SENATI

# Módulos y Paquetes para Machine Learning con Python

## MATERIAL TÉCNICO DE APOYO

**TAREA N°01**

**Realiza operaciones con las Librerías Pandas y Numpy.**

**Fundamentos de manipulación de datos con Pandas y Numpy.**

- **Fundamentos de vectores y matrices.**

  - Vectores
    - Definición:

      Un vector es una cantidad que tiene tanto magnitud como dirección. En el contexto de álgebra lineal y machine learning, un vector es una lista de números ordenados que pueden ser representados en el espacio n-dimensional.

    - Representación:

      Un vector puede ser representado de forma vertical (vector columna) o horizontal (vector fila). Por ejemplo, en un espacio tridimensional:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

  - Operaciones Básicas:
    - Suma de Vectores:

$$A + B = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ a_3 + b_3 \end{bmatrix}$$

    - Resta de Vectores:

$$A - B = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} - \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_1 - b_1 \\ a_2 - b_2 \\ a_3 - b_3 \end{bmatrix}$$

    - Multiplicación por un Escalar:

$$c.a = c.\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} c.a_1 \\ c.a_2 \\ c.a_3 \end{bmatrix}$$

- Producto Punto (Escalar):

$$a \,.\, b = a_1.b_1 + a_2.b_2 + a_3.b_3$$

- Matrices
  - Definición:

    Una matriz es una tabla bidimensional de números organizada en filas y columnas. Las matrices se usan para representar y manipular datos en machine learning y álgebra lineal.

  - Representación:

    Una matriz de m filas y n columnas se denota como A:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

  - Operaciones Básicas:
    - Suma de Matrices:

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{bmatrix}$$

    - Resta de Matrices:

$$\mathbf{A} - \mathbf{B} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} - \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} - b_{11} & a_{12} - b_{12} \\ a_{21} - b_{21} & a_{22} - b_{22} \end{bmatrix}$$

    - Multiplicación por un Escalar:

$$c \cdot \mathbf{A} = c \cdot \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} c \cdot a_{11} & c \cdot a_{12} \\ c \cdot a_{21} & c \cdot a_{22} \end{bmatrix}$$

    - Multiplicación de Matrices:

$$\mathbf{A} \cdot \mathbf{B} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{bmatrix}$$

- Aplicación en Machine Learning.

  Vectores y matrices son fundamentales en machine learning para:

  - Representar datasets: Filas como ejemplos de datos y columnas como características.
  - Operaciones de transformación: Transformar datos y aplicar operaciones matemáticas.
  - Redes neuronales: Pesos y biases se representan y calculan mediante matrices y vectores.
  - Algoritmos de optimización: Gradientes y funciones de pérdida se manejan usando álgebra lineal.

- Ejemplos en Python

  Usando NumPy, una biblioteca fundamental en Python para operaciones con vectores y matrices.

```
python
import numpy as np

# Crear vectores
vector_a = np.array([1, 2, 3])
vector_b = np.array([4, 5, 6])

# Suma de vectores
vector_sum = vector_a + vector_b

# Producto escalar
dot_product = np.dot(vector_a, vector_b)

# Crear matrices
matrix_a = np.array([[1, 2], [3, 4]])
matrix_b = np.array([[5, 6], [7, 8]])

# Suma de matrices
matrix_sum = matrix_a + matrix_b

# Multiplicación de matrices
matrix_product = np.dot(matrix_a, matrix_b)
```

```
print("Vector Sum:", vector_sum)
print("Dot Product:", dot_product)
print("Matrix Sum:\n", matrix_sum)
print("Matrix Product:\n", matrix_product)
```

Este código muestra cómo realizar operaciones básicas con vectores y matrices en Python usando NumPy.

- **Definición de Pandas y Numpy.**

Pandas es una biblioteca de código abierto de Python ampliamente utilizada en el análisis de datos y machine learning. Proporciona estructuras de datos eficientes y flexibles para manejar datos numéricos y de otro tipo en Python. Los principales componentes de Pandas son los DataFrame y las Series. Un DataFrame es una tabla de datos bidimensional con etiquetas de fila y columna, mientras que una Series es un arreglo unidimensional de datos etiquetados.

Pandas permite manipular y limpiar datos provenientes de diversas fuentes, como hojas de cálculo, archivos CSV, bases de datos SQL y formatos de datos populares en la web. También ofrece herramientas para el análisis de datos, que incluyen agregación, filtrado y transformación de datos, además de la creación de gráficos y visualizaciones para explorar patrones y tendencias.

Gracias a su integración con otras bibliotecas de Python utilizadas en el análisis de datos y machine learning, como NumPy y Matplotlib, Pandas se convierte en una herramienta esencial para cualquier científico de datos o analista que trabaje con datos en Python.

Pandas es una biblioteca de código abierto de Python utilizada para el análisis y manipulación de datos. Ofrece estructuras de datos eficientes y herramientas para trabajar con datos numéricos y de otro tipo en Python. Los dos principales objetos de Pandas son:

- DataFrame: Una tabla bidimensional con datos, que incluye etiquetas de fila y columna. Es ideal para representar y manejar datos tabulares.
- Series: Un arreglo unidimensional de datos etiquetados, que puede considerarse como una columna en un DataFrame.

Pandas permite importar datos desde diversas fuentes como archivos CSV, hojas de cálculo, bases de datos SQL y otros formatos de datos populares en la web. Además, proporciona funcionalidades para la limpieza, agregación, filtrado y

transformación de datos, así como herramientas para la visualización de datos y la exploración de patrones y tendencias.

**NumPy:**

NumPy, abreviatura de "Numerical Python", es una biblioteca de código abierto fundamental para la computación científica en Python. Proporciona soporte para matrices y arreglos multidimensionales, junto con una colección de funciones matemáticas para realizar operaciones sobre estos arrays de manera eficiente.

Las principales características de NumPy incluyen:

- Arrays N-dimensionales: Arrays eficientes y de tamaño fijo que permiten realizar operaciones matemáticas y lógicas.
- Funciones Matemáticas: Herramientas para realizar operaciones matemáticas avanzadas como álgebra lineal, transformadas de Fourier y generación de números aleatorios.
- Integración con Otras Bibliotecas: Se integra perfectamente con otras bibliotecas científicas en Python como Pandas, SciPy y Matplotlib.

NumPy es esencial para la mayoría de las tareas de computación científica en Python, y forma la base sobre la cual se construyen muchas otras bibliotecas de análisis de datos y machine learning.

- **Manipulación y análisis de estructura de datos.**

**Pandas:**

- Manipulación de Estructuras de Datos:

  Creación de DataFrames y Series:
  - Series: pd.Series(data, index=index)
  - DataFrame: pd.DataFrame(data, columns=columns, index=index)

- Selección y Filtrado de Datos:
  - Selección por columna: df['column_name']
  - Selección por fila: df.loc['row_label'] o df.iloc[row_index]
  - Filtrado por condición: df[df['column_name'] > value]

- Operaciones de Limpieza de Datos:
  - Manejo de valores nulos: df.dropna(), df.fillna(value)
  - Cambio de tipo de datos: df['column_name'].astype(new_type)
  - Eliminación de duplicados: df.drop_duplicates()

- Transformación de Datos:
  - Aplicación de funciones: df.apply(function)
  - Renombrado de columnas: df.rename(columns={'old_name': 'new_name'})
  - Creación de nuevas columnas: df['new_column'] = df['existing_column'].apply(transformation_function)

**Análisis de Estructuras de Datos:**

- Estadísticas Descriptivas:
  - Resumen estadístico: df.describe()
  - Media, mediana y moda: df.mean()`, `df.median()`, `df.mode()
  - Conteos: df['column_name'].value_counts()

- Agrupación y Agregación de Datos:
  - Agrupación: `df.groupby('column_name')`
  - Agregación: df.groupby('column_name').agg({'column1': 'mean', 'column2': 'sum'})
  - Pivot Tables: df.pivot_table(values='column_name', index='index_column', columns='column_name', aggfunc='mean')

- Unión y Combinación de Datos:*
  - Concatenación: pd.concat([df1, df2])
  - Merge: pd.merge(df1, df2, on='common_column')
  - Join: `df1.join(df2, on='common_column')

**NumPy:**

**Manipulación de Arrays:**

- Creación de Arrays:
  - Array unidimensional: np.array([1, 2, 3])
  - Array bidimensional: np.array([[1, 2], [3, 4]])
  - Arrays con valores inicializados: np.zeros((3, 3))`, `np.ones((3, 3))`, `np.arange(10)

- Indexación y Slicing:
  - Selección de elementos: array[0, 1]
  - Slicing: array[:, 1], array[1:3]

- Operaciones Básicas:
  - Operaciones element-wise: array1 + array2`, `array1 * array2
  - Operaciones matemáticas: np.sum(array), np.mean(array), `np.std(array)`

**Análisis de Arrays:**

- Transformación de Arrays:
  - Reshape: `array.reshape((rows, columns))`
  - Transpose: `array.T`
  - Flatten: `array.flatten()`

- Operaciones de Álgebra Lineal:
  - Producto punto: np.dot(array1, array2)
  - Determinante de una matriz: np.linalg.det(matrix)
  - Inversa de una matriz: np.linalg.inv(matrix)

- Funciones Universales (ufuncs):
  - Funciones matemáticas: `np.sin(array)`, `np.exp(array)
  - Comparaciones: np.maximum(array1, array2)

**Ejemplo en Python Usando Pandas y NumPy:**

```
import pandas as pd
import numpy as np

# Creación de un DataFrame en Pandas
data = {
   'A': [1, 2, 3, 4],
   'B': [5, 6, 7, 8]
}
df = pd.DataFrame(data)

# Selección y Filtrado
filtered_df = df[df['A'] > 2]

# Estadísticas Descriptivas
mean_value = df['A'].mean()

# Operaciones con NumPy
array = np.array([[1, 2], [3, 4]])
```

```
# Producto punto
dot_product = np.dot(array, array)

# Transformación de Arrays
reshaped_array = array.reshape((4, 1))

print("Filtered DataFrame:")
print(filtered_df)
print("Mean Value of Column A:", mean_value)
print("Dot Product of Array:")
print(dot_product)
print("Reshaped Array:")
print(reshaped_array)
```

Este ejemplo ilustra cómo crear y manipular un DataFrame en Pandas, realizar operaciones estadísticas, y cómo usar arrays en NumPy para realizar operaciones matemáticas y transformaciones de arrays.

- **Lectura de archivos CSV con Numpy y Pandas.**

Pandas proporciona una manera sencilla y eficiente de leer archivos CSV. A continuación, se muestra un ejemplo de cómo hacerlo:

```
import pandas as pd

# Leer el archivo CSV en un DataFrame de Pandas
df = pd.read_csv('ruta/al/archivo.csv')

# Mostrar las primeras 5 filas del DataFrame
print(df.head())
```

- Detalles Adicionales:

- Especificar Separador: Si el archivo CSV usa un separador diferente al de coma (`,`), puedes especificarlo usando el parámetro `sep`.

```
df = pd.read_csv('ruta/al/archivo.csv', sep=';')
```

- Manejo de Valores Nulos: Puedes especificar cómo manejar los valores nulos en el CSV.
  df = pd.read_csv('ruta/al/archivo.csv', na_values=['NA', 'N/A', 'null'])

- Especificar Columnas: Puedes leer solo un subconjunto de columnas del archivo CSV.
  df = pd.read_csv('ruta/al/archivo.csv', usecols=['columna1', 'columna2'])

**Lectura de Archivos CSV con NumPy.**

NumPy también permite leer archivos CSV, aunque es más adecuado para datos numéricos simples. Aquí hay un ejemplo de cómo hacerlo:

```
import numpy as np

# Leer el archivo CSV en un array de NumPy
data = np.genfromtxt('ruta/al/archivo.csv', delimiter=',', skip_header=1)

# Mostrar los datos leídos
print(data)
```

**Detalles Adicionales:**

- Especificar Tipo de Datos: Puedes especificar el tipo de datos de los elementos del array.
  data = np.genfromtxt('ruta/al/archivo.csv', delimiter=',', dtype=float)

- Manejo de Valores Faltantes: Puedes especificar cómo manejar los valores faltantes.
  data = np.genfromtxt('ruta/al/archivo.csv', delimiter=',', filling_values=0)

- Leer con Columnas Heterogéneas: Para leer columnas con diferentes tipos de datos, puedes usar el parámetro `dtype`.
  data = np.genfromtxt('ruta/al/archivo.csv', delimiter=',', dtype=None, names=True, encoding=None)

**Ejemplo Completo:**

```
import pandas as pd
import numpy as np

# Usando Pandas para leer el CSV
df = pd.read_csv('ruta/al/archivo.csv')
print("DataFrame con Pandas:")
print(df.head())

# Usando NumPy para leer el CSV
data = np.genfromtxt('ruta/al/archivo.csv', delimiter=',', skip_header=1)
print("Array con NumPy:")
print(data)
```

Este ejemplo muestra cómo leer un archivo CSV usando tanto Pandas como NumPy, permitiendo comparar las salidas y elegir la herramienta adecuada según las necesidades del análisis de datos.

**TAREA N°02**

**Estudia el uso de las Librerías Scikit-learn y Pytorch.**

- **Fundamentos de Machine Learning con Scikit-Learn y Pytorch.**

  - **Conceptos de Machine Learning.**
    Machine Learning (ML) es un subcampo de la inteligencia artificial que se centra en el desarrollo de algoritmos y técnicas que permiten a las computadoras aprender de los datos y hacer predicciones o decisiones sin ser programadas explícitamente para ello. Aquí se presentan algunos conceptos clave:

    - Tipos de Machine Learning

      - Aprendizaje Supervisado:
        En este tipo de aprendizaje, el modelo es entrenado con datos etiquetados. Es decir, cada entrada de entrenamiento viene con una etiqueta o resultado esperado. El objetivo es que el modelo aprenda a predecir la etiqueta o resultado para nuevos datos no vistos.
        Clasificación: El objetivo es predecir una etiqueta discreta (e.g., "spam" o "no spam").
        Regresión: El objetivo es predecir un valor continuo (e.g., precio de una casa).

      - Aprendizaje No Supervisado:
        Aquí, el modelo es entrenado con datos sin etiquetas. El objetivo es descubrir patrones o estructuras en los datos.
        Clustering: Agrupación de datos en clusters (e.g., segmentación de clientes).
        Reducción de Dimensionalidad: Simplificar los datos reduciendo el número de características (e.g., PCA).

      - Aprendizaje por Refuerzo:
        En este tipo de aprendizaje, un agente aprende a tomar decisiones mediante la interacción con un entorno. Recibe recompensas o castigos en función de sus acciones y el objetivo es maximizar la recompensa acumulada a lo largo del tiempo.

- Conceptos Clave
  - Datos y Conjuntos de Datos:
    Conjunto de Entrenamiento: Datos utilizados para entrenar el modelo.
  - Conjunto de Validación: Datos utilizados para ajustar los hiperparámetros del modelo.
  - Conjunto de Prueba: Datos utilizados para evaluar el rendimiento final del modelo.

- Funciones de Costo y Optimización:
  - Función de Costo: Una función que el modelo trata de minimizar (e.g., error cuadrático medio en regresión).
  - Algoritmos de Optimización: Métodos para minimizar la función de costo (e.g., gradiente descendente).

- Modelos y Algoritmos:
  - Regresión Lineal y Logística: Métodos básicos para regresión y clasificación.
  - Árboles de Decisión y Bosques Aleatorios: Métodos para tomar decisiones basados en estructuras de árbol.
  - Máquinas de Soporte Vectorial (SVM): Algoritmos para clasificación que encuentran el hiperplano que mejor separa las clases.
  - Redes Neuronales y Deep Learning: Modelos inspirados en la estructura del cerebro humano, adecuados para tareas complejas como reconocimiento de imágenes y procesamiento de lenguaje natural.

- Evaluación de Modelos

  - Métricas de Evaluación:
    Exactitud: Proporción de predicciones correctas sobre el total de predicciones.
    Precisión y Recall: Métricas utilizadas en clasificación para evaluar la calidad de las predicciones positivas.
    Matriz de Confusión: Tabla utilizada para describir el rendimiento de un modelo de clasificación.
    Error Cuadrático Medio (MSE): Métrica utilizada en regresión para medir la diferencia entre los valores predichos y los valores reales.

- Validación Cruzada:
  Técnica para evaluar el rendimiento de un modelo dividiendo los datos en múltiples subconjuntos (folds) y asegurando que cada subconjunto sea utilizado tanto para entrenamiento como para prueba.

- Preprocesamiento de Datos
  - Limpieza de Datos:
    Manejo de valores faltantes, eliminación de duplicados y corrección de errores.

  - Transformación de Datos:
    Estandarización y normalización de características, codificación de variables categóricas.

- Reducción de Dimensionalidad:
  Técnicas como Análisis de Componentes Principales (PCA) para reducir el número de características manteniendo la mayor parte de la variabilidad de los datos.

- Hiperparámetros y Tuning

  ✓ Hiperparámetros:
  Parámetros del modelo que no se aprenden directamente de los datos, sino que deben ser establecidos antes del entrenamiento (e.g., tasa de aprendizaje en redes neuronales).

  ✓ Tuning de Hiperparámetros:
  Técnicas como Grid Search y Random Search para encontrar la combinación óptima de hiperparámetros.

Estos conceptos forman la base del machine learning y son esenciales para entender y aplicar técnicas de aprendizaje automático en diversas aplicaciones.


**Figura N° 01:** Scikit-Learn y Python.

- **Definición de Scikit-Learn y Pytorch.**
  - ❖ Scikit-Learn.
    Scikit-Learn es una biblioteca de código abierto para machine learning en Python. Es una herramienta robusta y eficiente para análisis de datos y

modelado predictivo. Scikit-Learn incluye una variedad de algoritmos para tareas de aprendizaje supervisado y no supervisado, y proporciona utilidades para la evaluación y preprocesamiento de datos.

- Características Principales:
  - ✓ Algoritmos de Machine Learning: Incluye algoritmos para clasificación (e.g., SVM, k-NN, Random Forest), regresión (e.g., Linear Regression, Ridge, Lasso), clustering (e.g., k-Means, DBSCAN), y reducción de dimensionalidad (e.g., PCA, t-SNE).
  - ✓ Facilidad de Uso: Tiene una API simple y consistente, diseñada para facilitar la experimentación y el desarrollo rápido de modelos.
  - ✓ Integración con Otras Bibliotecas: Se integra bien con otras bibliotecas de Python como NumPy, SciPy, y Matplotlib, lo que facilita la manipulación y visualización de datos.
  - ✓ Evaluación y Validación: Proporciona herramientas para la validación cruzada, selección de modelos y métricas de evaluación para medir el rendimiento de los modelos.

- Ejemplo de Uso:

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Cargar el dataset
iris = load_iris()
X = iris.data
y = iris.target

# Dividir el dataset en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Crear y entrenar el modelo
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)

# Realizar predicciones
y_pred = model.predict(X_test)

# Evaluar el rendimiento del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

❖ PyTorch

PyTorch es una biblioteca de código abierto para el aprendizaje profundo (deep learning) desarrollada por Facebook's AI Research lab (FAIR). Es conocida por su flexibilidad y facilidad de uso, especialmente para la investigación y el desarrollo de modelos complejos de aprendizaje profundo.

Características Principales:

- ✓ Tensores: Al igual que NumPy, PyTorch utiliza tensores, que son estructuras de datos multidimensionales que permiten operaciones eficientes en GPU.
- ✓ Autograd: PyTorch incluye un sistema de autodiferenciación que facilita el cálculo automático de gradientes para la optimización de modelos.
- ✓ Modelado Dinámico: A diferencia de otras bibliotecas de deep learning, PyTorch permite la construcción de grafos computacionales dinámicos, lo que significa que el grafo de operaciones se construye sobre la marcha, permitiendo mayor flexibilidad.
- ✓ Integración con Python: PyTorch se integra perfectamente con el ecosistema de Python, lo que facilita su uso con otras bibliotecas de Python y la incorporación de código Python en los modelos.

❖ Ejemplo de Uso:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Crear datos de ejemplo
X = torch.tensor([[1.0], [2.0], [3.0], [4.0]])
y = torch.tensor([[2.0], [4.0], [6.0], [8.0]])

# Definir un modelo simple
class SimpleModel(nn.Module):
def __init__(self):
super(SimpleModel, self).__init__()
self.linear = nn.Linear(1, 1)

def forward(self, x):
return self.linear(x)

# Crear el modelo
model = SimpleModel()
```

```
# Definir la función de pérdida y el optimizador
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Entrenar el modelo
for epoch in range(100):
model.train()

# Forward pass
outputs = model(X)
loss = criterion(outputs, y)

# Backward pass y optimización
optimizer.zero_grad()
loss.backward()
optimizer.step()

if (epoch+1) % 10 == 0:
print(f'Epoch [{epoch+1}/100], Loss: {loss.item():.4f}')

# Evaluar el modelo
model.eval()
with torch.no_grad():
predicted = model(X)
print(f'Predicted: {predicted.numpy()}')
```

- ❖ Comparación
  - ✓ Scikit-Learn: Ideal para tareas de machine learning más tradicionales y estructuradas (como clasificación, regresión, clustering), fácil de usar, excelente para prototipado rápido y modelado estadístico.
  - ✓ PyTorch: Diseñado para deep learning y redes neuronales complejas, ofrece flexibilidad y control detallado sobre el entrenamiento de modelos, ideal para investigación y desarrollo de modelos de aprendizaje profundo.

- ❖ Ambas bibliotecas son fundamentales en el ecosistema de Python para el análisis de datos y el desarrollo de modelos de aprendizaje automático, y son ampliamente utilizadas por investigadores y profesionales en la industria.

- ○ **Identificación de principales aplicaciones.**

- ❖ Scikit-Learn

  Scikit-Learn se utiliza principalmente para tareas tradicionales de machine learning que no requieren el manejo de grandes cantidades de datos o la

implementación de redes neuronales complejas. Algunas de las aplicaciones más comunes incluyen:

- ✓ Clasificación: Identificación de categorías en datos. Ejemplos incluyen la clasificación de correos electrónicos como spam o no spam, y la clasificación de imágenes de dígitos manuscritos.
  Ejemplo: Clasificación de correos electrónicos usando un clasificador Naive Bayes.
  Sector: Seguridad informática, servicios de email.
- ✓ Regresión: Predicción de valores continuos. Ejemplos incluyen la predicción de precios de viviendas y el pronóstico de ventas.
  Ejemplo: Predicción de precios de viviendas utilizando regresión lineal.
  Sector: Bienes raíces, ventas y marketing.
- ✓ Clustering: Agrupación de datos no etiquetados. Ejemplos incluyen la segmentación de clientes y la agrupación de documentos similares.
  Ejemplo: Agrupación de clientes basada en sus patrones de compra usando k-means.
  Sector: Marketing, gestión de clientes.
- ✓ Reducción de Dimensionalidad: Simplificación de conjuntos de datos para mejorar la visualización y reducir el costo computacional. Ejemplos incluyen el análisis de componentes principales (PCA).
  Ejemplo: Reducción de la dimensionalidad de datos genéticos para facilitar su visualización.
  Sector: Bioinformática, análisis de datos.
- ✓ Model Selection: Evaluación y selección de modelos mediante validación cruzada y búsqueda de hiperparámetros.
  Ejemplo: Validación cruzada de modelos para seleccionar el mejor clasificador.
  Sector: Investigación y desarrollo en machine learning.

❖ **PyTorch**

PyTorch se utiliza principalmente para tareas de aprendizaje profundo, donde se requiere la implementación y entrenamiento de redes neuronales complejas. Algunas de las aplicaciones más comunes incluyen:

- ✓ Visión por Computadora: Procesamiento y análisis de imágenes y videos. Ejemplos incluyen el reconocimiento facial y la detección de objetos.
  Ejemplo: Detección de objetos en imágenes usando una red neuronal convolucional (CNN).
  Sector: Seguridad, automoción (vehículos autónomos), salud (análisis de imágenes médicas).

- ✓ Procesamiento de Lenguaje Natural (NLP): Análisis y generación de texto. Ejemplos incluyen la traducción automática y la generación de texto.

Ejemplo: Traducción de texto de un idioma a otro utilizando un modelo de Transformer.
Sector: Traducción, atención al cliente (chatbots), marketing (análisis de sentimiento).

- ✓ Generación de Imágenes: Creación de imágenes nuevas y realistas. Ejemplos incluyen redes generativas adversarias (GANs) para generar imágenes de alta resolución.
  Ejemplo: Generación de imágenes realistas de caras humanas usando GANs.
  Sector: Entretenimiento, diseño gráfico.

- ✓ Recomendación de Sistemas: Sugerencias personalizadas basadas en el comportamiento del usuario. Ejemplos incluyen recomendaciones de productos y contenido.
  Ejemplo: Recomendación de películas basadas en el historial de visualización utilizando un modelo de deep learning.
  Sector: E-commerce, plataformas de streaming.

- ✓ Modelado Predictivo en Series Temporales: Predicción de valores futuros en secuencias de datos temporales. Ejemplos incluyen la predicción de precios de acciones y la previsión de la demanda de energía.
  Ejemplo: Predicción de precios de acciones utilizando una red neuronal recurrente (RNN).
  Sector: Finanzas, energía.

- ❖ Comparación de Aplicaciones

  - Scikit-Learn: Ideal para tareas más estructuradas y tradicionales de machine learning, fácil de usar para prototipado rápido y análisis exploratorio de datos.
  - PyTorch: Ideal para tareas de deep learning y redes neuronales complejas, proporcionando flexibilidad y control detallado para la investigación y desarrollo de modelos avanzados.

- ❖ Ambas bibliotecas tienen sus fortalezas en diferentes tipos de aplicaciones y se pueden utilizar en conjunto para aprovechar lo mejor de ambos mundos en el desarrollo de soluciones de machine learning e inteligencia artificial.

- **Definición del aprendizaje supervisado y no supervisado.**
  - ✓ Aprendizaje Supervisado:
    El **aprendizaje supervisado** es un enfoque de machine learning donde se entrena un modelo utilizando un conjunto de datos etiquetado, es decir,

datos que están asociados con etiquetas o respuestas conocidas. En este tipo de aprendizaje, el objetivo es aprender una función que mapee las entradas a las salidas deseadas, basándose en ejemplos de entrenamiento que contienen pares de entrada-salida.

- Características:
  - Utiliza datos etiquetados para entrenar el modelo.
  - El modelo aprende a hacer predicciones basadas en la relación entre las características de entrada y las etiquetas de salida.
  - Las etiquetas proporcionadas en el conjunto de entrenamiento se utilizan para evaluar la precisión del modelo durante el entrenamiento.
  - Ejemplos comunes incluyen clasificación y regresión.

- Ejemplo: Un modelo de clasificación de correo electrónico spam podría entrenarse con un conjunto de datos que contiene correos electrónicos etiquetados como "spam" o "no spam", donde las características son el contenido del correo electrónico y la salida deseada es la etiqueta de clasificación.

✓ **Aprendizaje No Supervisado:**
El **aprendizaje no supervisado** es un enfoque de machine learning donde se entrena un modelo utilizando un conjunto de datos no etiquetado, es decir, datos que no tienen asociadas etiquetas o respuestas conocidas. En este tipo de aprendizaje, el objetivo es explorar la estructura subyacente de los datos para encontrar patrones interesantes o distribuciones en los datos.

- Características:
  - Utiliza datos no etiquetados para encontrar patrones o estructuras ocultas en los datos.
  - El modelo aprende a agrupar o representar los datos sin supervisión explícita de las etiquetas.
  - No se utilizan etiquetas para evaluar el modelo durante el entrenamiento, lo que puede hacer que la evaluación sea más subjetiva.
  - Ejemplos comunes incluyen clustering y reducción de dimensionalidad.

- Ejemplo: Un algoritmo de clustering podría utilizarse para agrupar clientes en diferentes segmentos de mercado basados en sus comportamientos de compra, sin necesidad de etiquetas previas que indiquen las categorías de los clientes.

En resumen, el aprendizaje supervisado se basa en datos etiquetados para aprender relaciones entre características y etiquetas, mientras que el

aprendizaje no supervisado se centra en encontrar patrones o estructuras subyacentes en datos no etiquetados. Ambos enfoques son fundamentales en el campo del machine learning y se utilizan en una variedad de aplicaciones.

- **Definición de redes neuronales.**

Las redes neuronales son modelos computacionales inspirados en la estructura y funcionamiento del cerebro humano, diseñados para aprender patrones complejos y realizar tareas diversas mediante el procesamiento de datos. Estas redes consisten en unidades básicas llamadas neuronas artificiales, organizadas en capas interconectadas que trabajan de manera colaborativa para resolver problemas específicos.

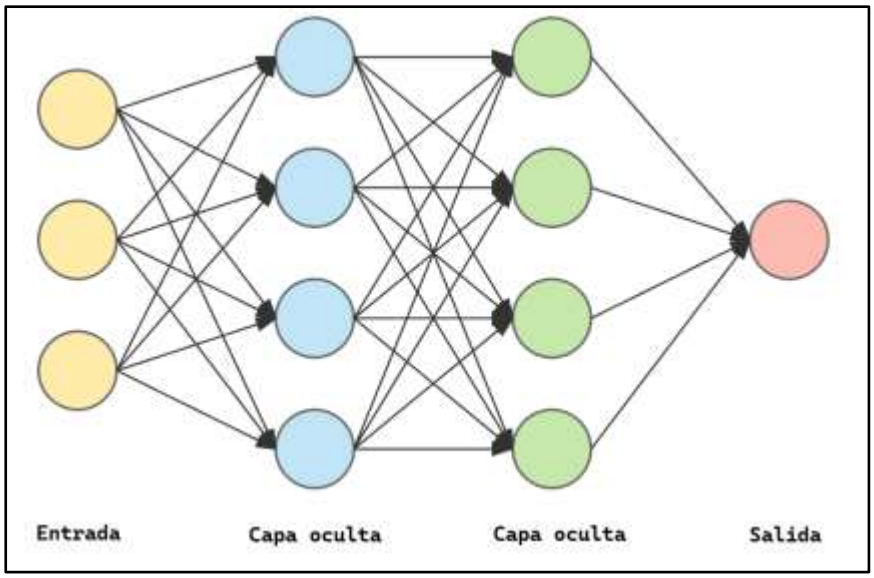


**Figura N° 02:** Redes neuronales.

**Características:**

- ✓ Neuronas Artificiales: Son unidades fundamentales que realizan cálculos simples y emulan el comportamiento de las neuronas biológicas. Cada neurona recibe entradas, realiza una operación matemática en ellas y produce una salida.

- ✓ Capas Ocultas: Son capas intermedias entre la capa de entrada y la capa de salida de la red neuronal. Estas capas permiten que la red aprenda representaciones complejas y abstractas de los datos de entrada.

- ✓ Conexiones Ponderadas: Cada conexión entre neuronas tiene asociado un peso que determina la importancia relativa de la entrada en el cálculo de la salida. Estos pesos se ajustan durante el entrenamiento de la red para mejorar su capacidad de aprendizaje.

- ✓ Funciones de Activación: Son funciones matemáticas aplicadas a la salida de cada neurona para introducir no linealidades en la red y permitir el aprendizaje de patrones complejos.

- ✓ Aprendizaje: Las redes neuronales aprenden a partir de ejemplos mediante algoritmos de optimización que ajustan los pesos de las conexiones para minimizar una función de pérdida o error.

**Tipos de Redes Neuronales:**

- Redes Neuronales Artificiales (ANNs): Modelos básicos de redes neuronales con una o más capas ocultas.

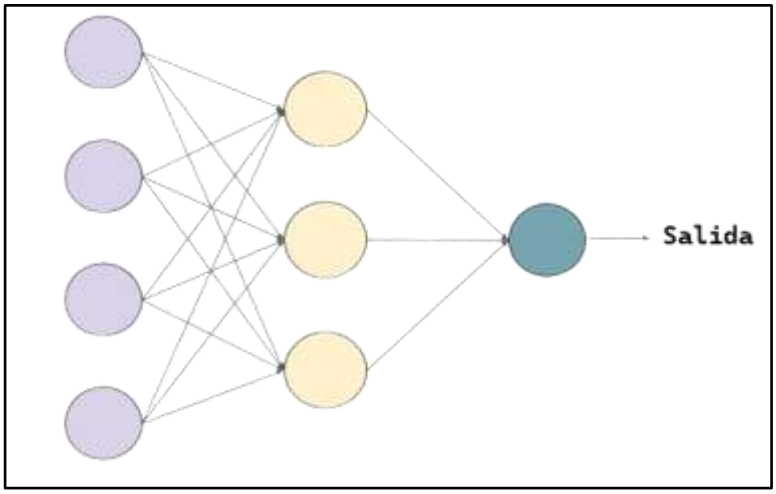


**Figura N° 03:** Redes neuronales feedforward.

- Redes Neuronales Convolucionales (CNNs): Especializadas en el procesamiento de datos de tipo grid, como imágenes y videos.

- Redes Neuronales Recurrentes (RNNs): Diseñadas para trabajar con datos de secuencia, como texto o series temporales.

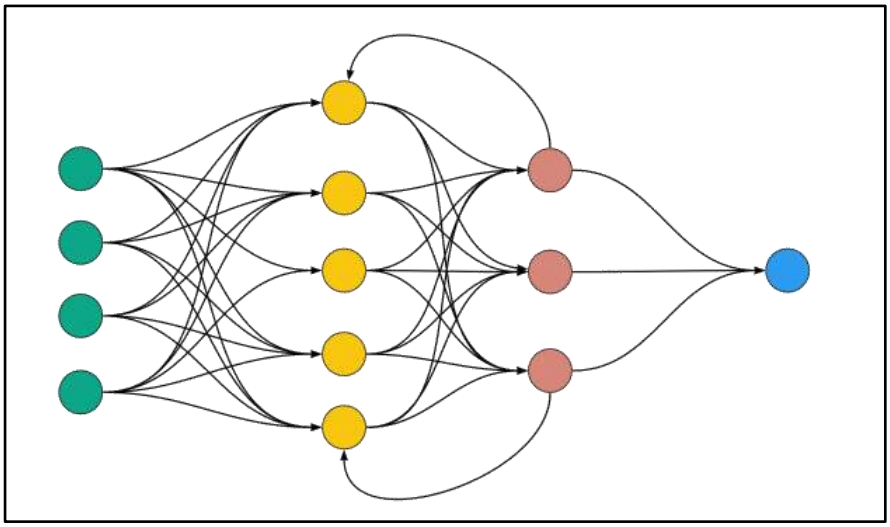

**Figura N° 04:** Redes Neuronales Recurrentes.

- Redes Neuronales Profundas (DNNs): Modelos con múltiples capas ocultas que pueden aprender representaciones jerárquicas de datos.

Las redes neuronales se utilizan en una amplia variedad de aplicaciones, incluyendo reconocimiento de imágenes, procesamiento de lenguaje natural, predicción de series temporales, sistemas de recomendación, entre otros. Su flexibilidad y capacidad para aprender patrones complejos las convierten en una herramienta poderosa en el campo del machine learning y la inteligencia artificial.

**TAREA N°03**

**Estudia el uso de las librerías SciPy y Nltk.**

**Procesamiento de Lenguaje Natural (NLP) con SciPy y NLTK.**

- **Definición de Procesamiento de Lenguaje Natural (NLP).**

  El Procesamiento de Lenguaje Natural (NLP) es una rama de la inteligencia artificial y la lingüística computacional que se centra en la interacción entre las computadoras y el lenguaje humano. Su objetivo es permitir a las máquinas comprender, interpretar y generar texto en forma natural, similar a como lo haría un humano.

  **Características:**

  - ✓ Comprensión del Lenguaje: El NLP permite a las computadoras entender el significado y la intención detrás del lenguaje humano. Esto incluye la comprensión de la semántica (significado), la sintaxis (estructura gramatical) y la pragmática (contexto).

  - ✓ Análisis de Texto: Las técnicas de NLP permiten a las máquinas analizar y procesar grandes cantidades de texto de manera eficiente. Esto incluye tareas como la tokenización (dividir el texto en palabras o frases), el etiquetado de partes del discurso, la extracción de entidades (como nombres de personas o lugares) y el análisis de sentimientos.

  - ✓ Generación de Texto: Además de comprender el lenguaje humano, el NLP también permite a las máquinas generar texto de manera inteligente. Esto incluye la generación de respuestas automáticas, la creación de resúmenes de texto y la redacción de contenido de manera automática.

  - ✓ Aplicaciones Diversas: El NLP se utiliza en una amplia variedad de aplicaciones, que van desde los asistentes virtuales y los motores de búsqueda hasta la traducción automática, el análisis de opiniones en redes sociales, la detección de spam y la asistencia médica basada en texto.

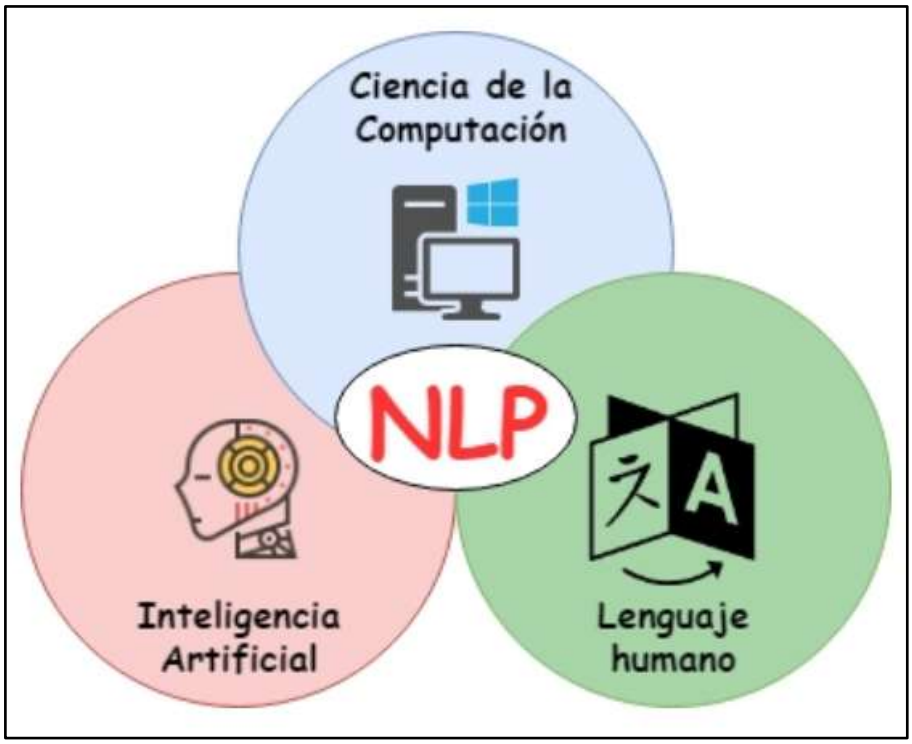


**Figura N° 05:** Procesamiento de Lenguaje Natural (PNL).

**Tecnologías Relacionadas:**

- ✓ Modelos de Lenguaje: Son modelos estadísticos o de aprendizaje profundo entrenados en grandes cantidades de texto para predecir la probabilidad de que una secuencia de palabras sea gramaticalmente correcta y tenga sentido.

- ✓ Transformers: Son arquitecturas de redes neuronales diseñadas específicamente para tareas de procesamiento de lenguaje natural, como la traducción automática y la generación de texto.

- ✓ Word Embeddings: Son representaciones vectoriales de palabras que capturan su significado semántico y su relación con otras palabras en un espacio vectorial de alta dimensión.

En resumen, el Procesamiento de Lenguaje Natural es una disciplina que permite a las máquinas comprender, interpretar y generar texto en forma natural, lo que abre la puerta a una amplia gama de aplicaciones en el ámbito de la inteligencia artificial y la automatización de tareas basadas en texto.

- **Definición de SciPy y NLTK.**

**SciPy:**

SciPy es una biblioteca de Python de código abierto que proporciona herramientas y algoritmos para realizar cálculos científicos y técnicos. Se construye sobre el popular paquete NumPy y extiende sus capacidades con funciones adicionales para la optimización, la interpolación, la integración, el álgebra lineal, la estadística y más.

**Características:**

- ✓ Funcionalidades Avanzadas: Ofrece una amplia gama de algoritmos matemáticos y científicos avanzados.
- ✓ Eficiencia Computacional: Está optimizado para el rendimiento y puede manejar matrices y arreglos multidimensionales de manera eficiente.
- ✓ Facilidad de Uso: Proporciona una interfaz amigable y fácil de usar para realizar cálculos complejos.
- ✓ Integración con NumPy: Se integra estrechamente con NumPy y otros paquetes científicos de Python, lo que permite una interoperabilidad sin problemas.

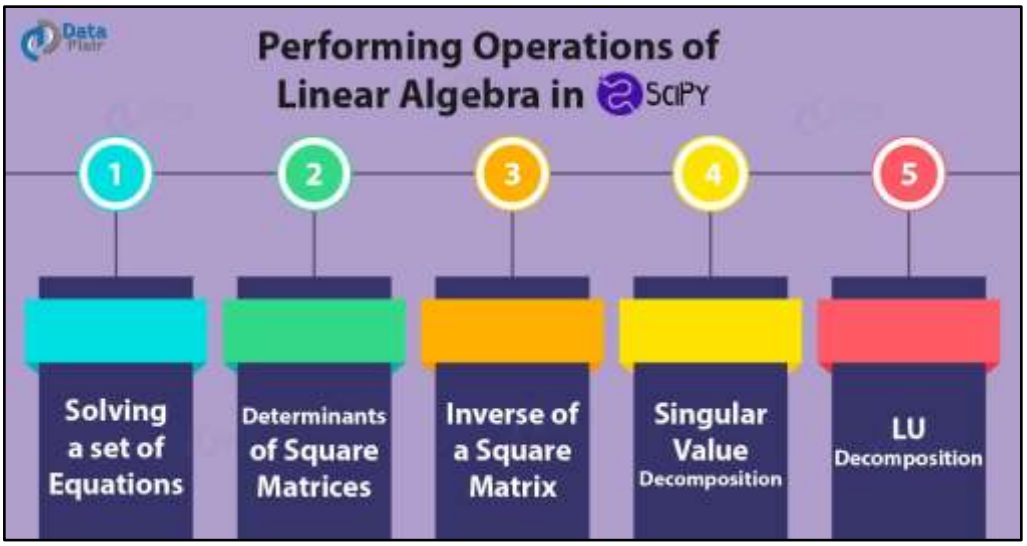


**Figura N° 06:** Operaciones de álgebra lineal en SciPy.

**NLTK:**

NLTK (Natural Language Toolkit) es una biblioteca de Python de código abierto diseñada para el procesamiento de lenguaje natural (NLP). Proporciona una colección de herramientas y recursos para trabajar con texto y realizar tareas como tokenización, etiquetado de partes del discurso, análisis de sentimientos, análisis sintáctico y más.

**Características:**

- ✓ Amplia Funcionalidad: Ofrece una amplia gama de algoritmos y herramientas para el procesamiento de texto y el análisis lingüístico.
- ✓ Facilidad de Uso: Proporciona una interfaz fácil de usar y documentación detallada para facilitar su uso.
- ✓ Recursos Lingüísticos: Incluye una variedad de recursos lingüísticos, como corpus de texto etiquetado y herramientas para el análisis de datos lingüísticos.
- ✓ Comunidad Activa: Cuenta con una comunidad activa de desarrolladores y usuarios que contribuyen con nuevos recursos y funcionalidades.

**Aplicaciones:**

NLTK se utiliza en una variedad de aplicaciones de procesamiento de lenguaje natural, incluyendo la construcción de sistemas de chatbots, análisis de sentimientos en redes sociales, extracción de información, y más.

En resumen, SciPy es una biblioteca para cálculos científicos y técnicos, mientras que NLTK es una biblioteca especializada en el procesamiento de lenguaje natural, ambas ampliamente utilizadas en el campo de la ciencia de datos y el análisis de texto en Python.

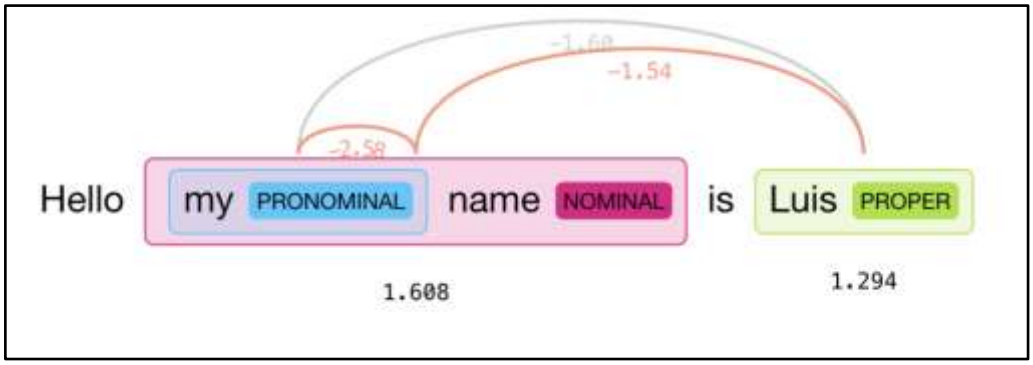


**Figura N° 07:** Ejemplo de correferencia.

- ➢ **Aplicaciones de NLP.**

El Procesamiento de Lenguaje Natural (NLP) tiene una amplia gama de aplicaciones en diversas industrias y campos de estudio. A continuación, se presentan algunas de las aplicaciones más comunes:

- ✓ Búsqueda de Información: Los motores de búsqueda utilizan técnicas de NLP para comprender las consultas de los usuarios y recuperar resultados relevantes de la web.

- ✓ Análisis de Sentimientos: Las empresas utilizan el análisis de sentimientos para comprender la opinión pública sobre sus productos o servicios mediante el procesamiento de reseñas, comentarios en redes sociales y otros textos.

- ✓ Traducción Automática: Herramientas de traducción automática como Google Translate utilizan modelos de NLP para traducir texto entre diferentes idiomas.

- ✓ Asistentes Virtuales: Los asistentes virtuales como Siri, Alexa y Google Assistant utilizan NLP para entender y responder preguntas de los usuarios de manera natural.

- ✓ Resumen Automático: Herramientas de resumen automático utilizan NLP para identificar las ideas principales de un texto largo y generar un resumen conciso.

- ✓ Análisis de Texto Legal: En el ámbito legal, el NLP se utiliza para analizar grandes volúmenes de documentos legales y extraer información relevante para casos judiciales.

- ✓ Clasificación de Texto: Las técnicas de NLP se utilizan para clasificar automáticamente documentos en categorías específicas, como spam o no spam en el caso de correos electrónicos.

- ✓ Procesamiento de Lenguaje Natural en la Salud: En el ámbito de la salud, el NLP se utiliza para extraer información de registros médicos electrónicos, realizar diagnósticos automáticos y analizar informes médicos.

- ✓ Generación de Texto: Algunos sistemas de generación de texto utilizan modelos de lenguaje basados en NLP para producir contenido escrito automáticamente, como noticias, informes financieros y descripciones de productos.

- ✓ Análisis de Opiniones en Redes Sociales: Las empresas utilizan el análisis de sentimientos en las redes sociales para comprender la percepción del público sobre sus marcas y productos mediante el procesamiento de publicaciones y comentarios.

Estas son solo algunas de las muchas aplicaciones del Procesamiento de Lenguaje Natural. Su versatilidad y capacidad para trabajar con datos de texto lo convierten en una herramienta fundamental en el mundo moderno, con aplicaciones que van desde la atención médica y la educación hasta el comercio electrónico y la seguridad nacional.

- **Modelado de lenguaje y tokenización de texto.**

**Modelado de Lenguaje:**
El modelado de lenguaje es una técnica utilizada en el procesamiento de lenguaje natural (NLP) que se centra en la creación de modelos estadísticos o de aprendizaje automático para predecir la probabilidad de que una secuencia de palabras sea gramaticalmente correcta y tenga sentido en un determinado contexto. Estos modelos son entrenados en grandes corpus de texto y pueden capturar patrones complejos en el lenguaje humano.

Características:

- ✓ Predicción de Palabras: Los modelos de lenguaje pueden predecir la siguiente palabra en una secuencia dada una serie de palabras anteriores.
- ✓ Generación de Texto: Basándose en la probabilidad de palabras sucesivas, los modelos de lenguaje pueden generar texto automáticamente que tenga coherencia y sea similar al texto en el corpus de entrenamiento.
- ✓ Evaluación de Frases Los modelos de lenguaje también pueden evaluar la fluidez y la coherencia de las frases o párrafos dados.
- ✓ Aplicaciones: Se utilizan en una amplia gama de aplicaciones, incluyendo la traducción automática, la generación de texto, la corrección gramatical y el análisis de sentimientos.

**Tokenización de Texto:**

La tokenización de texto es el proceso de dividir un texto en unidades más pequeñas llamadas tokens. Estos tokens pueden ser palabras individuales, números, signos de puntuación u otras unidades lingüísticas significativas. La tokenización es un paso fundamental en el preprocesamiento de texto para muchas tareas de procesamiento de lenguaje natural.

**Características:**

- ✓ División en Tokens: El texto se divide en unidades más pequeñas, como palabras individuales o caracteres, para facilitar su procesamiento y análisis.
- ✓ Eliminación de Caracteres Especiales: Los caracteres especiales como signos de puntuación y símbolos se pueden eliminar durante el proceso de tokenización.
- ✓ Normalización de Texto: La tokenización también puede implicar la normalización del texto, como la conversión de letras mayúsculas a minúsculas para evitar la distinción entre palabras que solo difieren en la capitalización.

- ✓ Preparación para Análisis: La tokenización prepara el texto para tareas posteriores de análisis de texto, como el etiquetado de partes del discurso, la extracción de entidades y el modelado de lenguaje.

Aplicaciones: La tokenización se utiliza en una variedad de aplicaciones de NLP, incluyendo la construcción de modelos de lenguaje, la generación de vectores de palabras (word embeddings), el análisis de sentimientos y más.

En resumen, el modelado de lenguaje se centra en la creación de modelos estadísticos para predecir el siguiente token en una secuencia de palabras, mientras que la tokenización de texto es el proceso de dividir un texto en unidades más pequeñas para su procesamiento y análisis en tareas de NLP.

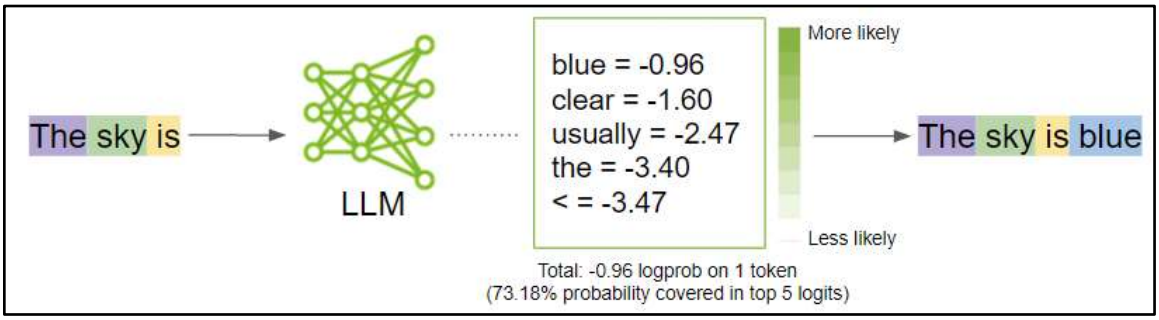


**Figura N° 08:** Flujo de trabajo de un LLM que predice la siguiente palabra.

**TAREA N°04**

**Estudia el uso de las librerías SciPy y Nltk.**

**DeepLearning con Tensorflow y Keras.**

- **Concepto de Deep Learning.**

  El Deep Learning es una rama del aprendizaje automático (Machine Learning) que se basa en la estructura y función de las redes neuronales artificiales, inspiradas en el funcionamiento del cerebro humano. Se caracteriza por el uso de algoritmos complejos y modelos computacionales con múltiples capas de procesamiento para aprender representaciones de datos de alto nivel.

  **Características:**

  - ✓ Redes Neuronales Profundas: Utiliza redes neuronales artificiales con múltiples capas ocultas (capas intermedias) para aprender representaciones jerárquicas de datos.
  - ✓ Aprendizaje Jerárquico de Características: Las capas intermedias de la red aprenden representaciones progresivamente más abstractas y complejas de los datos de entrada.
  - ✓ Representaciones de Datos de Alto Nivel: El Deep Learning es capaz de aprender automáticamente características y patrones complejos en datos no estructurados, como imágenes, audio y texto.
  - ✓ Abstracción y Generalización: A medida que las capas más profundas de la red aprenden representaciones más abstractas, el modelo puede generalizar mejor a datos nuevos y desconocidos.

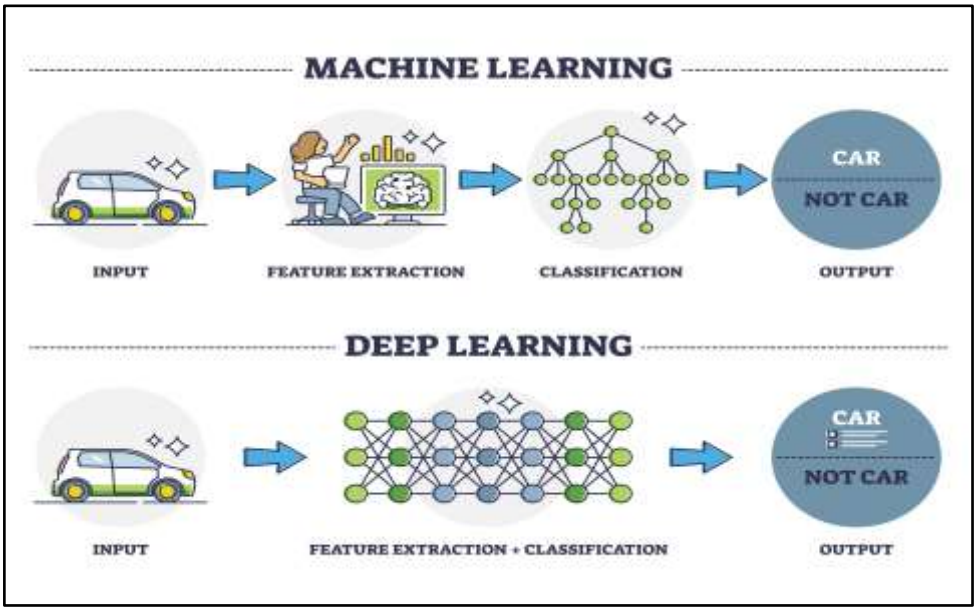


**Figura N° 09:** Principales diferencias entre el ML y el DL.

**Aplicaciones:**

- ✓ Visión por Computadora: Clasificación de imágenes, detección de objetos, reconocimiento facial.
- ✓ Procesamiento de Lenguaje Natural: Traducción automática, generación de texto, análisis de sentimientos.
- ✓ Reconocimiento de Voz: Reconocimiento de voz, síntesis de voz, transcripción automática.
- ✓ Biomedicina: Diagnóstico médico, análisis de imágenes médicas, descubrimiento de fármacos.
- ✓ Conducción Autónoma: Detección de peatones, seguimiento de carriles, reconocimiento de señales de tráfico.

**Desafíos:**

- ✓ Requiere Grandes Conjuntos de Datos: El Deep Learning requiere grandes conjuntos de datos etiquetados para entrenar modelos de manera efectiva.
- ✓ Computación Intensiva: El entrenamiento de modelos de Deep Learning puede ser computacionalmente costoso y requiere hardware especializado como GPU y TPU.
- ✓ Interpretación de Modelos: Los modelos de Deep Learning pueden ser difíciles de interpretar y explicar cómo toman decisiones.

En resumen, el Deep Learning es una técnica de aprendizaje automático que utiliza redes neuronales profundas para aprender representaciones jerárquicas de datos complejos. Tiene una amplia gama de aplicaciones en áreas como visión por

computadora, procesamiento de lenguaje natural, biomedicina y conducción autónoma.

- **Definición de Tensorflow y Keras.**

  - ✓ TensorFlow:

    TensorFlow es una biblioteca de código abierto desarrollada por Google para realizar cálculos numéricos y construir modelos de aprendizaje automático, especialmente redes neuronales. Se destaca por su flexibilidad y escalabilidad, permitiendo a los desarrolladores implementar y entrenar modelos complejos en una variedad de plataformas, desde dispositivos móviles hasta sistemas distribuidos.

  - ✓ Características de TensorFlow:
    - Grafos Computacionales: Define modelos de aprendizaje automático como grafos computacionales, donde los nodos representan operaciones matemáticas y las aristas representan los datos.
    - Optimización Automática: TensorFlow optimiza automáticamente el flujo de cálculos para aprovechar al máximo el hardware subyacente, incluyendo CPU, GPU y TPU.
    - Flexibilidad: Permite la construcción de una amplia gama de modelos, desde redes neuronales convolucionales hasta modelos de lenguaje natural y sistemas de recomendación.
    - Despliegue en Producción: Facilita el despliegue de modelos entrenados en producción a través de API REST, TensorFlow Serving y otras herramientas.

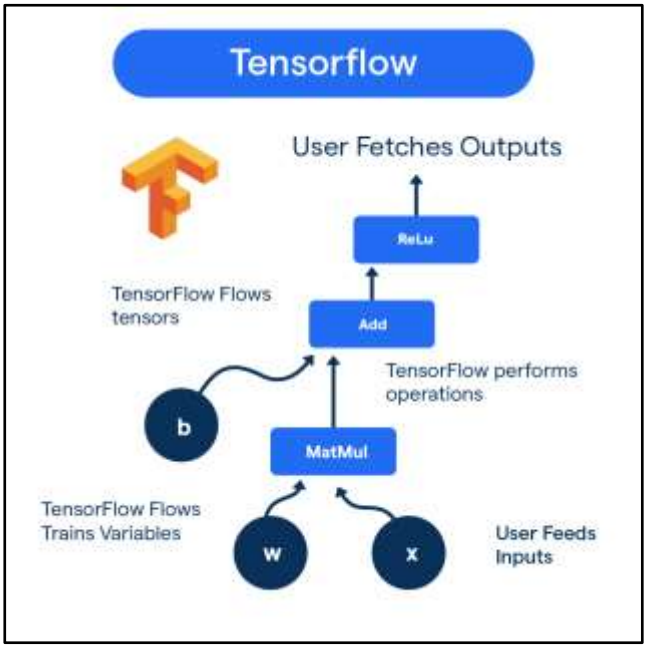


**Figura N° 10:** Tensorflow.

✓ Keras:

Keras es una biblioteca de redes neuronales de alto nivel escrita en Python que sirve como interfaz de alto nivel para TensorFlow y otras bibliotecas de aprendizaje automático como Theano y Microsoft Cognitive Toolkit (CNTK). Está diseñado para ser simple, modular y extensible, lo que facilita la experimentación con diferentes arquitecturas de redes neuronales.

✓ Características de Keras:

- Facilidad de Uso: Proporciona una API simple y coherente para construir y entrenar modelos de redes neuronales.
- Modularidad: Los modelos se definen como una secuencia o un grafo de capas, lo que permite una fácil composición y reutilización de componentes.
- Flexibilidad: Permite la creación de una amplia variedad de arquitecturas de redes neuronales, desde modelos simples hasta modelos complejos con múltiples ramificaciones y conexiones.
- Compatibilidad con TensorFlow: Keras se integra estrechamente con TensorFlow, lo que permite utilizar todas las funcionalidades y optimizaciones de TensorFlow en los modelos creados con Keras.

✓ Aplicaciones:

TensorFlow y Keras se utilizan en una amplia gama de aplicaciones de aprendizaje automático y deep learning, incluyendo visión por computadora, procesamiento de lenguaje natural, reconocimiento de voz, robótica, y más.

En resumen, TensorFlow es una biblioteca de bajo nivel para realizar cálculos numéricos y construir modelos de aprendizaje automático, mientras que Keras es una interfaz de alto nivel que simplifica la construcción y entrenamiento de modelos de redes neuronales utilizando TensorFlow y otras bibliotecas de aprendizaje automático.

➢ **Tipos de redes neuronales y funciones de activación.**

- Feedforward Neural Networks (Redes Neuronales Feedforward): También conocidas como redes neuronales densas o perceptrones multicapa, son el tipo más básico de red neuronal. La información fluye en una sola dirección, desde las capas de entrada, a través de una o más capas ocultas, hasta las capas de salida.

- Convolutional Neural Networks (Redes Neuronales Convolucionales): Diseñadas específicamente para el procesamiento de datos en forma de mallas (como

imágenes). Utilizan capas de convolución para aplicar filtros a pequeñas regiones de entrada, lo que permite aprender características espaciales jerárquicas.

- Recurrent Neural Networks (Redes Neuronales Recurrentes): Adecuadas para datos secuenciales, como texto o series temporales. Tienen conexiones retroalimentadas que les permiten mantener estados internos y aprender dependencias a largo plazo en secuencias de entrada.

- Long Short-Term Memory Networks (Redes Neuronales de Memoria a Corto y Largo Plazo): Una variante de las redes neuronales recurrentes que abordan el problema de la desaparición del gradiente. Utilizan unidades de memoria que pueden recordar secuencias de entrada durante largos períodos de tiempo.

- Autoencoder Neural Networks (Redes Neuronales Autoencoder): Utilizadas para aprendizaje no supervisado y reducción de dimensionalidad. Consisten en una red neuronal que aprende a reconstruir la entrada original a través de una representación latente de dimensionalidad reducida.

**Funciones de Activación:**

- Función Sigmoide (Sigmoid): Transforma la entrada en un rango de valores entre 0 y 1, útil para problemas de clasificación binaria.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Función ReLU (Rectified Linear Unit): Retorna cero para entradas negativas y la entrada misma para valores positivos, acelerando el entrenamiento y evitando el problema de la desaparición del gradiente.

$$f(x) = \max(0, x)$$

- Función Tanh (Tangente Hiperbólica): Similar a la función sigmoide pero con un rango entre -1 y 1, útil para problemas de clasificación binaria o regresión.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- Función Softmax: Utilizada en la capa de salida de redes neuronales para problemas de clasificación multiclase, transforma las salidas en una distribución de probabilidad.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- Función de Activación Leaky ReLU: Similar a ReLU, pero permite un pequeño gradiente para valores negativos, evitando que los nodos mueran durante el entrenamiento.

$$\begin{cases} x & si\ x > 0 \\ \propto x & si\ x \leq 0 \end{cases}$$

Estos tipos de redes neuronales y funciones de activación son fundamentales en la construcción y entrenamiento de modelos de deep learning para una variedad de tareas, desde clasificación y reconocimiento hasta generación de texto y procesamiento de lenguaje natural.

**TAREA N°05**

**Realiza operaciones con las librerías Matplotlib y Seaborn.**

**Visualización de datos con Matplotlib y Seaborn.**

- ✓ **Matplotlib:**
  Matplotlib es una biblioteca de visualización de datos en Python que permite crear gráficos de alta calidad y altamente personalizables. Ofrece una variedad de estilos de gráficos, incluyendo gráficos de líneas, dispersión, barras, histogramas, diagramas de caja, entre otros.

- ✓ Características de Matplotlib:
  - Flexibilidad: Permite control total sobre cada aspecto del gráfico, desde los ejes y etiquetas hasta los colores y estilos de línea.
  - Soporte para múltiples formatos de salida: Los gráficos se pueden guardar en una variedad de formatos de archivo, incluyendo PNG, PDF, SVG y más.
  - Integración con Jupyter Notebooks: Matplotlib se integra perfectamente con los Notebooks de Jupyter, lo que permite crear y mostrar gráficos directamente en el entorno de desarrollo.
  - Compatibilidad con Seaborn: Seaborn se basa en Matplotlib y proporciona estilos de gráficos adicionales y funciones de alto nivel para visualización de datos más avanzada.

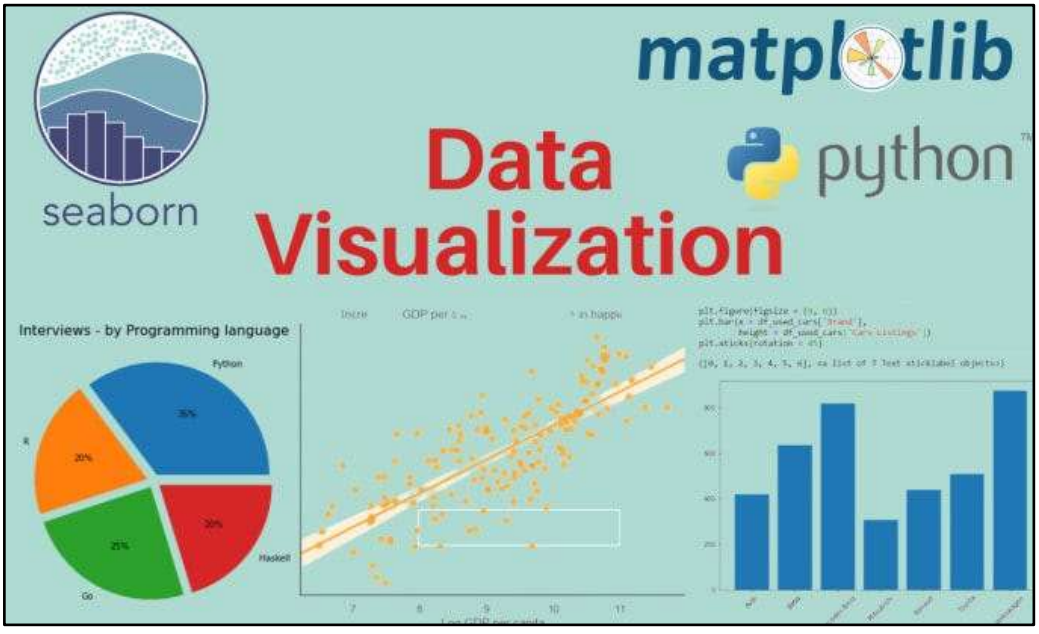


**Figura N° 11:** Visualización de los datos.

- ✓ **Seaborn:**

  Seaborn es una biblioteca de visualización de datos construida sobre Matplotlib que proporciona una interfaz de alto nivel para crear gráficos estadísticos atractivos y informativos. Seaborn simplifica la creación de gráficos complejos y ofrece una amplia variedad de paletas de colores y estilos de gráficos.

- ✓ Características de Seaborn:
  - Estilos de gráficos predefinidos: Seaborn proporciona una variedad de estilos de gráficos predefinidos que pueden aplicarse fácilmente a cualquier gráfico.
  - Gráficos estadísticos avanzados: Ofrece funciones especializadas para crear gráficos estadísticos complejos, como diagramas de violín, gráficos de regresión y mapas de calor.
  - Integración con pandas: Seaborn es compatible con DataFrames de pandas, lo que facilita la visualización de datos almacenados en estructuras de datos tabulares.
  - Facilidad de uso: Seaborn proporciona una API intuitiva y fácil de usar para la creación de gráficos, lo que permite a los usuarios enfocarse en la exploración y visualización de datos en lugar de detalles técnicos.

- ✓ Aplicaciones:

  Matplotlib y Seaborn se utilizan en una amplia gama de aplicaciones, incluyendo análisis exploratorio de datos, visualización de resultados de modelos de aprendizaje automático, comunicación de resultados científicos y más.

En resumen, Matplotlib y Seaborn son dos bibliotecas poderosas y complementarias para la visualización de datos en Python, que proporcionan una variedad de estilos de gráficos y funciones para crear gráficos informativos y visualmente atractivos.

- ➢ **Histogramas.**

  En Python, puedes crear histogramas fácilmente utilizando bibliotecas como Matplotlib y Seaborn. Aquí te muestro cómo hacerlo:

  Histogramas con Matplotlib:

```
import matplotlib.pyplot as plt

# Datos de ejemplo
data = [1, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 6, 6, 6, 7, 7, 7, 7, 7, 8]
```

```
# Crear el histograma
plt.hist(data, bins=5, color='skyblue', edgecolor='black')

# Personalizar el gráfico
plt.title('Histograma de Datos')
plt.xlabel('Valor')
plt.ylabel('Frecuencia')
plt.grid(True)

# Mostrar el histograma
plt.show()
```

Histogramas con Seaborn:

```
import seaborn as sns

# Datos de ejemplo
data = [1, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 6, 6, 6, 7, 7, 7, 7, 7, 8]

# Crear el histograma con Seaborn
sns.histplot(data, bins=5, color='skyblue', edgecolor='black')

# Personalizar el gráfico
plt.title('Histograma de Datos')
plt.xlabel('Valor')
plt.ylabel('Frecuencia')

# Mostrar el histograma
plt.show()
```

Estos ejemplos crearán un histograma de los datos proporcionados con 5 bins (cajas). Puedes ajustar el número de bins según tus necesidades para obtener una representación más detallada o menos detallada de la distribución de tus datos.

- **Visualización de datos, Importancia.**

  La visualización de datos es fundamental en el análisis y la interpretación de información, ya que proporciona una representación gráfica de los datos que facilita su comprensión y análisis. Aquí te presento algunas razones clave por las cuales la visualización de datos es importante:

- ✓ Comprensión más rápida y fácil:

  Las visualizaciones de datos permiten representar información compleja de manera visual, lo que facilita la comprensión de patrones, tendencias y relaciones en los datos de forma más rápida y fácil que mediante la exploración de tablas o números.

- ✓ Identificación de patrones y tendencias:

  Al representar datos visualmente, es más fácil identificar patrones, tendencias, picos, valles y anomalías en los datos. Esto puede proporcionar información valiosa para la toma de decisiones y la formulación de estrategias.

- ✓ Comunicación efectiva:

  Las visualizaciones de datos son herramientas poderosas para comunicar información de manera efectiva a audiencias diversas, incluidos colegas, clientes, gerentes y partes interesadas. Las imágenes pueden transmitir mensajes complejos de manera más clara y persuasiva que el texto o los números.

- ✓ Exploración y descubrimiento de datos:

  Las visualizaciones de datos facilitan la exploración y el descubrimiento de patrones ocultos o insights en los datos. Al interactuar con gráficos interactivos o explorar diferentes visualizaciones, los analistas de datos pueden descubrir relaciones y correlaciones interesantes que pueden haber pasado desapercibidas de otra manera.

- ✓ Soporte para la toma de decisiones:

  La visualización de datos proporciona información visual que puede ayudar a respaldar la toma de decisiones informadas en una amplia gama de campos, incluida la ciencia, los negocios, la medicina, la ingeniería y más. Las visualizaciones pueden ayudar a evaluar opciones, identificar oportunidades y mitigar riesgos.

- ✓ Monitoreo y seguimiento:

  Las visualizaciones de datos pueden utilizarse para monitorear métricas clave, seguimiento del progreso hacia objetivos y detectar cambios o desviaciones en tiempo real. Esto es especialmente útil en aplicaciones como el análisis de rendimiento empresarial, la vigilancia de la salud pública y el control de procesos industriales.

En resumen, la visualización de datos es una herramienta crucial en el análisis de datos y la toma de decisiones, ya que permite comprender rápidamente los datos, identificar patrones y tendencias, comunicar información de manera efectiva y respaldar la toma de decisiones informadas.

- **Definición de Matplotlib y Seaborn.**

**Matplotlib:**

Matplotlib es una biblioteca de visualización de datos en Python que permite crear gráficos de alta calidad y altamente personalizables. Es una de las bibliotecas más utilizadas para la visualización de datos en Python y ofrece una amplia variedad de estilos de gráficos, incluyendo gráficos de líneas, dispersión, barras, histogramas, diagramas de caja y más. Matplotlib es altamente flexible y permite controlar cada aspecto del gráfico, desde los ejes y etiquetas hasta los colores y estilos de línea. Se integra bien con otras bibliotecas de Python, como NumPy y Pandas, y es ampliamente utilizado en campos como la ciencia de datos, la investigación académica y la ingeniería.

**Seaborn:**

Seaborn es una biblioteca de visualización de datos construida sobre Matplotlib que proporciona una interfaz de alto nivel para crear gráficos estadísticos atractivos y informativos. Seaborn simplifica la creación de gráficos complejos y ofrece una amplia variedad de paletas de colores y estilos de gráficos. Seaborn proporciona funciones especializadas para crear gráficos estadísticos avanzados, como diagramas de violín, gráficos de regresión y mapas de calor. Se integra bien con DataFrames de Pandas y se utiliza comúnmente en campos como la ciencia de datos, la investigación biomédica y la visualización de datos en general.

En resumen, Matplotlib y Seaborn son dos bibliotecas poderosas y complementarias para la visualización de datos en Python, que proporcionan una variedad de estilos de gráficos y funciones para crear gráficos informativos y visualmente atractivos.

- **Ejemplos de aplicación.**

**Matplotlib:**

- ✓ Visualización de Datos Financieros: Matplotlib se utiliza para crear gráficos de líneas y gráficos de velas para mostrar la evolución de los precios de las acciones a lo largo del tiempo, ayudando a los inversores a identificar tendencias y patrones en los mercados financieros.

- ✓ Gráficos Científicos: Los científicos utilizan Matplotlib para representar datos experimentales, como espectros, curvas de crecimiento y mapas de contorno, facilitando la visualización y la interpretación de los resultados de sus investigaciones.

- ✓ Visualización de Datos Geoespaciales: Matplotlib se utiliza en la visualización de datos geoespaciales para crear mapas y gráficos que representan la distribución geográfica de datos, como la densidad de población, el clima o la distribución de recursos naturales.

**Seaborn:**

- ✓ Análisis de Datos Exploratorios: Seaborn es ampliamente utilizado en el análisis de datos exploratorios para visualizar relaciones entre variables y explorar patrones en los datos. Por ejemplo, se puede utilizar para crear gráficos de dispersión y diagramas de regresión para analizar la relación entre dos variables.

- ✓ Visualización de Datos Categóricos: Seaborn ofrece funciones especializadas para visualizar datos categóricos, como diagramas de barras y diagramas de violín, que permiten comparar la distribución de datos entre diferentes grupos o categorías.

- ✓ Mapas de Calor para Análisis de Correlación: Seaborn es útil para crear mapas de calor que visualizan la matriz de correlación entre variables en un conjunto de datos, lo que facilita la identificación de relaciones y patrones de correlación entre diferentes variables.

**Ejemplo de Código:**

```
import matplotlib.pyplot as plt
import seaborn as sns
# Datos de ejemplo
data = sns.load_dataset('iris')
# Crear un gráfico de dispersión con Seaborn
sns.scatterplot(x='sepal_length', y='petal_length', data=data, hue='species', palette='Set2')

# Personalizar el gráfico con Matplotlib
plt.title('Relación entre Longitud de Sépalo y Longitud de Pétalo')
plt.xlabel('Longitud de Sépalo')
plt.ylabel('Longitud de Pétalo')


# Mostrar el gráfico
plt.show()
```

Este ejemplo utiliza datos de la flor de iris cargados con Seaborn para crear un gráfico de dispersión que muestra la relación entre la longitud del sépalo y la longitud del pétalo para diferentes especies de iris. La función `scatterplot` de Seaborn se utiliza para crear el gráfico de dispersión, y luego Matplotlib se utiliza para personalizar el título y los ejes del gráfico antes de mostrarlo.

- **Exportar y guardar imágenes en Python.**

Puedes exportar y guardar imágenes generadas en Python utilizando las bibliotecas Matplotlib y Seaborn. Aquí te muestro cómo hacerlo:

**Exportar y Guardar Imágenes con Matplotlib:**

```
import matplotlib.pyplot as plt

# Crear un gráfico
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])

# Guardar el gráfico como imagen PNG
plt.savefig('grafico.png')

# Mostrar el gráfico
plt.show()
```

En este ejemplo, el gráfico se guarda como una imagen PNG con el nombre "grafico.png" en el directorio actual.

**Exportar y Guardar Imágenes con Seaborn:**

```
import seaborn as sns

# Cargar datos de ejemplo
data = sns.load_dataset('iris')

# Crear un gráfico de dispersión
sns.scatterplot(x='sepal_length', y='petal_length', data=data, hue='species', palette='Set2')
```

```
# Guardar el gráfico como imagen PNG
plt.savefig('scatterplot.png')

# Mostrar el gráfico
plt.show()
```

Aquí, el gráfico de dispersión se guarda como una imagen PNG con el nombre "scatterplot.png" en el directorio actual.

Ambos ejemplos utilizan la función `savefig()` de Matplotlib para guardar la imagen en el formato especificado. Puedes especificar la extensión del archivo en el nombre del archivo para guardar la imagen en el formato deseado (por ejemplo, PNG, JPG, PDF, SVG, etc.).

Recuerda que después de llamar a `savefig()`, debes llamar a `plt.show()` o `plt.close()` para limpiar el gráfico y evitar que se guarde una imagen vacía.

- **Tipos de gráficos estadísticos para visualización de datos.**

Existen varios tipos de gráficos estadísticos que se pueden utilizar para visualizar datos de manera efectiva. Aquí tienes algunos de los tipos más comunes:

- ✓ Gráfico de Barras: Ideal para comparar cantidades o categorías. Se utilizan barras rectangulares de longitud proporcional a los valores que representan.
- ✓ Gráfico de Líneas: Muestra la relación entre dos variables continuas a lo largo de un eje x. Útil para representar tendencias y cambios a lo largo del tiempo.
- ✓ Gráfico de Dispersión: Muestra la relación entre dos variables continuas a través de puntos en un plano cartesiano. Útil para identificar correlaciones entre variables.
- ✓ Histograma: Representa la distribución de una variable numérica dividida en intervalos (bins). Útil para comprender la distribución de los datos y detectar patrones.
- ✓ Diagrama de Caja (Boxplot): Representa la distribución de los datos a través de cuartiles. Proporciona información sobre la mediana, el rango intercuartílico y los valores atípicos.
- ✓ Diagrama de Violín: Combina un histograma y un diagrama de caja para mostrar la distribución de los datos de manera más detallada.

- ✓ Gráfico de Pastel: Muestra la proporción de cada categoría en un conjunto de datos como una porción de un círculo. Útil para representar distribuciones de datos categóricos.

- ✓ Diagrama de Área: Similar a un gráfico de líneas, pero el área debajo de la línea se colorea para resaltar la magnitud acumulada de los valores.

- ✓ Mapa de Calor: Muestra la densidad o la concentración de datos en un mapa bidimensional utilizando colores. Útil para visualizar correlaciones en matrices de datos.

- ✓ Diagrama de Sankey: Representa relaciones proporcionales entre variables a través de líneas de flujo con diferentes anchos.

Estos son solo algunos ejemplos de gráficos estadísticos comunes que se utilizan para visualizar datos de manera efectiva en diferentes contextos y situaciones. La elección del tipo de gráfico depende del tipo de datos que estés visualizando y del mensaje que desees comunicar.

## REFERENCIAS BIBLIOGRÁFÍCAS


1. Ali Awan, A. (09 de mayo de 2024). *¿Qué es la tokenización?* https://www.datacamp.com/es/blog/what-is-tokenization
2. Heidloff, N. (05 de setiembre de 2023). *Python y PyTorch para ingenieros de AI*. https://heidloff.net/article/python-pytorch/
3. Huet, P. (13 de abril de 2023). *Qué son las redes neuronales y sus aplicaciones*. https://openwebinars.net/blog/que-son-las-redes-neuronales-y-sus-aplicaciones/
4. ID boot camps. (18 de julio de 2023). *¿Qué son las librerías de Python?* https://iddigitalschool.com/bootcamps/que-son-las-librerias-de-python/
5. Johnson, D. (23 de diciembre de 2023). *Tutorial de Python Pandas: marco de datos, rango de fechas, uso de Pandas*. https://www.guru99.com/es/python-pandas-tutorial.html
6. Mehardil. (27 de marzo de 2024). *Matplotlib in python*. https://medium.com/@mehardil13302/matplotlib-in-python-d72a0d874447
7. Meneses, N. (14 de diciembre de 2023). *¿Qué es Pandas de Python? Guía para principiantes*. https://www.codingdojo.la/2023/12/14/que-es-pandas-de-python-guia-para-principiantes/
8. Merino Ulizarma, L. (27 de julio de 2023). *NLTK: tus primeros pasos con Procesamiento del Lenguaje Natural*. https://adictosaltrabajo.com/2023/07/27/nltk-python/
9. NumPy. (16 de setiembre de 2023). *El paquete fundamental para la computación científica con Python*. https://numpy.org/
10. Pandas. (10 de abril de 2024). *documentación de Pandas*. https://pandas.pydata.org/docs/
11. Pompas Gutiérrez, J. (19 de julio de 2023). *Introducción a PyTorch: Un tutorial paso a paso*. https://www.linkedin.com/pulse/introducci%C3%B3n-pytorch-un-tutorial-paso-jordi-pompas-guti%C3%A9rrez/
12. Surla, A. (18 de setiembre de 2023). *Cómo Obtener Mejores Resultados de su Grande Modelo de Lenguaje*. https://la.blogs.nvidia.com/blog/como-obtener-mejores-resultados-de-su-grande-modelo-de-lenguaje/
13. Toledano López, O. G. (13 de junio de 2022). *Procesamiento de lenguaje natural y sus aplicaciones (I)*. https://www.tecnoblog.org/desarrollo/procesamiento-lenguaje-natural/
14. Torres, A. (17 de julio de 2022). *Tutorial: ¿Qué es el Análisis de Datos? Cómo visualizar datos con Python, Numpy, Pandas, Matplotlib y Seaborn*. https://www.freecodecamp.org/espanol/news/tutorial-que-es-analisis-de-datos-como-visualizar-datos-con-python-numpy-pandas-matplotlib-y-seaborn/
15. Turing. (s.f.). *Aprendizaje profundo vs Machine Learning: La batalla final*. https://www.turing.com/kb/ultimate-battle-between-deep-learning-and-machine-learning
16. Zapata, J. (11 de noviembre de 2023). *NUMPY - Vectores y Matrices*. https://joserzapata.github.io/courses/python-ciencia-datos/numpy/

## REFERENCIAS DE IMÁGENES

**Figura N° 01:** Scikit-Learn y Python.
https://serinteligenciaartificial.com/wp-content/uploads/2023/12/f693dd79628fbdfa9bb751af7b1ea9888dfb2aee-2152x864-1-768x308.webp

**Figura N° 02:** Redes neuronales.
https://dc722jrlp2zu8.cloudfront.net/media/uploads/2023/04/04/4.png

**Figura N° 03:** Redes neuronales feedforward.
https://dc722jrlp2zu8.cloudfront.net/media/uploads/2023/04/04/10.png

**Figura N° 04:** Redes Neuronales Recurrentes.
https://dc722jrlp2zu8.cloudfront.net/media/uploads/2023/04/04/11.png

**Figura N° 05:** Procesamiento de Lenguaje Natural (PNL).
https://www.tecnoblog.org/wp-content/uploads/2021/06/nlp-2.png

**Figura N° 06:** Operaciones de álgebra lineal en SciPy.
https://data-flair.training/blogs/wp-content/uploads/sites/2/2018/07/Performing-Operations-of-Linear-Algebra-in-SciPy-01.jpg

**Figura N° 07:** Ejemplo de correferencia.
https://adictosaltrabajo.com/wp-content/uploads/2023/05/Captura-de-pantalla-2023-05-17-a-las-19.05.49-1068x367.png

**Figura N° 08:** Flujo de trabajo de un LLM que predice la siguiente palabra.
https://developer-blogs.nvidia.com/wp-content/uploads/2023/06/general-workflow-for-prompt.png

**Figura N° 09:** Principales diferencias entre el ML y el DL.
https://images.prismic.io/turing/652ebc26fbd9a45bcec81819_Deep_Learning_vs_Machine_Learning_3033723be2.webp?auto=format%2Ccompress&fit=max&w=3840

**Figura N° 10:** Tensorflow.
https://cdn.botpenguin.com/assets/website/Tensorflow_0e2dd82f94.png

**Figura N° 11:** Visualización de los datos.
https://miro.medium.com/v2/resize:fit:1100/format:webp/0*amy4SywgZtppJDlt.png

RDA

RECURSO DIDÁCTICO PARA EL APRENDIZAJE