Programación Avanzada Python Taller 1

Leandro Coral (00325644)

4 de septiembre de 2025

1. Pregunta 1

Crea una función que realice un análisis completo de un texto. El texto debe ser el resumen de un artículo cientifico La función debe devolver un diccionario con: el número total de palabras omitiendo artículos y preposiciones, número de palabras únicas omitiendo artículos y preposiciones, la palabra más larga, la palabra con más vocales, la frecuencia de cada letra (ignorando mayúsculas/minúsculas), las 5 palabras más comunes, palabras con mayor densidad (longitud/frecuencia) y la longitud promedio de palabras.

Resumen

Este trabajo presenta una función en Python diseñada para realizar un análisis textual de un resumen científico. El sistema procesa el texto eliminando símbolos, acentos, artículos y preposiciones, para posteriormente aplicar métricas de análisis léxico. Entre los resultados obtenidos, se calculan el número total y único de palabras, la palabra más larga y la que contiene mayor cantidad de vocales. Además, se determina la frecuencia de aparición de cada letra, las cinco palabras más comunes y aquellas con mayor densidad definida como la relación entre longitud y frecuencia de uso. Finalmente, se estima la longitud promedio de las palabras. Este análisis proporciona una visión integral de las características lingüísticas del texto.

1.1. Análisis del código

Para el analisis de codigo se usa siguiente repositorio: https://github.com/Lhao13

1.1.1. Primera parte del codigo

```
Articulo = "Cada d a se hace cada vez m s evidente el uso de la inteligencia
    artificial en nuestra vida cotidiana sin embargo a n quedan muchos elementos para
    trabajar y organizar antes de poder realizar una verdadera implementaci n de la
    inteligencia artificial en la educaci n , aun as  no deja de ser una gran ventaja
    y ayuda a la hora de impartir y generar conocimientos esto sumado a los retos del
    siglo XXI que buscan una integralidad y una verdadera transversalidad de la
    tecnolog a y en los diversos ejes del saber, este documento un tiene como objetivo
     hacer una reflexi n sobre la importancia y la verdadera utilidad de la
    implementaci n y asistencia de la IA en nuestra labor docente tambi n que permite
     ver claros ejemplos a nivel mundial sobre alfabetizaci n digital que apunta a
    encamina a comprender m s a profundidad sobre la verdadera utilidad y practicidad
    de la IA, tambi n enfocar y construir verdaderas competencias pedag gicas
    orientadas a construir un pensamiento cient fico y tecnol gico."

Simbolos= [".", ",", ";", ":", "!", "?"]
reemplazos = (("  ", "a"), ("  ", "e"), ("  ", "i"), ("  ", "o"), ("  ", "u"))
ArticulosYPrepocisiones = ["el", "la", "los", "lo", "las", "un", "una", "unos", "unas"
    ,"a", "ante", "bajo", "con", "contra", "de", "desde", "durante","en", "entre", "
    hacia", "hasta", "mediante", "para", "sin", "sobre", "a", "ante", "bajo", "cabe", "
    con", "contra", "de", "desde", "durante","en", "entre", "hacia", "hasta", "mediante
    ", "para", "por", "segun","sin", "so", "sobre", "tras", "versus"]

SinSimbolos = Articulo.lower()

for s in Simbolos:
    SinSimbolos = SinSimbolos.replace(s, "")
```

```
11
12 for original, nuevo in reemplazos:
13     SinSimbolos = SinSimbolos.replace(original, nuevo)
14
15 SinArticulos = SinSimbolos.split()
16
17 for p in ArticulosYPrepocisiones:
18     if p in SinArticulos:
19         SinArticulos = [x for x in SinArticulos if x != p]
```

Listing 1: Primera parte

En la primera parte del codigo se define el articulo como un string en la linea 1. Las lineas 3, 4 y 5 definen, *Simbolos*: caracteres que se eliminarán del texto para evitar interferencias en el conteo. *reemplazos*: tuplas que indican cómo normalizar vocales con tilde a su versión sin tilde (por ejemplo, "á"→ .ª"). *ArticulosYPrepocisiones*: lista de artículos y preposiciones que se eliminarán porque no aportan significado relevante en el análisis.

A continuación en la linea 7 se cambia todo el string a minúsculas. La linea 9 a la 19 se hacen diferentes bucles de eliminación y remplazo. Primer bucle recorre cada símbolo definido en la lista *Simbolos* y lo elimina del texto, dejándolo limpio de puntuaciones. Segundo bucle, Reemplaza las vocales acentuadas por su equivalente sin tilde para unificar palabras (ejemplo: "educación" → "educacion"). Tercer bucle, Recorre la lista de "*ArticulosYPrepocisiones* y elimina todas sus apariciones de la lista de palabras, dejando solo términos relevantes para el análisis.

1.1.2. Parte de analisis del codigo

```
1
2 # Cantidad de palabras
3 NumeroDePalabras = len(SinArticulos)
4
5 # Cantidad de palabras  nicas
6 NumeroDePalabrasUnicas = len(set(SinArticulos))
7
8 # Palabra m s larga
9 PalabraMasLarga = max(SinArticulos, key=len)
```

Listing 2: Parte de analisis

La primera función calcula el total de palabras que quedaron en el texto después de la limpieza. La segunda usa un set (conjunto) para eliminar duplicados y contar cuántas palabras distintas hay. La tercera encuentra la palabra con mayor número de caracteres.

```
1
2 # Palabra con m s vocales
3 PalabraConMasVocales = max(SinArticulos, key=lambda x: sum(1 for c in x if c in "aeiou
      "))
4
5 # Cadena sin art culos ni preposiciones
6 Cadena = " ".join(SinArticulos)
7 FrecuenciaLetras = {letra: Cadena.count(letra) for letra in set(Cadena) if letra.
      isalpha()}
8
9 # Palabras con m s frecuencia
10 PalabrasMasComunes = {palabra: SinArticulos.count(palabra) for palabra in set(
      SinArticulos)}
```

Listing 3: Parte de analisis

Las cuarta funcione recorre cada palabra y cuenta cuántas vocales contiene, seleccionando la que tenga más. La quinta función une todas las palabras en una sola cadena y genera un diccionario que indica cuántas veces aparece cada letra (ignorando espacios y símbolos). La sexta función Crea un diccionario donde cada palabra se asocia con la cantidad de veces que aparece en el texto.

```
1
2 #Palabras con m s densidad (longitud/frecuencia)
3 PalabrasConMasDensidad = {palabra: len(palabra) / SinArticulos.count(palabra) for
      palabra in set(SinArticulos)}
4
```

```
#Longitud promedio de palabras
LongitudPromedioPalabras = sum(len(p) for p in SinArticulos) / len(SinArticulos) if
    SinArticulos else 0
```

Listing 4: Parte de analisis

La septima función calcula un índice de "densidad" para cada palabra, definido como la longitud de la palabra dividida por su frecuencia de aparición. La octava función calcula el promedio de caracteres por palabra en el texto.

1.1.3. Impresión de los resultados

```

# Numero de palabras
print(f"N mero de palabras: {NumeroDePalabras}")

# Numero de palabras  nicas
print(f"N mero de palabras unicas: {NumeroDePalabrasUnicas}")

# Palabra m s larga
print(f"Palabra m s larga: {PalabraMasLarga}")

# Palabra con m s vocales
print(f"Palabra con m s vocales: {PalabraConMasVocales}")

# Frecuencia de letras
print(f"Frecuencia de letras: {FrecuenciaLetras}")

# Ordenar palabras por mayor frecuencia, solo mostrar las 5 mas altas
sorted_by_value_asc = dict(sorted(PalabrasMasComunes.items(), key=lambda item: item
    [1], reverse=True))
print(f"Palabras m s comunes: {list(sorted_by_value_asc.items())[:5]}")

# Ordenar palabras por mayor densidad, solo mostrar las 5 mas altas
sorted_by_value_asc2 = dict(sorted(PalabrasConMasDensidad.items(), key=lambda item:
    item[1], reverse=True))
print(f"Palabras con m s densidad: {list(sorted_by_value_asc2.items())[:5]}")

# Longitud promedio de palabras
print(f"Longitud promedio de palabras: {LongitudPromedioPalabras}")
```

Listing 5: Impresion de resultados

En esta sección se muestran los resultados del análisis realizado al texto. Se imprimen datos clave como el número total y único de palabras, la palabra más larga y la que contiene más vocales, así como la frecuencia de cada letra en el texto. También se destacan las cinco palabras más comunes y las cinco con mayor densidad (relación entre longitud y frecuencia), además de calcular la longitud promedio de las palabras. Estas salidas representan la síntesis del procesamiento y permiten interpretar de forma práctica las características lingüísticas del texto analizado.

Dentro del codigo en la lineas 13 y 22 se usan dos funciones lambdas. En la linea 13 el lambda recibe cada elemento del diccionario (item, que es un par (palabra, frecuencia)) y retorna la frecuencia (item[1]). Esa frecuencia se usa como criterio de ordenación en sorted(). En la linea 22 es similar al caso anterior, pero ahora el lambda toma como referencia la densidad de cada palabra (longitud / frecuencia).

1.2. Resultados

Como ejemplo de funcionamiento del codigo usamos como ejemplo el articulo de *La llegada de la inteligencia artificial a la educación* Obteniendo los siguientes resultados

```

N mero de palabras: 114
N mero de palabras  nicas : 88
Palabra m s larga: transversalidad
Palabra con m s vocales: alfabetizacion
Frecuencia de letras: {'a': 95, 's': 35, 'z': 4, 'n': 54, 'b': 9, 'l': 25, 'q': 4, 'm'
    : 24, 'd': 45, 'v': 14, 'c': 39, 'x': 3, 'u': 22, 'e': 88, 'i': 72, 'f': 7, 'j': 6,
     'y': 11, 'p': 14, 'g': 13, 'o': 48, 'h': 4, 'r': 52, 't': 44}
```

```
Palabras m s comunes: [('y', 10), ('verdadera', 4), ('que', 3), ('del', 2), ('cada', 2)]
Palabras con m s densidad: [('transversalidad', 15.0), ('alfabetizacion', 14.0), ('conocimientos', 13.0), ('competencias', 12.0), ('integralidad', 12.0)]
Longitud promedio de palabras: 6.421052631578948
```

Listing 6: Resultados

Los resultados obtenidos reflejan que el código realizó correctamente un análisis completo del texto siguiendo los pasos de preprocesamiento y métricas definidas. El total de palabras fue 114, de las cuales 88 eran únicas, lo que indica cierta repetición pero también diversidad léxica en el texto. La palabra más larga identificada fue "transversalidadz la que contiene más vocales fue .ªlfabetizacion", lo que demuestra que las funciones max() con criterios de longitud y conteo de vocales operaron correctamente. La frecuencia de letras muestra un patrón esperado de aparición, con predominio de las vocales 'a', 'e', 'i' y consonantes comunes como 'n', 'r' y 'd', evidenciando que el conteo se realizó ignorando mayúsculas y símbolos. Las palabras más comunes y aquellas con mayor densidad reflejan la relevancia de términos clave del artículo, como "verdadera", "transversalidad" y "alfabetizacion", mostrando que el código pudo calcular correctamente la relación entre longitud y frecuencia para determinar densidad. Finalmente, la longitud promedio de palabras de aproximadamente 6.42 caracteres confirma que el cálculo de métricas agregadas funcionó según lo esperado. En conjunto, estos resultados validan que el código analiza de manera efectiva el texto, limpiando símbolos y stopwords, y generando estadísticas precisas y útiles para estudios lingüísticos o de procesamiento de lenguaje natural.

2. Pregunta 2

Crea un sistema de gestión de estudiantes utilizando tuplas para representar estudiantes (nombre, edad, carrera), sets para almacenar carreras únicas (al menos cinco carreras), diccionarios para calificaciones por estudiante, y funciones lambda para operaciones. Implementa funciones para: agregar estudiantes, asignar calificaciones, obtener promedios, filtrar estudiantes por carrera, encontrar los top N estudiantes por promedio y los N peores promedios.

Resumen

Este proyecto presenta un sistema de gestión de estudiantes implementado en Python, que utiliza tuplas para representar la información básica de cada alumno (nombre, edad y carrera), un conjunto para almacenar cinco carreras únicas y diccionarios para registrar sus calificaciones. El sistema incorpora funciones lambda para calcular promedios de manera eficiente y funciones específicas para la administración académica: agregar estudiantes, asignar calificaciones, obtener promedios individuales y generales, filtrar estudiantes por carrera, así como identificar a los mejores y peores estudiantes según su promedio. De esta manera, se ofrece una herramienta práctica y modular que permite organizar y analizar datos académicos de manera sencilla, facilitando la evaluación comparativa del rendimiento estudiantil.

2.1. Análisis del código

Para el analisis de codigo se usa el siguiente repositorio: https://github.com/Lhao13

2.1.1. Definición de variables

```python
#5 carreras unicas
carreras = {"Ingenieria", "Medicina", "Arquitectura", "Derecho", "Psicologia"}

#Tuplas de estudiantes (nombre, edad, carrera)
estudiantes = []

#Diccionario de calificaciones
calificaciones = {}

# Funci n para calcular el promedio
promedio = lambda notas: round(sum(notas) / len(notas), 2) if notas else 0
```

Listing 7: Definicion de variables

En la deficion de variables en primer lugar se define un set para almacenar las cinco carreras disponibles. Esto garantiza que cada carrera sea única y permite verificaciones rápidas de existencia al agregar estudiantes. En segunda instancia se crea una lista vacía que contendrá tuplas de estudiantes, donde cada tupla tiene la estructura (nombre, edad, carrera). Esta lista permitirá almacenar y recorrer todos los estudiantes registrados. En tercera instancia se define un diccionario vacío donde la clave es el nombre del estudiante y el valor es una lista de calificaciones asociadas a ese estudiante. Esto facilita agregar, consultar y calcular promedios de forma rápida. Finalmente se define una función anónima (lambda) que recibe una lista de notas y devuelve su promedio, redondeado a dos decimales. Si la lista está vacía, retorna 0.

2.1.2. Creación de funciones

```
# Funciones para manejar estudiantes y calificaciones
def agregar_estudiante(nombre, edad, carrera):
    if carrera in carreras:
        estudiantes.append((nombre, edad, carrera))
        calificaciones[nombre] = []
        print(f"Estudiante {nombre} con {edad} a os agregado a la carrera de {carrera}.")
    else:
        print(f"La carrera {carrera} no est  disponible.")

# Funci n para agregar calificaciones
def agregar_calificacion(nombre, calificacion):
    if nombre in calificaciones:
        calificaciones[nombre].append(calificacion)
        print(f"Calificaci n {calificacion} agregada para {nombre}.")
    else:
        print(f"Estudiante {nombre} no encontrado.")

# Funci n para obtener el promedio de calificaciones de un estudiante
def promedio_calificaciones(nombre):
    return {nombre: promedio(calificaciones[nombre])} if nombre in calificaciones else None
```

Listing 8: Analisis de funciones

La primera función Verifica que la carrera exista en el conjunto *carreras*, Si es válida, crea una tupla (nombre, edad, carrera) y la agrega a la lista *estudiantes*. Inicializa una entrada vacía en el diccionario *calificacione*s para ese estudiante, lista para recibir notas. Si la carrera no existe, informa que no está disponible.

La segunda función, comprueba que el estudiante exista en el diccionario *calificaciones*. Si existe, agrega la calificación a la lista correspondiente. Si no, informa que el estudiante no se encuentra registrado.

La tercera función verifica que el estudiante exista en *calificaciones*. Calcula su promedio usando la función lambda definida anteriormente (promedio). Devuelve un diccionario con el nombre del estudiante y su promedio, o *None* si no se encuentra.

```
# Funci n para obtener el promedio de calificaciones de todos los estudiantes
def promedio_calificaciones_general():
    return {nombre: promedio(notas) for nombre, notas in calificaciones.items()}

# Funci n para obtener estudiantes por carrera
def obtener_estudiantes_por_carrera(carrera):
    return [estU for estU in estudiantes if estU[2] == carrera]

# Funci n para obtener los mejores  estudiantes por promedio
def top_estudiantes_por_promedio(n=3):
    promediostemp  = {nombre: promedio(notas) for nombre, notas in calificaciones.items()}
    return sorted(promediostemp.items(), key=lambda item: item[1], reverse=True)[:n]

# Funci n para obtener los peores estudiantes por promedio
def peores_estudiantes_por_promedio(n=3):
    promediostemp  = {nombre: promedio(notas) for nombre, notas in calificaciones.items()}
```

```
    return sorted(promediostemp.items(), key=lambda item: item[1])[:n]
```

Listing 9: Analisis de funciones

Cuarta función recorre el diccionario *calificaciones*.Calcula el promedio de cada estudiante usando la función lambda (promedio). Devuelve un diccionario con cada estudiante y su promedio.

La Quinta función filtra la lista de tuplas estudiantes seleccionando solo aquellos cuya carrera coincida con la solicitada.

La Sexta función calcula los promedios de todos los estudiantes. Ordena los pares (nombre, promedio) de mayor a menor usando lambda como criterio de ordenamiento. Retorna los n mejores estudiantes, se puede cambiar este input para devolver mas estudiantes.

La séptima función es similar a la sexta, pero ordena de menor a mayor para obtener los estudiantes con los promedios más bajos.

2.1.3. Uso de las funciones

```
if __name__ == "__main__":
    agregar_estudiante("Ana", 20, "Ingenieria")
    agregar_estudiante("Luis", 22, "Medicina")
    agregar_estudiante("Marta", 19, "Arquitectura")
    agregar_estudiante("Carlos", 21, "Derecho")
    agregar_estudiante("Sofia", 23, "Psicologia")
    agregar_estudiante("Pedro", 24, "Medicina")

    agregar_calificacion("Ana", 95)
    agregar_calificacion("Ana", 92)
    agregar_calificacion("Luis", 88)
    agregar_calificacion("Luis", 90)
    agregar_calificacion("Marta", 85)
    agregar_calificacion("Carlos", 78)
    agregar_calificacion("Sofia", 80)
    agregar_calificacion("Pedro", 70)
    agregar_calificacion("Pedro", 75)
    agregar_calificacion("Pedro", 99)

    print("Promedios de calificaciones de Ana:", promedio_calificaciones("Ana"))

    print("Promedios generales de calificaciones:", promedio_calificaciones_general())

    print("Estudiantes en Medicina:", obtener_estudiantes_por_carrera("Medicina"))

    print("Top 3 estudiantes por promedio:", top_estudiantes_por_promedio(3))

    print("Peores 3 estudiantes por promedio:", peores_estudiantes_por_promedio(3))
```

Listing 10: Ejemplo de uso

En este ejemplo se agregan seis *estudiantes* a la lista estudiantes usando la función *agregar_estudiante*. Cada estudiante se vincula automáticamente con su entrada vacía en el diccionario *calificaciones*.

A continuacion se Se asignan varias notas a cada estudiante usando la función *agregar_calificacion*. Cada nota se agrega a la lista correspondiente del estudiante en *calificaciones*.

Finalmente se imprime el promedio de un estudiante específico (Ana) con *promedio_calificaciones*. Se obtiene y muestra el promedio general de todos los estudiantes con *promedio_calificaciones_general*. Se filtran los estudiantes de una carrera específica (Medicina) con *obtener_estudiantes_por_carrera*. Se identifican los top 3 estudiantes y los peores 3 estudiantes según promedio usando *top_estudiantes_por_promedio* y *peores_estudiantes_por_promedio*.

2.2. Resultados

Como ejemplo de funcionamiento del código ingresamos en el *main* los distintos estudiantes y obtenemos la siguiente impresion

```

Estudiante Ana con 20 a os agregado a la carrera de Ingenieria.
Estudiante Luis con 22 a os agregado a la carrera de Medicina.
Estudiante Marta con 19 a os agregado a la carrera de Arquitectura.
```

```
Estudiante Carlos con 21 a os agregado a la carrera de Derecho.
Estudiante Sofia con 23 a os agregado a la carrera de Psicologia.
Estudiante Pedro con 24 a os agregado a la carrera de Medicina.
Calificaci n 95 agregada para Ana.
Calificaci n 92 agregada para Ana.
Calificaci n 88 agregada para Luis.
Calificaci n 90 agregada para Luis.
Calificaci n 85 agregada para Marta.
Calificaci n 78 agregada para Carlos.
Calificaci n 80 agregada para Sofia.
Calificaci n 70 agregada para Pedro.
Calificaci n 75 agregada para Pedro.
Calificaci n 99 agregada para Pedro.
Promedios de calificaciones de Ana: {'Ana': 93.5}
Promedios generales de calificaciones: {'Ana': 93.5, 'Luis': 89.0, 'Marta': 85.0, '
    Carlos': 78.0, 'Sofia': 80.0, 'Pedro': 81.33}
Estudiantes en Medicina: [('Luis', 22, 'Medicina'), ('Pedro', 24, 'Medicina')]
Top 3 estudiantes por promedio: [('Ana', 93.5), ('Luis', 89.0), ('Marta', 85.0)]
Peores 3 estudiantes por promedio: [('Carlos', 78.0), ('Sofia', 80.0), ('Pedro',
    81.33)]
```

Listing 11: Resultados

Los resultados muestran que el sistema de gestión de estudiantes funcionó correctamente, cumpliendo con todas sus funcionalidades principales. Se confirma que los estudiantes fueron agregados exitosamente a la lista de tuplas *estudiantes* y que sus calificaciones se registraron de manera adecuada en el diccionario *calificaciones*, permitiendo cálculos precisos de promedios. El promedio individual de Ana y los promedios generales reflejan correctamente la media de las notas de cada estudiante, evidenciando que la función lambda para cálculo de promedios se aplicó de manera correcta. La función de filtrado por carrera identificó correctamente a los estudiantes inscritos en Medicina, mostrando que la comprensión de listas para segmentación por carrera funciona como se esperaba. Asimismo, los rankings de desempeño evidencian que las funciones *top_estudiantes_por_promedio* y *peores_estudiantes_por_promedio* ordenan correctamente a los alumnos de acuerdo con sus promedios, usando lambda para el criterio de ordenamiento. En conjunto, estos resultados validan que el código implementa de manera efectiva un sistema modular y confiable para registrar, consultar y analizar el rendimiento académico de los estudiantes.

3. Conclución

El análisis integral de los dos proyectos demuestra la efectividad del uso de estructuras de datos y funciones en Python para el procesamiento y gestión de información. En el primer proyecto, el código de análisis de texto permitió limpiar, procesar y extraer métricas lingüísticas clave, como conteo de palabras, frecuencia de letras, palabras más comunes y densidad, mostrando la utilidad de técnicas de preprocesamiento y funciones lambda para generar estadísticas precisas y útiles. En el segundo proyecto, el sistema de gestión de estudiantes evidenció la correcta implementación de estructuras como tuplas, sets y diccionarios para almacenar información de estudiantes y calificaciones, así como el uso de funciones y lambda para calcular promedios, filtrar por carrera y generar rankings de desempeño. En conjunto, ambos trabajos reflejan cómo Python permite construir sistemas modulares, eficientes y fáciles de mantener para distintos tipos de datos, combinando manipulación, análisis y presentación de información de manera clara, confiable y escalable.

3.1. How to add Citations and a References List