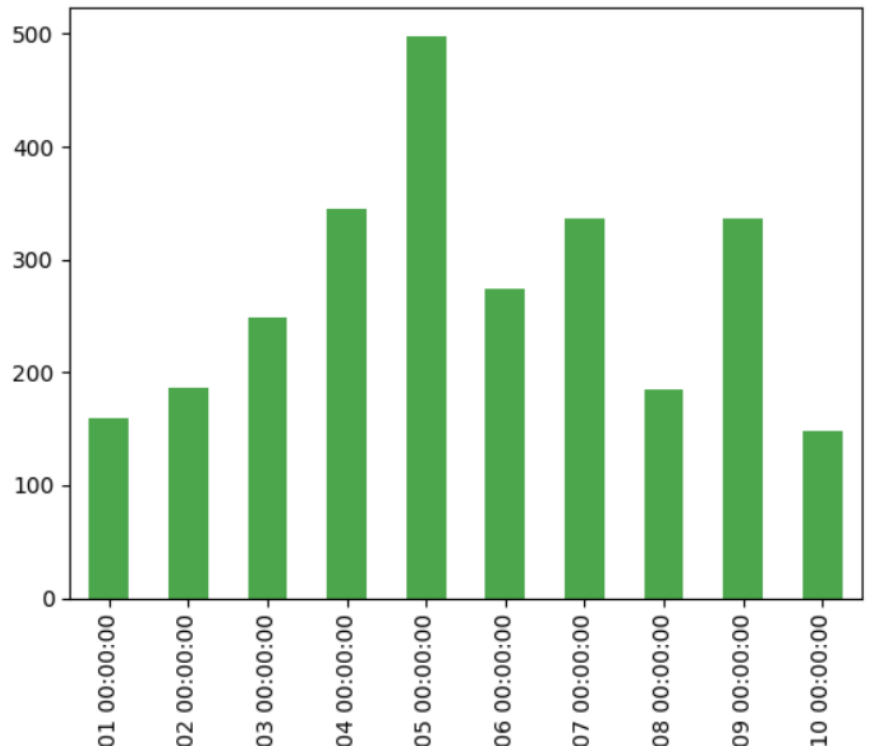
# EJEMPLOS

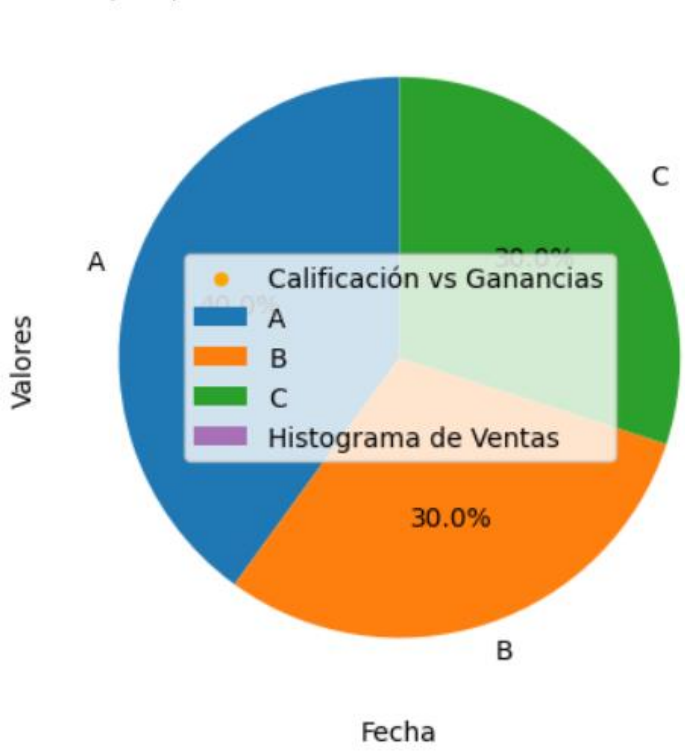
- **Ejemplo 2:** Visualización de Datos en gráficos

```python
# Crear un DataFrame de ejemplo
data = {'Fecha': pd.date_range(start='2022-01-01', periods=10),
        'Ventas': np.random.randint(50, 200, size=10),
        'Ganancias': np.random.uniform(100, 500, size=10),
        'Producto': ['A', 'B', 'A', 'C', 'B', 'C', 'A', 'B', 'C', 'A'],
        'Calificación': np.random.randint(1, 6, size=10)}

df = pd.DataFrame(data)

# Configurar el índice como la columna 'Fecha'
df.set_index('Fecha', inplace=True)

# Crear gráfico de líneas para 'Ventas'
df['Ventas'].plot(label='Ventas', marker='o', linestyle='-', color='blue')

# Crear gráfico de barras para 'Ganancias'
df['Ganancias'].plot.bar(label='Ganancias', color='green', alpha=0.7)

# Crear gráfico de dispersión para 'Calificación' vs 'Ganancias'
df.plot.scatter(x='Calificación', y='Ganancias', label='Calificación vs Ganancias', color='orange')

# Crear gráfico de torta para la distribución de 'Producto'
df['Producto'].value_counts().plot.pie(label='Distribución de Producto', autopct='%1.1f%%', startangle=90)

# Crear histograma para 'Ventas'
df['Ventas'].plot.hist(label='Histograma de Ventas', bins=10, alpha=0.5, color='purple')

# Personalizar el gráfico
plt.title('Ejemplo de Gráficos Básicos en Pandas')
plt.xlabel('Fecha')
plt.ylabel('Valores')
plt.legend()
plt.grid(True)
plt.show()
```

# EJEMPLOS

- **Ejemplo 2:** Al ejecutar el anterior código podremos ver los diferentes gráficos



# ¿Preguntas?