Imports:

```
import psycopg2
import pandas as pd
import nltk
import json
import re
import warnings
import time

nltk.download("punkt")
nltk.download("punkt_tab")
nltk.download("wordnet")
nltk.download("omw-1.4")
stemmer = nltk.stem.SnowballStemmer("spanish")
lemmatizer = nltk.stem.WordNetLemmatizer()
```

Omitir advertencia de pandas sobre SQLAlchemy

```
warnings.filterwarnings(
    "ignore",
    message="pandas only supports SQLAlchemy connectable",
    category=UserWarning,
)
```

1. Configuración en PostgreSQL y carga de datos

- Creación de tablas:

```
CREATE TABLE noticias
(
   id           SERIAL PRIMARY KEY,
   url          TEXT,
   contenido    TEXT,
   categoria    VARCHAR(50),
   bag_of_words JSONB
);

CREATE TABLE stopwords
(
   id   SERIAL PRIMARY KEY,
   word TEXT UNIQUE NOT NULL
);
```

- Carga de datos:

```
SELECT *
FROM noticias;
```

id	url	contenido	categoria
1	url	contenido	categoria
2	https://www.larepublica.co/redirect/post/3201905	Durante el foro La banca articulador empresarial para…	Otra
3	https://www.larepublica.co/redirect/post/3210288	El regulador de valores de China dijo el domingo que …	Regulaciones
4	https://www.larepublica.co/redirect/post/3240676	En una industria históricamente masculina como lo es …	Alianzas
5	https://www.larepublica.co/redirect/post/3342889	Con el dato de marzo el IPC interanual encadena su de…	Macroeconomia

```sql
SELECT *
FROM stopwords;
```

id	word
1	a
2	actualmente
3	adelante
4	además
5	afirmó

- Leer desde PostgreSQL con Python

  Primero, instalamos los paquetes necesarios y recursos de ntlk, creamos las tablas (noticias y stopwords), e importamos los datos a las tablas creadas. Luego, probamos que la lectura de datos desde Python funcionara correctamente:

```python
def connect_db():
    conn = psycopg2.connect(
        dbname="Lab5.1",
        user="postgres",
        password="postgres",
        host="localhost",
        port="5432",
    )
    return conn

def fetch_data():
    conn = connect_db()
    query = "SELECT id, contenido FROM noticias;"
    df = pd.read_sql(query, conn)
    conn.close()
    return df

def fetch_stopwords():
    conn = connect_db()
    query = "SELECT word FROM stopwords;"
    df = pd.read_sql(query, conn)
    conn.close()
    stopword_list = df["word"].tolist()
    return stopword_list
```

```
noticias_df = fetch_data()
stopwords = fetch_stopwords()
print(noticias_df)
print(stopwords)
```

```
        id                                          contenido
0        1                                          contenido
1        5  Con el dato de marzo el IPC interanual encaden...
2       26  Desde que empezara a finales de febrero la inv...
3       66  La Medicina se encuentra en un punto de transf...
4       87  Para impulsar la descarbonización del planeta,...
...    ...                                                ...
1213  1209  La política monetaria la dirige el Banco Repúb...
1214  1210  Sorprendió el silencio de los candidatos presi...
1215  1211  La Asociación de Transporte Aéreo Internaciona...
1216  1215  La espiral alcista de los precios continúa y g...
1217  1216  Las grandes derrotas nacionales son experienci...

[1218 rows x 2 columns]
['a', 'actualmente', 'adelante', 'además', 'afirmó', 'agregó', 'ahora', 'ahí',
, 'algún', 'alrededor', 'ambos', 'ampleamos', 'ante', 'anterior', 'antes', 'ape
s', 'aqui', 'aquí', 'arriba', 'aseguró', 'así', 'atras', 'aunque', 'ayer', 'aña
a', 'buenas', 'bueno', 'buenos', 'cada', 'casi', 'cerca', 'cierta', 'ciertas',
, 'conocer', 'conseguimos', 'conseguir', 'considera', 'consideró', 'consigo', '
, 'creo', 'cual', 'cuales', 'cualquier', 'cuando', 'cuanto', 'cuatro', 'cuenta'
eben', 'debido', 'decir', 'dejó', 'del', 'demás', 'dentro', 'desde', 'después',
erentes', 'dijeron', 'dijo', 'dio', 'donde', 'dos', 'durante', 'e', 'ejemplo',
```

2. Preprocesamiento de texto

La función preprocess() debe recibir un texto, convertirlo a minúsculas, tokenizarlo, eliminar las *stopwords* (importadas previamente a la tabla stopwords), y hacer *stemming* (reducir las palabras a su raíz).

```
def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^a-záéíóúñü\s]', '', text)
    tokens = nltk.word_tokenize(text, "spanish")
    filtered = [token for token in tokens if token not in stopwords]
    stem = [stemmer.stem(w) for w in filtered]
    return stem
```

- Calcular frecuencia de términos

La función compute_bow() debe recibir un texto y calcular la *bag of words*. Nuestra implementación preprocesa el texto con preprocess() y cuenta la frecuencia de las palabras. El resultado final es un diccionario de forma [palabra, frecuencia], el cual puede ser convertido a JSON para ser guardado en la base de datos.

```
def compute_bow(text):
    processed_text = preprocess(text)
    bow = dict()
    for word in processed_text:
        if word in bow:
            bow[word] += 1
        else:
            bow[word] = 1
    return bow
```

3. Actualizar la base de datos con los Bags of Words

- Actualización de BOW en BD

Para actualizar la columna bag_of_words en la base de datos, iteramos por todas las noticias del *dataframe*, construimos las *bag of words*, y actualizamos la tabla.

```
def update_bow_in_db(dataframe):
    conn = connect_db()
    cursor = conn.cursor()
    for index, row in dataframe.iterrows():
        bow = compute_bow(row["contenido"])
        query = "UPDATE noticias SET bag_of_words = %s WHERE id = %s;"
        cursor.execute(query, (json.dumps(bow), row["id"]))
    conn.commit()
    cursor.close()
    conn.close()
```

4. Consultas booleanas

Para poder realizar consultas desde Python, implementamos un *parser* secuencial simple en la función apply_boolean_query(). Intercalamos el parsing de *keywords* y operadores booleanos, y producimos una *query* final que utiliza las raíces de las palabras de la consulta original. También añadir detección de errores para facilitar el *debugging*.

```
def apply_boolean_query(query, table):
    tokens = query.split()  # list containing query arguments

    # simple sequential parser:
    expects_keyword = True
    final_query = f"SELECT * FROM {table} WHERE bag_of_words ? "
```

```
    for w in tokens:
        if expects_keyword:
            stemmed = stemmer.stem(w.lower())
            final_query += "'" + stemmed + "'"
            expects_keyword = False
        else:
            w = w.lower()
            if w == "or":
                final_query += " OR bag_of_words ? "
            elif w == "and":
                final_query += " AND bag_of_words ? "
            elif w == "and-not":
                final_query += " AND NOT bag_of_words ? "
            else:
                print("ERROR: INVALID QUERY, UNRECOGNIZED OPERATOR " + w)
                return pd.DataFrame()
            expects_keyword = True

    if expects_keyword is True:
        print("ERROR: EXPECTED KEYWORD AT END")

    final_query += ";"

    # actual query time
    conn = connect_db()
    df = pd.read_sql(final_query, conn)
    conn.close()

    return df
```

Gramática utilizada:
S -> S A B | keyword
B -> S | keyword
A -> OR | AND | AND-NOT

- Pruebas funcionales

Para probar el correcto funcionamiento del *parser*, hicimos 10 pruebas de consulta, combinando *keywords* y operadores para crear *queries* más complejas.

```
def test():
    test_queries = [
        "ingeniería OR software AND desarrollo",
        "inteligencia AND artificial AND-NOT humano",
        "ciencia OR tecnología AND-NOT medicina",
        "educación AND aprendizaje OR enseñanza",
```

```
        "computción AND matemática AND-NOT física",
        "derecho OR leyes AND justicia",
        "historia OR geografía AND-NOT política",
        "arte OR cultura AND-NOT entretenimiento",
        "musica AND danza OR teatro",
        "salud AND bienestar OR medicina",
    ]

    tables = ["noticias", "noticias600", "noticias300", "noticias150"]
    results = []
    time_totals = {
        "noticias": [],
        "noticias150": [],
        "noticias300": [],
        "noticias600": []
    }

    for table in tables:
        for query in test_queries:
            start = time.time()
            df = apply_boolean_query(query, table)
            end = time.time()
            elapsed_ms = (end - start) * 1000  # Convertir a milisegundos
            time_totals[table].append(elapsed_ms)
            results.append(
                {"tabla": table, "query": query, "tiempo_ms": elapsed_ms,
"resultados": len(df)}
            )
            print(f"{table} | {query} | {elapsed_ms:.2f} ms | {len(df)}
resultados")

    print("\nPromedio de tiempo por tabla:")
    for table in tables:
        times = time_totals[table]
        avg = sum(times) / len(times) if times else 0
        print(f"{table}: {avg:.2f} ms")

    with open("resultados.json", "w", encoding="utf-8") as f:
        json.dump(results, f, ensure_ascii=False, indent=4)

    print("Resultados guardados en resultados.csv")
```

5. Actividad final

- Lematización:

Investigamos con respecto a la lematización de palabras con la librería mtlk para python, aunque ciertamente esto es soportado utilizando el WorldNet, solo se encuentra disponible para el inglés, por lo que no se podría aplicar a textos en español y esperar que funcione correctamente ya que las reglas de ambos lenguajes son incompatibles entre sí. Por este motivo, decidimos omitir este ítem.

- Tiempo de ejecución de los tests:

  A continuación se muestra el tiempo de ejecución promedio de las 10 consultas para cada tamaño de tabla planteado.

Total	600	300	150
107.31	65.88	63.25	92.32

- Exportación de resultados:

  Se implementó como parte del testing la función de exportar los resultados de las pruebas a una archivo JSON con el formato: Tabla, Query, Tiempo en ms, Cantidad de resultados. El código se puede ver en el item anterior.

```
noticias_df = fetch_data()
stopwords = fetch_stopwords()
#print(noticias_df)
#print(stopwords)
update_bow_in_db(noticias_df)
test()
```

Ejecución:

```
noticias300 | educación AND aprendizaje OR enseñanza | 60.12 ms | 8 resultados
noticias300 | computción AND matemática AND-NOT física | 58.29 ms | 0 resultados
noticias300 | derecho OR leyes AND justicia | 63.75 ms | 20 resultados
noticias300 | historia OR geografía AND-NOT política | 68.93 ms | 31 resultados
noticias300 | arte OR cultura AND-NOT entretenimiento | 64.25 ms | 15 resultados
noticias300 | musica AND danza OR teatro | 57.16 ms | 3 resultados
noticias300 | salud AND bienestar OR medicina | 62.19 ms | 11 resultados
noticias150 | ingeniería OR software AND desarrollo | 154.86 ms | 9 resultados
noticias150 | inteligencia AND artificial AND-NOT humano | 174.37 ms | 6 resultados
noticias150 | ciencia OR tecnología AND-NOT medicina | 67.84 ms | 42 resultados
noticias150 | educación AND aprendizaje OR enseñanza | 56.45 ms | 1 resultados
noticias150 | computción AND matemática AND-NOT física | 155.87 ms | 0 resultados
noticias150 | derecho OR leyes AND justicia | 76.98 ms | 9 resultados
noticias150 | historia OR geografía AND-NOT política | 75.83 ms | 18 resultados
noticias150 | arte OR cultura AND-NOT entretenimiento | 59.11 ms | 6 resultados
noticias150 | musica AND danza OR teatro | 57.47 ms | 2 resultados
noticias150 | salud AND bienestar OR medicina | 58.58 ms | 6 resultados

Promedio de tiempo por tabla:
noticias: 73.91 ms
noticias600: 84.54 ms
noticias300: 63.37 ms
noticias150: 93.74 ms
Resultados guardados en resultados.csv
```

Archivo JSON resultante:


```
[
    {
        "tabla": "noticias",
        "query": "ingeniería OR software AND desarrollo",
        "tiempo_ms": 80.47151565551758,
        "resultados": 51
    },
    {
        "tabla": "noticias",
        "query": "inteligencia AND artificial AND-NOT humano",
        "tiempo_ms": 70.97578048706055,
        "resultados": 30
    },
    {
        "tabla": "noticias",
        "query": "ciencia OR tecnología AND-NOT medicina",
        "tiempo_ms": 111.96470260620117,
        "resultados": 341
    },
    {
        "tabla": "noticias",
        "query": "educación AND aprendizaje OR enseñanza",
        "tiempo_ms": 70.66941261291504,
        "resultados": 24
    },
```