Unidad N°2 Aplicación de herramientas informaticas al Ciclo de Vida de los Datos - Fase II

Sitio:	Centro de E-Learning - UTN.BA	Imprimido por:	Virginia Marich
Curso:	Curso de Data Science	Día:	Wednesday, 27 de December de 2023, 09:52
Libro:	Unidad N°2 Aplicación de herramientas informaticas al Ciclo de Vida de los Datos - Fase II		

Descripción

Tabla de contenidos

1. Introducción

Bienvenidos a esta unidad de formación en la que se abordarán dos temas importantes relacionados con la ciencia de datos y la programación. En el primer tema, explicaremos por qué es importante aprender un lenguaje de programación en la ciencia de datos, y en particular, por qué Python es una excelente opción para aquellos que desean incursionar en este campo. También discutiremos los IDE o entornos de desarrollo más utilizados en Python, como Jupyter Notebook y Google Colab.

En el segundo tema, nos enfocaremos en las funcionalidades y librerías clave de Python para el procesamiento y análisis exploratorio de datos. Aprenderás sobre cómo utilizar estas herramientas para explorar y visualizar datos, y cómo implementar algoritmos de aprendizaje automático para el análisis predictivo.

Al final de esta unidad, tendrás una sólida comprensión de por qué Python es un lenguaje de programación esencial en la ciencia de datos, cómo utilizar los entornos de desarrollo más populares, y cómo aplicar las funcionalidades y librerías clave de Python para el procesamiento y análisis exploratorio de datos. ¡Comencemos!

2. ¿Por qué necesito aprender un lenguaje de programación en la ciencia de datos?. Elección de Python y sus IDE o entornos de desarrollo más utilizados: Jupyter Notebook & Google Colab

¿Por qué necesito aprender un lenguaje de programación en la ciencia de datos?

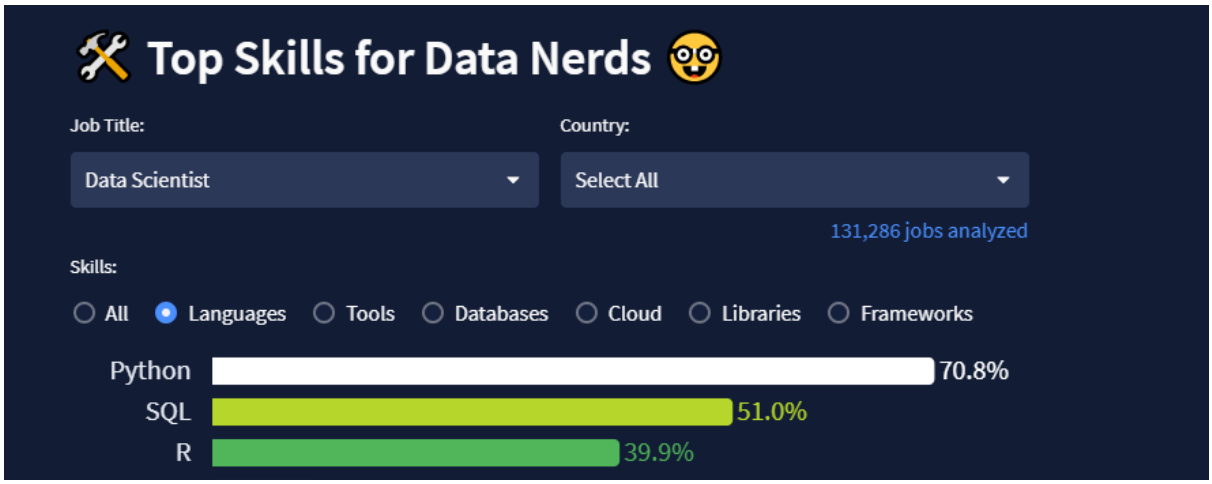
Hasta ahora, hemos estado explicando cada unidad con pequeños ejemplos de datos. Sin embargo, en una gran compañía, trabajar con datos no se limita a bases de datos con solo 5 columnas y 4 filas. Para un científico de datos, trabajar con datos significa enfrentarse a BIG DATA y trabajar con múltiples fuentes. El manejo de un gran volumen de datos y múltiples conexiones hace que las herramientas de análisis comunes ya no sean viables.

Todo lo que has aprendido hasta ahora te permite extraer datos (SQL), explorar y analizarlos (con conocimientos estadísticos) para, finalmente, visualizar e informar sobre ellos (usando TABLEAU o Power BI). Pero, ¿es esa la única razón por la que viniste a este curso? ¿Quieres convertirte en un analista de datos o aspiras a convertirte en un científico de datos?

Para convertirte en un CIENTÍFICO DE DATOS o DATA SCIENTIST, necesitas aprender a programar. Un científico de datos aporta valor realizando todo lo que hace un analista de datos, pero con BIG DATA y además, extrayendo un valor mayor. Puede realizar predicciones a través de modelos para anticiparse a los hechos, pero esto es imposible hacerlo en una hoja de papel o con los softwares que hemos venido conociendo. Necesitamos la ayuda de nuestras computadoras, las cuales serán nuestro principal aliado para analizar y generar algoritmos que nos ayuden a entender grandes volúmenes de datos. Para comunicarnos con nuestras computadoras, debemos aprender su lenguaje: la programación.

Según varias encuestas y análisis los lenguajes de programación más usados para la ciencia de datos son R y Python,¿te acordas de la web que te mencione en la primera unidad? Skills (datanerd.tech)

Es impresionante pero estos lenguajes no son solo solicitados para los científicos de datos, sino que se pueden utilizar y son requeridos para múltiples profesiones en el mundo actual. Si nos concentramos en el Data Science, tenemos lo siguiente:



Hay tres que son los más populares y los que se exigen más frecuentemente en las ofertas de trabajo: SQL, Python y R. Pero no es necesario que un Data Scientist sea experto en los tres. Normalmente, se pide que maneje SQL y que trabaje con Python o con R.

Se considera que Python ocupa el primer lugar en la lista de lenguajes de programación debido a su simplicidad. Las sintaxis de Python son muy simples y se pueden aprender fácilmente, por lo que en esta unidad nos centraremos en aprender su uso aplicado al data science.

2.1. Elección de Python y sus IDE o entornos de desarrollo más utilizados: Jupyter Notebook & Google Colab

Python es una de las mejores opciones para la ciencia de datos debido a su amplia gama de funcionalidades, librerías y su enfoque en la simplicidad y flexibilidad. Además, es un lenguaje de programación de código abierto, lo que significa que su comunidad de desarrolladores ha creado una gran cantidad de recursos y documentación en línea. Esto hace que Python sea fácil de aprender y utilizar para cualquier persona interesada en la ciencia de datos.

Otra ventaja de Python es su escalabilidad, lo que lo hace ideal para proyectos tanto pequeños como grandes. Por último, permíteme mencionar que si es tu primera vez aprendiendo a programar para entender más la ciencia de datos o incluso luego deseas mantener tus opciones de carrera abiertas, Python es una excelente opción para empezar, ya que es ampliamente utilizado en la comunidad de desarrollo e IT.

2.2. ¿Por dónde empezar para entender Python?¿Qué es un IDE?

Primero debemos saber que así como los escritores tienen cuadernos o editores de texto y los contadores usan hojas de cálculo para hacer su trabajo, los programadores, en este caso científicos de datos, también utilizan un editor para escribir su código. A esto se le llama IDE, que significa Integrated Development Environment o Entorno de Desarrollo Integrado en español.

Además de funcionar como "editores de texto o de código", los IDEs incorporan varias funcionalidades. Conocen las reglas del lenguaje, permiten el autocompletado y proporcionan retroalimentación sobre errores de escritura, lo que hace que programar sea mucho más intuitivo. Pero eso no es todo: para entender la profundidad de esto, es necesario saber que las computadoras sólo entienden un lenguaje, el lenguaje binario. Dentro de los IDEs se integran "intérpretes" como si fueran traductores para traducir el lenguaje, en este caso Python, que es un lenguaje intermedio entre los humanos y las computadoras, hacia el lenguaje binario que realmente entiende la computadora y utiliza para ejecutar las instrucciones que se le dan.

Existen muchos de estos entornos para Python, los que abordaremos en este curso serán: Jupyter Notebook y Google Colab, los cuales tienen una interfaz que permite una interacción agradable para el primer aprendizaje de este poderoso lenguaje.

Tutorial de Instalación de Jupyter Notebook y conocimiento de la interfaz en video

Tutorial de Instalación de Google Colab y conocimiento de la interfaz en video

A lo largo de esta sección introduciremos funcionalidades y librerías básicas de Python a través de Google Colab.

3. Data Science con Python: Introducción a las Funcionalidades y Librerías Clave para el Procesamiento y Análisis Exploratorio de Datos

Sintaxis y funcionalidades básicas

La sintaxis básica de Python en Google Colab es la misma que en cualquier otra plataforma de programación con Python. Google Colab utiliza Jupyter Notebook como su entorno de programación, lo que significa que puede escribir y ejecutar código Python en bloques de código llamados "celdas".

Para crear una nueva celda en Google Colab, haga clic en el botón "+ Code" en la barra de herramientas o presione "Ctrl+M" y luego "B". Una vez que se crea una celda, se puede escribir código Python en ella.

Sintaxis básica de Python

¿Qué tipos de datos se pueden asignar a las variables?

En Python, se pueden asignar diferentes tipos de datos a las variables, incluyendo:

1. Enteros (int): números enteros, como 1, 2, 3, -4, -5, etc.
2. Flotantes (float): números decimales, como 1.0, 2.5, -3.14, etc.
3. Booleanos (bool): valores lógicos True o False.
4. Cadenas de caracteres (str): secuencias de caracteres, como "hola", "mundo", "123", etc.
5. Listas (list): colecciones ordenadas y modificables de elementos, como [1, 2, 3], ["hola", "mundo"], etc.
6. Tuplas (tuple): colecciones ordenadas e inmutables de elementos, como (1, 2, 3), ("hola", "mundo"), etc.
7. Conjuntos (set): colecciones no ordenadas y sin elementos repetidos, como {1, 2, 3}, {"hola", "mundo"}, etc.
8. Diccionarios (dict): colecciones no ordenadas de pares clave-valor, como {"nombre": "Juan", "edad": 25}, {"producto": "libro", "precio": 10.99}, etc.
9. None: representa la ausencia de valor

Se pueden utilizar funciones para realizar la conversión de tipos de datos. Algunas de las funciones más comunes para convertir tipos de datos son:

- int(): convierte un valor a un número entero.
- float(): convierte un valor a un número flotante.
- str(): convierte un valor a una cadena de caracteres.
- bool(): convierte un valor a un valor booleano.

Para utilizar estas funciones, simplemente se llama a la función con el valor que se desea convertir como argumento. Por ejemplo, para convertir un valor numérico a una cadena de caracteres, se puede utilizar la función str():

```
num = 42
```

```
cadena = str(num)
```

Python es un lenguaje orientado a objetos, cuenta con:

- Datos
- Atributos o propiedades (un punto y una palabra sin paréntesis):

x.atributo

- Funcionalidad o métodos (un punto y una palabra con paréntesis):

x.método(). Un método es algo que el dato puede hacer, por lo tanto al ejecutarlo le estamos pidiendo al dato que ejecute una acción

Ejemplificación para entender que es un atributo y un método:

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre    # Atributo de instancia: nombre
        self.edad = edad        # Atributo de instancia: edad

    def saludar(self):
        print(f"Hola, mi nombre es {self.nombre} y tengo {self.edad} años.")   # Método: saludar

    def cumplir_anios(self):
        self.edad += 1          # Método: cumplir_anios. Modifica el atributo de instancia: edad

# Creación de un objeto de la clase Persona
persona1 = Persona("Juan", 25)

# Acceso a atributos de instancia
print(persona1.nombre)  # Salida: Juan
print(persona1.edad)    # Salida: 25

# Llamada a un método
persona1.saludar()      # Salida: Hola, mi nombre es Juan y tengo 25 años.

# Modificación de un atributo de instancia a través de un método
persona1.cumplir_anios()
print(persona1.edad)    # Salida: 26
```

En este ejemplo, se puede observar la diferencia entre los atributos y los métodos. Los atributos nombre y edad son utilizados para almacenar información en los objetos de la clase Persona, mientras que los

métodos saludar() y cumplir_anios() definen el comportamiento del objeto y pueden realizar acciones o modificar los atributos del objeto.

3.1. Estructuras de datos esenciales: listas, tuplas, conjuntos y diccionario

En el mundo del análisis de datos y ciencia de datos, es esencial conocer las estructuras de datos fundamentales en Python: listas, tuplas, conjuntos y diccionarios. A continuación, se detalla cada una de estas estructuras y se presentan algunos ejemplos prácticos para ilustrar su uso en la ciencia de datos.

LISTAS

1. Las listas son una de las estructuras de datos más comunes en Python. Son colecciones ordenadas y modificables de elementos. Cada elemento en una lista puede ser de un tipo de datos diferente. Las listas se crean utilizando corchetes [] y separando cada elemento con una coma.

```
# Crear una lista
lista = [1, 2, "tres", True, 5.6]

# Acceder a elementos de la lista
print(lista[0])  # 1
print(lista[2])  # "tres"

# Modificar elementos de la lista
lista[1] = "dos"
print(lista)  # [1, "dos", "tres", True, 5.6]

# Agregar elementos a la lista
lista.append(7)
print(lista)  # [1, "dos", "tres", True, 5.6, 7]
```

TUPLAS

Las tuplas son similares a las listas, pero son inmutables, lo que significa que no se pueden modificar después de su creación. Las tuplas se crean utilizando paréntesis () y separando cada elemento con una coma.

```
# Crear una tupla
tupla = (1, 2, "tres", True, 5.6)

# Acceder a elementos de la tupla
print(tupla[0])  # 1
print(tupla[2])  # "tres"
```

CONJUNTOS

Los conjuntos son colecciones desordenadas de elementos únicos. No se puede acceder a los elementos de un conjunto mediante índices. Los conjuntos se crean utilizando llaves {} o la función set().

Ejemplo:

```
# Crear un conjunto
conjunto = {1, 2, "tres", True, 5.6}

# Agregar elementos al conjunto
conjunto.add(7)

# Eliminar elementos del conjunto
conjunto.remove("tres")
```

DICCIONARIOS

Los diccionarios son colecciones desordenadas de pares clave-valor. Las claves son únicas y los valores pueden ser de diferentes tipos de datos. Los diccionarios se crean utilizando llaves {} y separando cada par clave-valor con dos puntos : y cada elemento con una coma.

```
# Crear un diccionario
diccionario = {"nombre": "Juan", "edad": 35, "ciudad": "Lima"}

# Acceder a valores del diccionario
print(diccionario["nombre"])  # "Juan"
print(diccionario["edad"])  # 35

# Agregar nuevos pares clave-valor al diccionario
diccionario["profesion"] = "Ingeniero"
```

3.2. Sentencias condicionales, ciclos for y while

Las sentencias condicionales, ciclos for y ciclos while son estructuras fundamentales en la programación que permiten controlar el flujo de ejecución del programa y realizar tareas repetitivas.

Las sentencias condicionales se utilizan para ejecutar un bloque de código sólo si se cumple una condición determinada.

IF

En Python, se utiliza la palabra clave "if" seguida de la condición y un bloque de código indentado que se ejecutará si la condición es verdadera. Por ejemplo:

```
x = 5
if x > 0:
    print("x es positivo")
```

En este ejemplo, se evalúa la condición "x > 0" y si es verdadera, se ejecuta la instrucción "print("x es positivo")".

FOR

Los ciclos for se utilizan para iterar sobre una secuencia de elementos, como una lista o una cadena de texto, y ejecutar un bloque de código para cada elemento. En Python, se utiliza la palabra clave "for" seguida de una variable que tomará el valor de cada elemento de la secuencia, la palabra clave "in" y la secuencia. Por ejemplo:

```
lista = [1, 2, 3, 4, 5]
for elemento in lista:
    print(elemento)
```

En este ejemplo, se itera sobre la lista de elementos y se imprime cada elemento en la consola.

WHILE

Los ciclos while se utilizan para repetir un bloque de código mientras se cumpla una condición determinada. En Python, se utiliza la palabra clave "while" seguida de la condición y un bloque de código indentado que se repetirá mientras la condición sea verdadera. Por ejemplo:

```
i = 0
while i < 5:
    print(i)
```

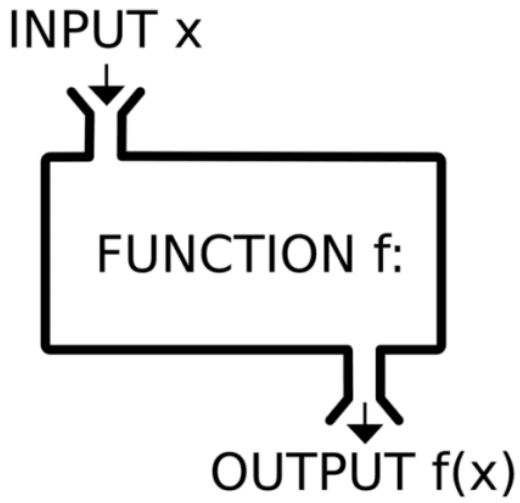
```
    i += 1
```

En este ejemplo, se imprime el valor de la variable "i" mientras sea menor que 5, y se incrementa su valor en cada iteración del ciclo.

Es importante tener en cuenta que en Python, la indentación es fundamental para indicar el inicio y fin de un bloque de código. En el caso de las sentencias condicionales y ciclos, el bloque de código que se ejecuta si se cumple la condición o en cada iteración del ciclo debe estar indentado.

FUNCIONES

- Para trabajar profesionalmente en programación, el código que se usa en forma repetitiva se organiza en funciones.
- Puede hacerse una analogía con una función matemática y = f(x): la función f recibe un argumento x, ejecuta una serie de comandos y devuelve un valor y.



La sintaxis para definir una función es la siguiente:

```
def nombre_de_la_funcion(parametros):
    # cuerpo de la función
    return valor_de_retorno
```

- def es la palabra clave que indica que se está definiendo una función.
- nombre_de_la_funcion es el nombre que se le dará a la función. Este debe seguir las mismas reglas que los nombres de las variables.
- parametros son los valores que se le pasan a la función para que los utilice en su ejecución. Pueden ser opcionales o requeridos.
- El cuerpo de la función es el conjunto de instrucciones que la función ejecutará cada vez que sea llamada. Debe estar indentado con cuatro espacios.
- return es la instrucción que indica el valor que devolverá la función. Es opcional y puede devolver cualquier tipo de datos.

Ejemplo de función que suma dos números:

```
def sumar(num1, num2):
    resultado = num1 + num2
    return resultado
```

Para llamar a la función, simplemente se escribe su nombre seguido de los valores que se quieren pasar como argumentos:

```
resultado_suma = sumar(5, 7)
```

Una vez que hayamos cubierto la introducción a la sintaxis básica y funcionalidades del lenguaje, es importante reflexionar sobre la importancia del procesamiento y análisis exploratorio de datos en cualquier proyecto de Data Science. Estos pasos son fundamentales, ya que nos permiten comprender y obtener información valiosa de los datos antes de aplicar técnicas de modelado. En Python, utilizaremos algunas de las librerías más populares y ampliamente utilizadas, como Pandas, NumPy, Matplotlib y Seaborn, para llevar a cabo estas tareas. Con Pandas, podremos cargar, limpiar y transformar datos tabulares en DataFrames, una estructura de datos poderosa que facilita la manipulación y análisis de datos. Además, utilizaremos NumPy para realizar cálculos numéricos y operaciones matemáticas avanzadas en nuestros datos. También exploraremos cómo crear visualizaciones atractivas e informativas con Matplotlib y Seaborn, lo cual nos permitirá analizar y comunicar de manera efectiva los patrones y tendencias presentes en los datos.

3.3. ¿Qué son las librerías?

Las librerías (o bibliotecas, en algunos casos) son conjuntos de código predefinido que ofrecen una serie de funciones y herramientas que se pueden utilizar en proyectos de programación.

En lugar de tener que escribir todo el código desde cero para realizar ciertas tareas, se pueden importar las librerías y utilizar sus funciones ya definidas. Esto permite ahorrar tiempo y evitar errores comunes en la implementación de ciertas funciones.

Las librerías son creadas por desarrolladores de software que tienen experiencia en un área específica y quieren ofrecer herramientas y soluciones a otros programadores que trabajan en ese campo.

Por ejemplo, Numpy fue creado por Travis Olliphant en 2005, quien era un desarrollador de software científico. Olliphant estaba trabajando en un proyecto de análisis de datos en Python y notó que las herramientas básicas de Python no eran suficientes para realizar operaciones matemáticas complejas y eficientes. Así que decidió crear Numpy como una librería que permitiera a los programadores trabajar con matrices y realizar operaciones matemáticas de forma más eficiente.

Introducción a Numpy para operaciones matemáticas

Numpy es una librería fundamental para la computación científica, es decir, para realizar operaciones matemáticas, por eso es el origen de su nombre “Num” por la solución que brinda de manera matemática y “Py” por el lenguaje.

Numpy es una librería de código abierto en Python que ofrece una gran cantidad de funciones y herramientas para el procesamiento y análisis de datos numéricos. Una de las características principales de Numpy es su capacidad para trabajar con arreglos multidimensionales de manera eficiente y rápida.

Para comenzar la explicación y su aplicación en Python, nos contextualizamos en Google Colab. Para comenzar a utilizar una librería es necesario “importarla”, para ello se escribe el siguiente código:

```
import Numpy as np
```

Los arreglos en Numpy se denominan “Array”, Un array es una estructura de datos que consiste en una colección de elementos (valores o variables), cada uno identificado por al menos un índice o clave. Un array se almacena de modo que la posición de cada elemento se pueda calcular a partir de su tupla de índice mediante una fórmula matemática. El tipo más simple de array es un array lineal, también llamado array unidimensional.

En numpy:

- Cada dimensión se denomina axis
- El número de dimensiones se denomina rank
- La lista de dimensiones con su correspondiente longitud se denomina shape
- El número total de elementos (multiplicación de la longitud de las dimensiones) se denomina size

Para ver en el código: Bases de Numpy

Aunque los arrays son una herramienta útil para trabajar con datos numéricos, pueden presentar algunas incomodidades al trabajar con tablas.

Algunas de las incomodidades que se pueden presentar al trabajar con arrays incluyen:

1. Los arrays sólo permiten un tipo de dato: Esto significa que todos los elementos del array deben ser del mismo tipo de datos, lo que puede ser limitante si se están trabajando con datos heterogéneos.
2. Pérdida de referencia de los nombres de las columnas: Al trabajar con arrays, es necesario recordar el índice o la posición de cada columna, lo que puede resultar difícil y confuso si se está trabajando con un gran número de columnas.

Es por esto que se desarrolló la librería “Pandas” para trabajar con tablas. A diferencia de los arrays de NumPy, que son adecuados para datos homogéneos, las estructuras de datos de Pandas, como los Data Frames, están diseñadas para manejar datos heterogéneos en forma de tablas.

3.4. Introducción a Pandas para la manipulación y análisis de datos

Pandas es una de las herramientas más populares para la manipulación y análisis de datos en Python. Fue creada por Wes McKinney en 2008 y se ha convertido en una de las bibliotecas de Python más utilizadas en el mundo de la ciencia de datos.

- Pandas proporciona estructuras de datos flexibles y de alto rendimiento para trabajar con datos tabulares, como hojas de cálculo y bases de datos. El objeto fundamental de Pandas es el Data Frame, que es una tabla de datos bidimensional con etiquetas de fila y columna, similar a una hoja de cálculo o una tabla de una base de datos relacional. Los Data Frames en Pandas pueden contener diferentes tipos de datos en cada columna, incluyendo números, cadenas, objetos y valores faltantes.
- Pandas también proporciona una amplia variedad de herramientas para manipular y transformar datos, como seleccionar y filtrar datos, agregar y combinar tablas de datos, realizar cálculos estadísticos y realizar visualizaciones.
- Además, Pandas tiene una estrecha integración con otras bibliotecas de análisis de datos de Python, como NumPy y Matplotlib, lo que lo convierte en una herramienta poderosa y completa para la ciencia de datos.

Pandas es una herramienta esencial para el análisis de datos en Python porque permite trabajar con datos estructurados y manipularlos de manera eficiente y sencilla. Pandas proporciona una amplia gama de funcionalidades para la importación, limpieza, transformación y análisis de datos, lo que hace que sea fácil trabajar con datos de diferentes fuentes y en diferentes formatos.

Algunas de las razones por las que Pandas es importante para el análisis de datos incluyen:

- Permite la carga y procesamiento de datos en diferentes formatos, incluyendo CSV, Excel, SQL y HTML.
- Permite la limpieza y transformación de datos, lo que incluye la eliminación de valores nulos, la selección de subconjuntos de datos y la agregación de datos.
- Permite la manipulación de datos, incluyendo la selección y filtrado de datos, la unión y combinación de tablas, y la creación de nuevas variables y variables dummy.
- Ofrece herramientas para el análisis estadístico, incluyendo la agregación de datos, la correlación y regresión, y el análisis de series de tiempo.
- Proporciona una amplia gama de herramientas de visualización, incluyendo histogramas, gráficos de barras y de dispersión, y mapas de calor, que permiten la exploración y comunicación de datos de manera efectiva.

Para comenzar la explicación y su aplicación en Python, nos contextualizamos en Google Colab. Para comenzar a utilizar una librería es necesario “importarla”, para ello se escribe el siguiente código:

```
import pandas as pd
```

3.5. Series y DataFrames: ¿Qué son y cómo se crean?

En Pandas, una serie es un objeto unidimensional que contiene un conjunto de datos etiquetados. Cada elemento de la serie está indexado por una etiqueta. Podemos pensar en una serie como una columna en una tabla de datos.

Podemos crear una serie a partir de una lista, un arreglo NumPy o un diccionario. Por ejemplo:

```
import pandas as pd
import numpy as np

# Crear una Serie a partir de una lista
lista = [10, 20, 30, 40]
serie = pd.Series(lista)
print(serie)

# Crear una Serie a partir de un arreglo NumPy
arreglo = np.array([1, 2, 3, 4])
serie = pd.Series(arreglo)
print(serie)

# Crear una Serie a partir de un diccionario
diccionario = {'a': 10, 'b': 20, 'c': 30}
serie = pd.Series(diccionario)
print(serie)
```

Por otro lado, un Data Frame es una estructura de datos bidimensional que se utiliza para almacenar y manipular datos tabulares. Cada columna de un DataFrame es una serie. Podemos pensar en un DataFrame como una tabla de datos.

Podemos crear un DataFrame a partir de un diccionario, una lista de diccionarios, una lista de listas, un archivo CSV, una consulta SQL y muchos otros formatos de datos.

Por ejemplo:

```
import pandas as pd

# Crear un DataFrame a partir de un diccionario
datos = {'nombre': ['Juan', 'María', 'Pedro', 'Lucía'],
        'edad': [25, 30, 20, 35],
```

```
        'ciudad': ['Bogotá', 'Medellín', 'Cali', 'Barranquilla']}
df = pd.DataFrame(datos)
print(df)

# Crear un DataFrame a partir de una lista de listas
datos = [['Juan', 25, 'Bogotá'], ['María', 30, 'Medellín'], ['Pedro', 20, 'Cali'], ['Lucía', 35, 'Barranquilla']]
df = pd.DataFrame(datos, columns=['nombre', 'edad', 'ciudad'])
print(df)
```

3.6. Importación y exportación de datos: Cómo leer y escribir datos en diferentes formatos de archivo, como CSV y Excel

En Pandas, se puede leer y escribir datos en diferentes formatos de archivo, como CSV, Excel, JSON, HTML, SQL, entre otros.

Para leer datos de un archivo CSV, por ejemplo, se utiliza la función read_csv() de Pandas, que permite cargar los datos en un DataFrame. Por ejemplo:

```
import pandas as pd

data = pd.read_csv('datos.csv')
```

Para exportar datos a un archivo CSV, se utiliza la función to_csv(). Por ejemplo:

```
import pandas as pd

data = pd.read_csv('datos.csv')

data.to_csv('nuevos_datos.csv', index=False)
```

En este caso, index=False indica que no se deben incluir los índices de las filas en el archivo CSV de salida.

Para leer y escribir en otros formatos de archivo, como Excel, HTML o SQL, se utilizan funciones similares, como read_excel(), to_excel(), read_html(), to_sql(), entre otras.

Al importar datos con Pandas, es común encontrarse con diferentes errores que pueden dificultar la lectura o el análisis de los datos. Algunos de los errores más comunes son:

1. Error de codificación: Este error ocurre cuando los datos están en un formato de codificación diferente al que utiliza Pandas por defecto. Para solucionarlo, se puede especificar la codificación correcta al leer el archivo usando el parámetro "encoding".

```
import pandas as pd df = pd.read_csv("datos.csv", encoding='utf-8')
```

2. Error de formato: Este error se produce cuando los datos en el archivo no se ajustan al formato especificado por el usuario. Por ejemplo, si se intenta leer un archivo CSV con columnas separadas por comas, pero algunas filas contienen valores con comas adicionales, Pandas puede interpretarlos como columnas adicionales. Para solucionarlo, se puede especificar el delimitador correcto usando el parámetro "delimiter" o "sep".

```
import pandas as pd df = pd.read_csv("datos.csv", delimiter=';')
```

3. Error de valores faltantes: Cuando los datos contienen valores faltantes, Pandas puede tener dificultades para leerlos correctamente. Por defecto, Pandas rellena los valores faltantes con

"NaN". Se pueden usar diferentes métodos para manejar los valores faltantes, como eliminar las filas o columnas que contienen valores faltantes, reemplazarlos con valores promedio o interpolados, o ignorarlos completamente.

```
import pandas as pd df = pd.read_csv("datos.csv")

# Eliminar filas con valores faltantes

df.dropna(inplace=True)

 # Reemplazar valores faltantes con el valor promedio

df.fillna(df.mean(), inplace=True)
```

En general, es importante revisar cuidadosamente los datos y especificar correctamente los parámetros de lectura de Pandas para evitar errores al importar los datos.

Mas práctica: https://colab.research.google.com/drive/1e4ZAE9Fr0fHzB-DrGZKsNlse8TwW0Bhv?usp=share_link

3.7. Selección y filtrado de datos en Pandas

La selección y el filtrado de datos son tareas muy comunes en el análisis de datos con Pandas. Aquí hay una introducción a algunas de las herramientas de selección y filtrado de datos en Pandas:

- Selección de columnas: para seleccionar una columna en un DataFrame, puede utilizar la sintaxis de corchetes y pasar el nombre de la columna como una cadena. Por ejemplo, si tenemos un DataFrame llamado df y queremos seleccionar la columna "edad", podemos hacerlo de la siguiente manera: df['edad'].
- Selección de filas: para seleccionar filas en un DataFrame, podemos utilizar el método .loc[] y pasar una etiqueta de fila o una lista de etiquetas de fila. Por ejemplo, si tenemos un DataFrame llamado df con una columna llamada "nombre" y queremos seleccionar todas las filas para las personas llamadas "Juan", podemos hacerlo de la siguiente manera: df.loc[df['nombre'] == 'Juan'].
- Filtrado de datos: para filtrar filas en un DataFrame, podemos utilizar la función .query() y pasar una expresión booleana que evalúe a verdadero o falso para cada fila. Por ejemplo, si tenemos un DataFrame llamado df con una columna llamada "edad" y queremos filtrar todas las filas para las personas mayores de 30 años, podemos hacerlo de la siguiente manera: df.query('edad > 30').
- Selección de columnas y filas: podemos utilizar la función .loc[] para seleccionar tanto filas como columnas en un DataFrame. Por ejemplo, si tenemos un DataFrame llamado df con columnas "nombre", "edad" y "ciudad" y queremos seleccionar las filas para las personas llamadas "Juan" y "Ana" y las columnas "edad" y "ciudad", podemos hacerlo de la siguiente manera: df.loc[df['nombre'].isin(['Juan', 'Ana']), ['edad', 'ciudad']].
- La selección de columnas por posición nos permite seleccionar una o varias columnas del Data Frame por su posición numérica. Por ejemplo, si queremos seleccionar la primera y tercera columna de un DataFrame, podemos hacer lo siguiente

df.iloc[:, [0, 2]]

Este código seleccionará todas las filas del DataFrame df y las columnas de posición 0 y 2 (la primera y tercera columna, respectivamente).

Finalmente, la selección de filas por número de índice se realiza mediante el método iloc. Por ejemplo, si queremos seleccionar las primeras 5 filas de un DataFrame, podemos hacer lo siguiente:

df.iloc[0:5, :]

Este código seleccionará las primeras 5 filas del Data Frame df y todas las columnas.

3.8. Manipulación de datos

La manipulación de datos en Pandas incluye muchas operaciones para agregar, eliminar y cambiar los valores de los datos en los DataFrames. Algunas de las operaciones más comunes son:

- Agregar una nueva columna: se puede agregar una nueva columna a un DataFrame simplemente asignando una lista o un arreglo NumPy a una nueva columna con un nombre específico. Por ejemplo, si tenemos un DataFrame llamado df y queremos agregar una nueva columna llamada "edad", podemos hacerlo de la siguiente manera:

```
df['edad'] = [22, 35, 27, 19]
```

- Eliminar una columna: se puede eliminar una columna del DataFrame usando el método drop() con el parámetro axis=1. Por ejemplo, si queremos eliminar la columna "edad" del DataFrame df, podemos hacerlo así:

```
df = df.drop('edad', axis=1)
```

- Cambiar el nombre de las columnas: se puede cambiar el nombre de las columnas del DataFrame usando el método rename(). Por ejemplo, si queremos cambiar el nombre de la columna "edad" a "años", podemos hacerlo así:

```
df = df.rename(columns={'edad': 'años'})
```

- Agregar una nueva fila: se puede agregar una nueva fila a un DataFrame usando el método append(). Por ejemplo, si tenemos un DataFrame df y queremos agregar una nueva fila con los valores [5, 'Juan', 'Pérez'], podemos hacerlo así:

```
df = df.append(pd.Series([5, 'Juan', 'Pérez'], index=df.columns), ignore_index=True)
```

- Eliminar una fila: se puede eliminar una fila del DataFrame usando el método drop() con el parámetro axis=0. Por ejemplo, si queremos eliminar la primera fila del DataFrame df, podemos hacerlo así:

```
df = df.drop(0, axis=0)
```

3.9. Combinación de datos

Por ejemplo, supongamos que tenemos dos DataFrames que contienen información de ventas de diferentes regiones. El primer DataFrame contiene información de ventas de la región "A" y el segundo DataFrame contiene información de ventas de la región "B". Ambos DataFrames tienen una columna en común llamada "Fecha" que contiene la fecha de la venta.

Merge

Para combinar estos dos conjuntos de datos en un solo DataFrame, podemos usar la función "merge" de Pandas. Por ejemplo:

```
df_a = pd.read_csv("ventas_region_a.csv")
df_b = pd.read_csv("ventas_region_b.csv")
df_merge = pd.merge(df_a, df_b, on="Fecha")
```

Esto creará un nuevo DataFrame llamado "df_merge" que contiene todas las filas de los DataFrames "df_a" y "df_b" donde las fechas coinciden.

Concat

También podemos combinar DataFrames utilizando la función "concat" de Pandas.

Por ejemplo, si queremos agregar las ventas de la región "C" a nuestro DataFrame combinado, podemos hacer lo siguiente:

```
df_c = pd.read_csv("ventas_region_c.csv")
df_merge = pd.concat([df_merge, df_c], ignore_index=True)
```

Esto agregará todas las filas del DataFrame "df_c" a nuestro DataFrame combinado "df_merge", y establece el índice para que sea secuencial (ignorando los índices de los DataFrames originales).

Join

El join es una operación de combinación de datos que se utiliza para unir dos DataFrames a lo largo de una o varias columnas comunes (llamadas claves). La sintaxis básica del join es la siguiente:

```
df1.join(df2, on='key')
```

donde df1 y df2 son los DataFrames que se desean unir y key es la columna común que se utilizará como clave para la unión.

Por defecto, el join se realiza mediante la operación de unión interna (inner join), lo que significa que sólo se conservarán las filas que tengan valores coincidentes en ambas tablas. Sin embargo, también es

posible especificar otros tipos de unión, como la unión izquierda (left join), la unión derecha (right join) y la unión externa completa (full outer join), utilizando el parámetro how.

Por ejemplo, supongamos que tenemos dos DataFrames ventas y productos que contienen información sobre las ventas y los productos de una tienda, respectivamente, y queremos unirlos por el campo producto_id. Podemos hacerlo de la siguiente manera:

```
ventas_productos = ventas.join(productos, on='producto_id')
```

Esto nos devolverá un nuevo Data Frame ventas_productos que contiene toda la información de las ventas y los productos unidos por el campo producto_id.

3.10. Análisis de datos: Cómo realizar cálculos estadísticos y resúmenes de datos

En Pandas, es posible realizar cálculos estadísticos y resúmenes de datos de manera rápida y sencilla. Algunos ejemplos de funciones que pueden utilizarse para este propósito son:

- describe(): devuelve un resumen estadístico de las columnas numéricas del Data Frame, incluyendo la media, la desviación estándar, el valor mínimo y máximo, entre otros.
- mean(): devuelve la media de los valores de una columna.
- median(): devuelve la mediana de los valores de una columna.
- std(): devuelve la desviación estándar de los valores de una columna.
- var(): devuelve la varianza de los valores de una columna.
- count(): cuenta el número de valores no nulos en una columna.
- unique(): devuelve los valores únicos de una columna.

Estas son solo algunas de las funciones disponibles en Pandas para el análisis de datos. También es posible utilizar funciones de agregación, como groupby(), para agrupar los datos por una o varias columnas y realizar cálculos estadísticos en cada grupo.

3.11. Manipulación de nulos

La manipulación de datos nulos es una tarea común en el análisis de datos, ya que los datos faltantes o nulos pueden afectar negativamente la precisión de los análisis. Pandas proporciona varias funciones para manejar datos nulos.

- Para identificar y contar los valores nulos en un DataFrame, podemos usar la función isnull() y sum(). Por ejemplo:

```
import pandas as pd

df = pd.read_csv("data.csv")
print(df.isnull().sum())
```

Esto devolverá el número de valores nulos en cada columna del DataFrame df.

- Para eliminar los valores nulos, podemos usar la función dropna(). Por ejemplo:

```
import pandas as pd

df = pd.read_csv("data.csv")
df = df.dropna()
```

Esto eliminará todas las filas que contengan al menos un valor nulo en el DataFrame df.

- También podemos reemplazar los valores nulos con un valor específico utilizando la función fillna(). Por ejemplo:

```
import pandas as pd

df = pd.read_csv("data.csv")
df = df.fillna(0)
```

Esto imputará todos los valores nulos con ceros.

La imputación implica reemplazar los valores nulos con valores estimados o calculados. Existen varias técnicas de imputación, como la imputación de media, mediana, moda, interpolación, entre otras. Es importante elegir la técnica de imputación adecuada según el tipo de datos y el problema que se esté abordando. Además, es importante tener en cuenta que la imputación puede afectar la distribución de los datos y, por lo tanto, debe ser utilizada con precaución y validada adecuadamente. También es importante documentar cualquier imputación realizada y tener en cuenta el impacto en los resultados finales del análisis de datos.

Mas información: Working with Missing Data in Pandas - GeeksforGeeks

Podemos aplicar estas tecnicas mencionadas:

- Valor constante o cero
- Valores estadisticos

O podriamos optar por aplicarlo a traves de una libreria que veremos más adelante llamada Scikit-learn (también conocido como sklearn).

sklearn.impute.SimpleImputer — scikit-learn 1.2.2 documentation

3.12. Introducción a librerías de visualización: Matplotlib y Seaborn

Matplotlib y Seaborn son dos librerías de visualización de datos muy utilizadas en Python para crear gráficos y visualizaciones de alta calidad.

Matplotlib es una librería base de visualización en Python que permite crear gráficos básicos como líneas, barras, histogramas y más. Es muy flexible y permite personalizar cada aspecto de los gráficos, aunque puede ser un poco tedioso en la creación de gráficos más complejos.

Por otro lado, Seaborn es una librería de visualización de datos construida sobre Matplotlib, que ofrece una interfaz más sencilla para crear gráficos más complejos y atractivos visualmente. Seaborn es particularmente útil para crear gráficos de distribución, diagramas de violín, mapas de calor y más.

Ambas librerías son muy utilizadas en análisis de datos, ciencia de datos y machine learning para crear visualizaciones de datos que permitan entender mejor los patrones y relaciones en los datos.

Además de estas dos librerías, existen otras opciones como Plotly, Bokeh, Altair y más, cada una con sus propias fortalezas y debilidades en términos de facilidad de uso, flexibilidad y tipos de gráficos que pueden crear.

Matplotlib

- Es una librería de visualización de datos en 2D y 3D para Python.
- Es una de las herramientas más utilizadas para la visualización de datos en Python.
- Puede crear gráficos estáticos, animaciones, diagramas de contorno, entre otros.
- Ofrece una gran cantidad de opciones de personalización para los gráficos, lo que permite crear visualizaciones de alta calidad y adaptadas a las necesidades de cada proyecto.

Ejemplos

Tipo de gráfico	Descripción	Código de ejemplo
Gráfico de línea	Muestra una representación visual de la relación entre dos variables continuas a lo largo de un eje	Código de ejemplo
Gráfico de dispersión	Muestra la relación entre dos variables continuas en un plano cartesiano	Código de ejemplo
Gráfico de barras	Muestra la comparación de valores entre diferentes categorías usando barras rectangulares	Código de ejemplo
Histograma	Muestra la distribución de una variable continua en forma de barras verticales	Código de ejemplo
Gráfico de torta	Muestra la proporción de diferentes categorías en forma de un círculo dividido en segmentos	Código de ejemplo

Seaborn

- Es una librería de visualización de datos para Python basada en Matplotlib.
- Se enfoca en la creación de gráficos estadísticos atractivos e informativos.

- Ofrece una interfaz más sencilla y de alto nivel para crear visualizaciones complejas de manera más rápida.
- Tiene una gran cantidad de herramientas para la visualización de datos univariados, bivariados y multivariados.
- Además de las herramientas básicas, Seaborn también incluye una serie de gráficos especializados para la exploración de datos estadísticos, como gráficos de violín, boxplot y diagramas de dispersión con ajustes de regresión.

Ejemplos

Tipo de Gráfico	Descripción	Ejemplo de Código
Gráfico de Distribución (histograma, kde, etc.)	Visualiza la distribución de datos, como histogramas o gráficos de densidad.	seaborn.histplot(data=df, x='columna')
Gráfico de Caja (boxplot)	Muestra la distribución y la variabilidad de datos en diferentes categorías.	seaborn.boxplot(data=df, x='columna_x', y='columna_y')
Gráfico de Violín	Muestra la distribución de datos en diferentes categorías, con detalles adicionales de densidad.	seaborn.violinplot(data=df, x='columna_x', y='columna_y')
Gráfico de Regresión (lmplot)	Visualiza relaciones lineales entre dos variables, con posibilidad de ajuste de regresión.	seaborn.lmplot(data=df, x='columna_x', y='columna_y')
Gráfico de Matriz de Correlación (heatmap, pairplot, etc.)	Visualiza la correlación entre varias variables en una sola visualización.	seaborn.heatmap(data=df.corr()) seaborn.pairplot(data=df)

Matplotlib y Seaborn son dos librerías de visualización de datos en Python que ofrecen diferentes características y enfoques para la creación de gráficos. Aquí hay algunas diferencias principales entre Matplotlib y Seaborn:

1. Estilo visual: Matplotlib proporciona una gran cantidad de opciones de personalización y permite un control detallado sobre los detalles visuales de los gráficos, lo que significa que puedes crear gráficos altamente personalizados y adaptarlos a tus necesidades específicas. Por otro lado, Seaborn ofrece estilos de gráficos predefinidos que son más atractivos visualmente y generalmente requieren menos configuración para obtener resultados visualmente agradables.
2. Niveles de abstracción: Seaborn ofrece un nivel de abstracción más alto en comparación con Matplotlib, lo que significa que puedes crear gráficos más complejos con menos código. Seaborn también proporciona funciones de alto nivel para la visualización de datos estadísticos, como gráficos de regresión, gráficos de violín y gráficos de distribución conjunta, que no están disponibles en Matplotlib.
3. Integración con Pandas: Seaborn está diseñado para funcionar de manera nativa con la librería de manipulación de datos en Python, Pandas. Esto significa que Seaborn tiene una integración más

fluida con los DataFrames de Pandas, lo que permite una visualización de datos más fácil y rápida para análisis de datos.

4. Paleta de colores: Seaborn ofrece paletas de colores predefinidas que son estéticamente agradables y optimizadas para la visualización de datos, lo que facilita la elección de esquemas de colores atractivos para tus gráficos. Matplotlib, por otro lado, requiere una configuración manual de colores.
5. Facilidad de uso: Seaborn es conocido por su sintaxis simple y concisa, lo que hace que sea más fácil de usar y configurar gráficos rápidamente. Matplotlib, por otro lado, puede requerir más código y configuración para obtener los mismos resultados.

En resumen, Matplotlib es una librería de visualización de datos altamente personalizable y de nivel bajo, mientras que Seaborn es una librería de visualización de datos de nivel más alto, que ofrece estilos predefinidos, integración con Pandas y funciones estadísticas adicionales. La elección entre Matplotlib y Seaborn dependerá de tus necesidades específicas y preferencias de estilo de visualización.

Aquí está el enlace a la galería de ejemplos de Seaborn, donde encontrarás una gran cantidad de ejemplos de código para crear diferentes tipos de gráficos con Seaborn:

https://seaborn.pydata.org/examples/index.html

En esta galería, encontrarás ejemplos de código para gráficos de distribución, gráficos de regresión, gráficos de categoría, gráficos de matriz de correlación y muchos más. Cada ejemplo incluye el código necesario para crear el gráfico, junto con una breve descripción del propósito del gráfico y cómo se puede personalizar.

Esta galería es una excelente fuente de referencia para aprender cómo utilizar Seaborn y crear visualizaciones atractivas y efectivas con esta librería. Te recomiendo revisar los ejemplos de código en la galería para obtener ideas y aprender más sobre las capacidades de Seaborn en la visualización de datos.

4. Data Science con Python: Introducción a las Funcionalidades y Librerías Clave para el Modelado de Datos

El modelado de datos es una parte esencial en los proyectos de Data Science, ya que nos permite construir modelos y realizar predicciones basadas en los datos disponibles. En Python, existen diversas funcionalidades y librerías clave que nos ayudan en este proceso. Algunas de las librerías más populares y ampliamente utilizadas para el modelado de datos en Python son Scikit-Learn, TensorFlow y Keras.

- Scikit-Learn es una librería de aprendizaje automático (machine learning) que proporciona una amplia gama de algoritmos para la clasificación, regresión, clustering, y más. También incluye herramientas para la selección de características, validación de modelos y evaluación de su rendimiento.
- TensorFlow es una librería de código abierto desarrollada por Google que se utiliza principalmente para construir y entrenar modelos de aprendizaje automático y redes neuronales. Es especialmente popular para aplicaciones de Deep Learning y es ampliamente utilizado en la comunidad de Data Science.
- Keras es una interfaz de alto nivel para la construcción de modelos de aprendizaje automático y redes neuronales. Está diseñado para ser fácil de usar y ofrece una gran flexibilidad en la construcción de modelos, lo que lo hace una opción popular para aquellos que se están iniciando en el modelado de datos.

Además de estas librerías, también existen otras herramientas y funcionalidades en Python que son útiles para el modelado de datos, como Statsmodels para realizar análisis de estadísticas y modelado de datos, y XGBoost para el uso de algoritmos de Gradient Boosting, entre otros.

A continuación explicaremos las funcionalidades de Scikit-Learn y Tensor Flow, aprenderemos cómo utilizar sus diferentes algoritmos y herramientas para construir y entrenar modelos de aprendizaje automático y redes neuronales, y entenderemos cómo aplicarlos en casos reales de Data Science. Estas dos librerías son fundamentales en el arsenal de herramientas de cualquier Data Scientist en Python, y te brindarán una sólida base para el modelado de datos en tus proyectos de Data Science.

Introducción a librerías de modelado: Scikit-Learn y Tensor Flow

Scikit-Learn, también conocido como sklearn, es una de las bibliotecas de código abierto más populares en Python para el aprendizaje automático (machine learning). Proporciona una amplia gama de herramientas y algoritmos para realizar tareas de aprendizaje supervisado y no supervisado, así como herramientas para evaluación de modelos, selección de características, preprocesamiento de datos y optimización de modelos.

Scikit-Learn se basa en NumPy, otra popular biblioteca de Python para la computación numérica, y ofrece una interfaz consistente y sencilla de usar para desarrollar modelos de machine learning. Algunos de los algoritmos de machine learning que se encuentran en Scikit-Learn incluyen regresión lineal, regresión logística, clasificación, clustering, reducción de dimensionalidad, y más.

Una de las principales ventajas de Scikit-Learn es su amplia documentación y su comunidad activa, lo que hace que sea una herramienta ampliamente utilizada en la comunidad de data science. Además,

Scikit-Learn se integra fácilmente con otras bibliotecas populares de Python como Pandas para el manejo de datos y Matplotlib para la visualización de resultados.

Algunos de los principales beneficios de Scikit-Learn son:

1. Amplia gama de algoritmos: Scikit-Learn ofrece una amplia variedad de algoritmos de machine learning para tareas de clasificación, regresión, clustering, reducción de dimensionalidad, entre otros. Esto permite a los data scientists seleccionar el algoritmo adecuado para su problema específico.
2. Interfaz consistente: Scikit-Learn ofrece una interfaz coherente y fácil de usar para desarrollar y evaluar modelos de machine learning, lo que facilita el desarrollo y experimentación con diferentes modelos.
3. Preprocesamiento de datos integrado: Scikit-Learn ofrece herramientas integradas para el preprocesamiento de datos, como la manipulación de datos faltantes, la codificación de variables categóricas y la normalización de datos, lo que simplifica el flujo de trabajo de data preprocessing.
4. Evaluación de modelos: Scikit-Learn proporciona herramientas para la evaluación de modelos, como métricas de evaluación y validación cruzada, lo que permite a los data scientists evaluar y comparar el rendimiento de diferentes modelos.
5. Integración con otras bibliotecas de Python: Scikit-Learn se integra fácilmente con otras bibliotecas populares de Python, como NumPy, Pandas y Matplotlib, lo que permite a los data scientists utilizar herramientas complementarias para el manejo de datos y la visualización de resultados.

Para utilizar Scikit-Learn en Google Colab, te recomiendo los siguientes pasos:

Paso 1: Importar la biblioteca

En una celda de código en Google Colab, puedes importar la biblioteca scikit-learn usando el siguiente código:

```
from sklearn import *
```

Paso 2: Utilizar la biblioteca

Una vez que hayas importado la biblioteca scikit-learn en tu entorno de Google Colab, puedes usar sus clases y funciones para implementar algoritmos de aprendizaje automático. Por ejemplo, puedes crear un modelo de regresión lineal de la siguiente manera:

```
from sklearn.linear_model import LinearRegression

# Crear un objeto de modelo de regresión lineal
modelo = LinearRegression()

# Entrenar el modelo con datos de entrenamiento
modelo.fit(X_entrenamiento, y_entrenamiento)

# Hacer predicciones con el modelo entrenado
y_prediccion = modelo.predict(X_prueba)
```

Ejemplo en Google Colab

Aquí te proporciono algunos enlaces útiles para investigar más sobre cómo utilizar scikit-learn en Google Colab:

- Documentación oficial de scikit-learn: La documentación oficial de scikit-learn es una excelente fuente de información para aprender sobre cómo utilizar esta biblioteca de aprendizaje automático. Puedes encontrar la documentación en el siguiente enlace: https://scikit-learn.org/stable/
- IPython Cookbook - 8.1. Getting started with scikit-learn (ipython-books.github.io)
- | notebook.community

¿Y Tensor Flow?

TensorFlow es un marco de trabajo de código abierto para aprendizaje automático desarrollado por Google que proporciona herramientas y librerías para construir y entrenar diversos tipos de modelos de aprendizaje automático. Es ampliamente utilizado en una amplia gama de aplicaciones, incluyendo reconocimiento de imágenes y voz, procesamiento del lenguaje natural y sistemas de recomendación.

TensorFlow se basa en el concepto de tensores, que son arreglos multidimensionales utilizados para representar datos. Permite a los desarrolladores definir cálculos matemáticos como gráficos computacionales, donde los nodos representan operaciones y las aristas representan el flujo de datos.

Ejemplo en Google Colab

Para aprender todo sobre Tensor Flow → TensorFlow

5. Conclusiones

En esta unidad, hemos destacado la importancia de aprender Python como lenguaje de programación en la ciencia de datos, debido a su simplicidad, flexibilidad y amplia comunidad de desarrolladores, lo que lo convierte en una herramienta poderosa en esta disciplina. Además, hemos explorado cómo Python es ampliamente utilizado en la comunidad de desarrollo e IT, lo que lo hace aún más relevante en el campo de la ciencia de datos a nivel universitario.

Hemos resaltado la necesidad de utilizar un Entorno de Desarrollo Integrado (IDE) eficiente para programar en Python, y en este curso hemos utilizado la poderosa herramienta Google Colab. También hemos aprendido sobre las librerías clave de Python, como NumPy, Pandas, Matplotlib y Seaborn, que son esenciales para el procesamiento y análisis de datos en la ciencia de datos.En el contexto específico del modelado de datos, hemos descubierto que Python ofrece librerías poderosas como Scikit-learn, Statsmodels y TensorFlow, que son ampliamente utilizadas en la comunidad de data science para el desarrollo y entrenamiento de modelos de machine learning y análisis estadístico de datos. Estas librerías proporcionan herramientas y algoritmos avanzados para la construcción de modelos predictivos, clasificación, regresión, clustering y más, lo que nos permite realizar tareas de modelado de datos de manera eficaz y obtener resultados precisos.

En el próximo paso de nuestro aprendizaje, aplicaremos estos conceptos y técnicas en ejercicios prácticos para consolidar y aplicar de manera efectiva los conocimientos adquiridos. Esto nos preparará para enfrentar proyectos reales en la ciencia de datos, donde podremos aplicar todas estas herramientas y técnicas para resolver problemas y obtener insights valiosos a partir de datos reales.

6. Bibliografía utilizada y sugerida

Aquí te recomiendo un libro y enlaces a la documentación oficial de Python y algunas de las librerías mencionadas durante la Unidad N°2:

Si buscas un libro gratuito y ampliamente utilizado para aprender sobre ciencia de datos con Python:

1. Vanderplas, J. T., & VanderPlas, J. (2016). Python data science handbook: Essential tools for working with data. O'Reilly Media.

Enlaces a la documentación oficial de Python y sus librerías:

1. Documentación oficial de Matplotlib. (n.d.). Retrieved April 19, 2023, from https://matplotlib.org/stable/contents.html
2. Documentación oficial de NumPy. (n.d.). Retrieved April 19, 2023, from https://numpy.org/doc/
3. Documentación oficial de Pandas. (n.d.). Retrieved April 19, 2023, from https://pandas.pydata.org/pandas-docs/stable/
4. Documentación oficial de Python. (n.d.). Retrieved April 19, 2023, from https://docs.python.org/3/
5. Documentación oficial de Scikit-learn. (n.d.). Retrieved April 19, 2023, from https://scikit-learn.org/stable/
6. Documentación oficial de Seaborn. (n.d.). Retrieved April 19, 2023, from https://seaborn.pydata.org/
7. The python tutorial. (n.d.). Python Documentation. Retrieved April 19, 2023, from https://docs.python.org/3/tutorial/index.html