```
# download nltk stopwords
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
# Install a particular version of `google-cloud-storage` because (oddly enough)
# the  version on Colab and GCP is old. A dependency error below is okay.
!pip install -q google-cloud-storage==1.43.0
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 106.6/106.6 kB 3.6 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is
google-adk 1.21.0 requires google-cloud-storage<4.0.0,>=2.18.0, but you have google-cloud-storage 1.43.0 which is incompatib
bigframes 2.30.0 requires google-cloud-storage>=2.0.0, but you have google-cloud-storage 1.43.0 which is incompatible.
```

```
# authenticate below for Google Storage access as needed
from google.colab import auth
auth.authenticate_user()
```

```
# Create the local directory
!mkdir -p PKL

# Copy all .pkl files from your GCP bucket to the local PKL folder
!gsutil -m cp gs://badashbucket1/PKL/*.pkl ./PKL/
```

```
Copying gs://badashbucket1/PKL/postings_gcp_index.pkl...
Copying gs://badashbucket1/PKL/anchor_index.pkl...
Copying gs://badashbucket1/PKL/pageviews.pkl...
Copying gs://badashbucket1/PKL/doc_titles.pkl...
Copying gs://badashbucket1/PKL/pagerank.pkl...
Copying gs://badashbucket1/PKL/titles_index.pkl...
==> NOTE: You are downloading one or more large file(s), which would
run significantly faster if you enabled sliced object downloads. This
feature is enabled by default but requires that compiled crcmod be
installed (see "gsutil help crcmod").

\ [6/6 files][330.7 MiB/330.7 MiB] 100% Done  26.2 MiB/s ETA 00:00:00
Operation completed over 6 objects/330.7 MiB.
```

```
# Create the local directory structure
!mkdir -p index_files/bin_files

# Copy all .bin files from your GCP bucket to the local folder
!gsutil -m cp gs://badashbucket1/index_files/bin_files/*.bin ./index_files/bin_files/
```

```
Copying gs://badashbucket1/index_files/bin_files/117_008.bin
```

```
Copying gs://badashbucket1/index_files/bin_files/117_009.bin...
Copying gs://badashbucket1/index_files/bin_files/117_010.bin...
Copying gs://badashbucket1/index_files/bin_files/117_011.bin...
Copying gs://badashbucket1/index_files/bin_files/117_012.bin...
Copying gs://badashbucket1/index_files/bin_files/117_013.bin...
Copying gs://badashbucket1/index_files/bin_files/117_014.bin...
Copying gs://badashbucket1/index_files/bin_files/117_015.bin...
Copying gs://badashbucket1/index_files/bin_files/117_016.bin...
Copying gs://badashbucket1/index_files/bin_files/117_017.bin...
Copying gs://badashbucket1/index_files/bin_files/117_018.bin...
Copying gs://badashbucket1/index_files/bin_files/117_019.bin...
Copying gs://badashbucket1/index_files/bin_files/117_020.bin...
Copying gs://badashbucket1/index_files/bin_files/117_021.bin...
Copying gs://badashbucket1/index_files/bin_files/117_022.bin...
Copying gs://badashbucket1/index_files/bin_files/117_023.bin...
Copying gs://badashbucket1/index_files/bin_files/117_024.bin...
Copying gs://badashbucket1/index_files/bin_files/117_025.bin...
Copying gs://badashbucket1/index_files/bin_files/118_000.bin...
Copying gs://badashbucket1/index_files/bin_files/118_001.bin...
Copying gs://badashbucket1/index_files/bin_files/118_002.bin...
Copying gs://badashbucket1/index_files/bin_files/118_003.bin...
Copying gs://badashbucket1/index_files/bin_files/118_004.bin...
```

Run the app

```
# you need to upload your implementation of search_app.py
import search_frontend as se
```

```
# uncomment the code below and execute to reload the module when you make
# changes to search_frontend.py (after you upload again).
import importlib
importlib.reload(se)
```

```
Starting to load data from GCS bucket: badashbucket1
Loading index from PKL/postings_gcp_index.pkl ...
Index loaded successfully
Loading titles index from PKL/titles_index.pkl ...
Titles index loaded successfully
Loading anchor index from PKL/anchor_index.pkl ...
Anchor index loaded successfully
Loading doc titles from PKL/doc_titles.pkl ...
Doc titles loaded successfully. Total documents: 6348910
Loading pagerank from PKL/pagerank.pkl ...
Pagerank loaded successfully
Loading pageviews from PKL/pageviews.pkl ...
Pageviews loaded successfully
All data loaded successfully from GCS!

--- Index Debug Info ---
Body index attributes: ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__g
Body index has 'posting_locs': True
Body index has 'posting_lists': False
Sample terms in body index: ['warhawks', 'knowingly', 'morfa']
  'warhawks' posting_locs: [('6_015.bin', 1623042)]
  'knowingly' posting_locs: [('45_000.bin', 1189140)]
  'morfa' posting_locs: [('70_007.bin', 1298262)]
--- End Debug Info ---

<module 'search_frontend' from '/content/search_frontend.py'>
```

```
# find Colab's public URL
from google.colab.output import eval_js
server_url = eval_js("google.colab.kernel.proxyPort(5000)")
print(f"""Test your search engine by navigating to
{server_url}/search?query=hello+world
This URL is only accessible from the same browser session. In other words, this
will not be accessible from a different machine, browser, or incognito session.
""")

# Uncomment the following line of code to run the frontend in the main
# process and wait for HTTP requests (colab will hang). The debug parameter
# lets you see incoming requests and get debug print outs if exceptions occur.
#se.run(debug=False, use_reloader=False)

# # Alternatively, the next few lines run the frontend in a background process.
# # Just don't forget to terminate the process when you update your search engine
# # or want to reload it.
import multiprocessing
import time
proc = multiprocessing.Process(target=se.run,
                               kwargs={"debug": True, "use_reloader": False,
                                       "host": "0.0.0.0", "port": 5000})
proc.start()
```

```
proc.start()

time.sleep(1) # give Flask time to boot

from google.colab.output import eval_js
server_url = eval_js("google.colab.kernel.proxyPort(5000)")

print(f"Open this URL:\n{server_url}/search?query=hello+world")
# Use proc.terminate() to stop the process
```

```
Test your search engine by navigating to
https://5000-m-s-2cj19nm4gjlu1-c.us-west1-2.prod.colab.dev/search?query=hello+world
This URL is only accessible from the same browser session. In other words, this
will not be accessible from a different machine, browser, or incognito session.

 * Serving Flask app 'search_frontend'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.28.0.12:5000
INFO:werkzeug:Press CTRL+C to quit
Open this URL:
https://5000-m-s-2cj19nm4gjlu1-c.us-west1-2.prod.colab.dev/search?query=hello+world
```

Testing your app

Once your app is running you can query it. You can simply do that by clicking on the URL printed above (the one looking like https://XXXXX-5000-colab.googleusercontent.com/search?query=hello+world or by issuing an HTTP request through code (from colab).

The code below shows how to issue a query from python. This is also how our testing code will issue queries to your search engine, so make sure to test your search engine this way after you deploy it to GCP and before submission. Command line instructions for deploying your search engine to GCP are available at `run_frontend_in_gcp.sh`. Note that we will not only issue training queries to your search engine, but also test queries, i.e. queries that you've never seen before.

```
import json

with open('queries_train.json', 'rt') as f:
  queries = json.load(f)
```

```
def average_precision(true_list, predicted_list, k=40):
    true_set = frozenset(true_list)
    predicted_list = predicted_list[:k]
    precisions = []
    for i,doc_id in enumerate(predicted_list):
        if doc_id in true_set:
            prec = (len(precisions)+1) / (i+1)
            precisions.append(prec)
    if len(precisions) == 0:
        return 0.0
    return round(sum(precisions)/len(precisions),3)
```

```
def precision_at_k(true_list, predicted_list, k):
    true_set = frozenset(true_list)
    predicted_list = predicted_list[:k]
    if len(predicted_list) == 0:
        return 0.0
    return round(len([1 for doc_id in predicted_list if doc_id in true_set]) / len(predicted_list), 3)
def recall_at_k(true_list, predicted_list, k):
    true_set = frozenset(true_list)
    predicted_list = predicted_list[:k]
    if len(true_set) < 1:
        return 1.0
    return round(len([1 for doc_id in predicted_list if doc_id in true_set]) / len(true_set), 3)
def f1_at_k(true_list, predicted_list, k):
    p = precision_at_k(true_list, predicted_list, k)
    r = recall_at_k(true_list, predicted_list, k)
    if p == 0.0 or r == 0.0:
        return 0.0
    return round(2.0 / (1.0/p + 1.0/r), 3)
def results_quality(true_list, predicted_list):
    p5 = precision_at_k(true_list, predicted_list, 5)
    f1_30 = f1_at_k(true_list, predicted_list, 30)
    if p5 == 0.0 or f1_30 == 0.0:
        return 0.0
    return round(2.0 / (1.0/p5 + 1.0/f1_30), 3)
```

```
assert precision_at_k(range(10), [1,2,3] , 2) == 1.0
assert recall_at_k(   range(10), [10,5,3], 2) == 0.1
assert precision_at_k(range(10), []      , 2) == 0.0
assert precision_at_k([],        [1,2,3],  5) == 0.0
assert recall_at_k(   [],        [10,5,3], 2) == 1.0
assert recall_at_k(   range(10), [],       2) == 0.0
assert f1_at_k(       [],        [1,2,3],  5) == 0.0
assert f1_at_k(       range(10), [],       2) == 0.0
assert f1_at_k(       range(10), [0,1,2],  2) == 0.333
assert f1_at_k(       range(50), range(5), 30) == 0.182
assert f1_at_k(       range(50), range(10), 30) == 0.333
assert f1_at_k(       range(50), range(30), 30) == 0.75
assert results_quality(range(50), range(5))  == 0.308
assert results_quality(range(50), range(10)) == 0.5
assert results_quality(range(50), range(30)) == 0.857
assert results_quality(range(50), [-1]*5 + list(range(5,30))) == 0.0
```

```
import requests
from time import time
# In GCP the public URL for your engine should look like this:
# url = 'http://35.232.59.3:8080'
# In colab, we are going to send HTTP requests to localhost (127.0.0.1)
# and direct them to port where the server is listening (5000).
url = 'http://127.0.0.1:5000'

qs_res = []
for q, true_wids in queries.items():
  duration, ap = None, None
  t_start = time()
  try:
    res = requests.get(url + '/search', {'query': q}, timeout=35)
    duration = time() - t_start
    if res.status_code == 200:
      pred_wids, _ = zip(*res.json())
      rq = results_quality(true_wids, pred_wids)
  except:
    pass

  qs_res.append((q, duration, rq))
```