Using APIs to help automate information retrieval for evidence identification

Harry Scells

Leipzig University

26.04.2024

Preliminaries

Want to follow along?

https://scells.me/irm24

Preliminaries

Want to follow along?

https://scells.me/irm24

This workshop will

- provide an introduction to APIs, and explain (briefly) how they work
- highlight use cases of APIs for automating evidence identification
- empower you to use APIs in your workflows

Preliminaries

Want to follow along?

https://scells.me/irm24

This workshop will

- provide an introduction to APIs, and explain (briefly) how they work
- highlight use cases of APIs for automating evidence identification
- empower you to use APIs in your workflows

This workshop will not

- provide an introduction to programming (but still gives you tools!)
- cover all the APIs that are out there (but covers the main ones!)
- go into deep technical details (but is a good starting point!)

What are APIs?

How to use APIs?

Use cases for APIs

Summary

Conceptual Introduction to APIs

Imagine you sit down at a restaurant…

Conceptual Introduction to APIs

Imagine you sit down at a restaurant…

The waiter takes your order…

Conceptual Introduction to APIs

Imagine you sit down at a restaurant…

The waiter takes your order…

Sends it to the kitchen…

Conceptual Introduction to APIs

Imagine you sit down at a restaurant…

The waiter takes your order…

Sends it to the kitchen…

Then brings your food back.

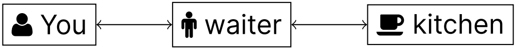
Conceptual Introduction to APIs

Imagine you sit down at a restaurant...

The waiter takes your order...

Sends it to the kitchen...

Then brings your food back.



You

waiter

kitchen

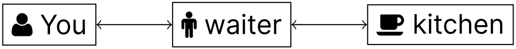
Conceptual Introduction to APIs

Imagine you sit down at a restaurant…

The waiter takes your order…

Sends it to the kitchen…

Then brings your food back.



The *waiter* is an *API*, and the *kitchen* is a *web server*

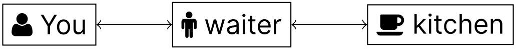
Conceptual Introduction to APIs

Imagine you sit down at a restaurant…

The waiter takes your order…

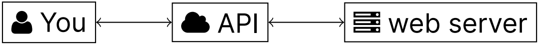
Sends it to the kitchen…

Then brings your food back.



You ⟷ waiter ⟷ kitchen

The *waiter* is an *API*, and the *kitchen* is a *web server*



You ⟷ API ⟷ web server

Conceptual Introduction to APIs

You can also make different *requests*

Conceptual Introduction to APIs

You can also make different *requests*

`GET` 🧍 order something from the menu
☁ retrieve data from the server

Conceptual Introduction to APIs

You can also make different *requests*

`GET` 🧍 order something from the menu
☁ retrieve data from the server

`POST` 🧍 add your own menu item to the restaurant
☁ send some data to the server

Conceptual Introduction to APIs

You can also make different *requests*

`GET` 🧍 order something from the menu
☁ retrieve data from the server

`POST` 🧍 add your own menu item to the restaurant
☁ send some data to the server

`PUT` 🧍 change your menu item in the restaurant
☁ update some data on the server

Conceptual Introduction to APIs

You can also make different *requests*

`GET`	🧍	order something from the menu
	☁	retrieve data from the server
`POST`	🧍	add your own menu item to the restaurant
	☁	send some data to the server
`PUT`	🧍	change your menu item in the restaurant
	☁	update some data on the server
`DELETE`	🧍	remove your menu item from the restaurant
	☁	remove some data on the server

Conceptual Introduction to APIs

You can also make different *requests*

`GET` 🧍 order something from the menu
☁ retrieve data from the server

`POST` 🧍 add your own menu item to the restaurant
☁ send some data to the server

`PUT` 🧍 change your menu item in the restaurant
☁ update some data on the server

`DELETE` 🧍 remove your menu item from the restaurant
☁ remove some data on the server

In this workshop, we will only deal with `GET` requests.

Limitations of APIs

APIs are powerful automation tools, but they have limitations

Limitations of APIs

APIs are powerful automation tools, but they have limitations

Accessibility

- Require payment or be completely private
- Rate limited, restricting the number of requests
- Restrictions on the types of data you can access

Limitations of APIs

APIs are powerful automation tools, but they have limitations

Accessibility

- Require payment or be completely private
- Rate limited, restricting the number of requests
- Restrictions on the types of data you can access

Privacy

- You are sending data over the internet when you use an API
- Be aware that this can be intercepted by third parties
- Whatever data you send will be stored by the server

Limitations of APIs

APIs are powerful automation tools, but they have limitations

Accessibility

- Require payment or be completely private
- Rate limited, restricting the number of requests
- Restrictions on the types of data you can access

Privacy

- You are sending data over the internet when you use an API
- Be aware that this can be intercepted by third parties
- Whatever data you send will be stored by the server

Complexity

- Getting started can have a steep learning curve
- Errors and unexpected behaviour can be difficult to troubleshoot
- Be persistent! Seek help if you get stuck!

Finding APIs

How to access an API in the first place can be difficult

Finding APIs

How to access an API in the first place can be difficult

In this workshop, we'll cover

- Entrez (PubMed)
- ClinicalTrials.gov
- MeshMate [1]

[1] Wang et al. 2022

Finding APIs

How to access an API in the first place can be difficult

In this workshop, we'll cover

- Entrez (PubMed)
- ClinicalTrials.gov
- MeshMate [1]

The online supplementary material has a more comprehensive list

[1] Wang et al. 2022

What are APIs?

How to use APIs?

Use cases for APIs

Summary

Data Formats

When you use an API, you normally get back machine-readable data

Data Formats

When you use an API, you normally get back machine-readable data

```
{
  'firstName': 'John',
  'lastName': 'Smith',
  'isAlive': true,
  'age': 27,
  'address': {
    'streetAddress': '21 2nd Street',
    'city': 'New York',
    'state': 'NY',
    'postalCode': '10021-3100'
  },
  'phoneNumbers': [
    {
      'type': 'home',
      'number': '212 555-1234'
    },
    {
      'type': 'office',
      'number': '646 555-4567'
    }
  ],
  'children': [],
  'spouse': null
}
```

Data Formats

When you use an API, you normally get back machine-readable data

```
{
  'firstName': 'John',
  'lastName': 'Smith',
  'isAlive': true,
  'age': 27,
  'address': {
    'streetAddress': '21 2nd Street',
    'city': 'New York',
    'state': 'NY',
    'postalCode': '10021-3100'
  },
  'phoneNumbers': [
    {
      'type': 'home',
      'number': '212 555-1234'
    },
    {
      'type': 'office',
      'number': '646 555-4567'
    }
  ],
  'children': [],
  'spouse': null
}
```

The online supplementary material has more data format examples

Using APIs via HTTPie

https://httpie.io/app

Using APIs via HTTPie

https://httpie.io/app

HTTPie is a tool for interacting with APIs

- Quickly test and debug API requests
- No coding required to use it
- Completely free

Using APIs via HTTPie

https://httpie.io/app

HTTPie is a tool for interacting with APIs

- Quickly test and debug API requests
- No coding required to use it
- Completely free

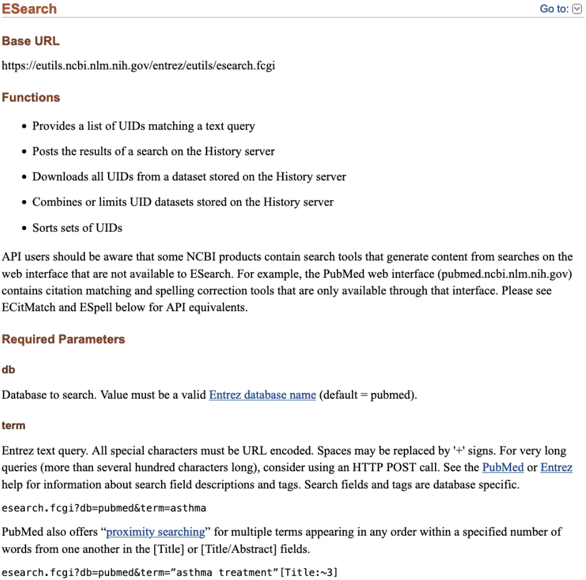
Let’s see an example of searching PubMed using HTTPie

Using APIs via HTTPie

https://www.ncbi.nlm.nih.gov/books/NBK25499/#chapter4.ESearch

Using APIs via HTTPie

https://www.ncbi.nlm.nih.gov/books/NBK25499/#chapter4.ESearch



ESearch

Go to:

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi

Functions

- Provides a list of UIDs matching a text query
- Posts the results of a search on the History server
- Downloads all UIDs from a dataset stored on the History server
- Combines or limits UID datasets stored on the History server
- Sorts sets of UIDs

API users should be aware that some NCBI products contain search tools that generate content from searches on the web interface that are not available to ESearch. For example, the PubMed web interface (pubmed.ncbi.nlm.nih.gov) contains citation matching and spelling correction tools that are only available through that interface. Please see ECitMatch and ESpell below for API equivalents.

Required Parameters

db

Database to search. Value must be a valid Entrez database name (default = pubmed).

term

Entrez text query. All special characters must be URL encoded. Spaces may be replaced by '+' signs. For very long queries (more than several hundred characters long), consider using an HTTP POST call. See the PubMed or Entrez help for information about search field descriptions and tags. Search fields and tags are database specific.

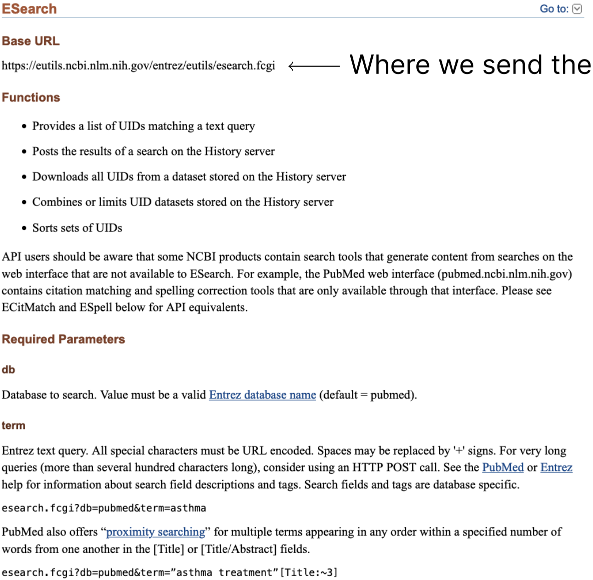
```
esearch.fcgi?db=pubmed&term=asthma
```

PubMed also offers "proximity searching" for multiple terms appearing in any order within a specified number of words from one another in the [Title] or [Title/Abstract] fields.

```
esearch.fcgi?db=pubmed&term="asthma treatment"[Title:~3]
```

Using APIs via HTTPie

https://www.ncbi.nlm.nih.gov/books/NBK25499/#chapter4.ESearch



ESearch

Go to: ⌄

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi ← Where we send the request (endpoint)

Functions

- Provides a list of UIDs matching a text query
- Posts the results of a search on the History server
- Downloads all UIDs from a dataset stored on the History server
- Combines or limits UID datasets stored on the History server
- Sorts sets of UIDs

API users should be aware that some NCBI products contain search tools that generate content from searches on the web interface that are not available to ESearch. For example, the PubMed web interface (pubmed.ncbi.nlm.nih.gov) contains citation matching and spelling correction tools that are only available through that interface. Please see ECitMatch and ESpell below for API equivalents.

Required Parameters

db

Database to search. Value must be a valid Entrez database name (default = pubmed).

term

Entrez text query. All special characters must be URL encoded. Spaces may be replaced by '+' signs. For very long queries (more than several hundred characters long), consider using an HTTP POST call. See the PubMed or Entrez help for information about search field descriptions and tags. Search fields and tags are database specific.

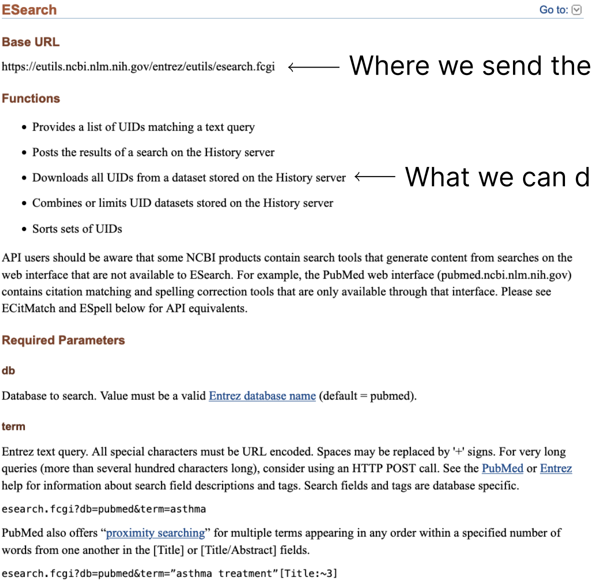
```
esearch.fcgi?db=pubmed&term=asthma
```

PubMed also offers "proximity searching" for multiple terms appearing in any order within a specified number of words from one another in the [Title] or [Title/Abstract] fields.

```
esearch.fcgi?db=pubmed&term="asthma treatment"[Title:~3]
```

Using APIs via HTTPie

https://www.ncbi.nlm.nih.gov/books/NBK25499/#chapter4.ESearch



ESearch

Go to:

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi ⟵ Where we send the request (endpoint)

Functions

- Provides a list of UIDs matching a text query
- Posts the results of a search on the History server
- Downloads all UIDs from a dataset stored on the History server ⟵ What we can do
- Combines or limits UID datasets stored on the History server
- Sorts sets of UIDs

API users should be aware that some NCBI products contain search tools that generate content from searches on the web interface that are not available to ESearch. For example, the PubMed web interface (pubmed.ncbi.nlm.nih.gov) contains citation matching and spelling correction tools that are only available through that interface. Please see ECitMatch and ESpell below for API equivalents.

Required Parameters

db

Database to search. Value must be a valid Entrez database name (default = pubmed).

term

Entrez text query. All special characters must be URL encoded. Spaces may be replaced by '+' signs. For very long queries (more than several hundred characters long), consider using an HTTP POST call. See the PubMed or Entrez help for information about search field descriptions and tags. Search fields and tags are database specific.

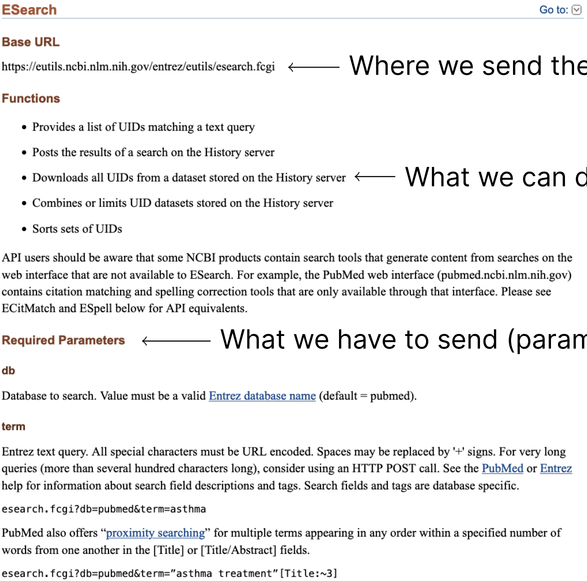
```
esearch.fcgi?db=pubmed&term=asthma
```

PubMed also offers "proximity searching" for multiple terms appearing in any order within a specified number of words from one another in the [Title] or [Title/Abstract] fields.

```
esearch.fcgi?db=pubmed&term="asthma treatment"[Title:~3]
```

Using APIs via HTTPie

https://www.ncbi.nlm.nih.gov/books/NBK25499/#chapter4.ESearch



ESearch

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi ⟵ Where we send the request (endpoint)

Functions

- Provides a list of UIDs matching a text query
- Posts the results of a search on the History server
- Downloads all UIDs from a dataset stored on the History server ⟵ What we can do
- Combines or limits UID datasets stored on the History server
- Sorts sets of UIDs

API users should be aware that some NCBI products contain search tools that generate content from searches on the web interface that are not available to ESearch. For example, the PubMed web interface (pubmed.ncbi.nlm.nih.gov) contains citation matching and spelling correction tools that are only available through that interface. Please see ECitMatch and ESpell below for API equivalents.

Required Parameters ⟵ What we have to send (parameters)

db

Database to search. Value must be a valid Entrez database name (default = pubmed).

term

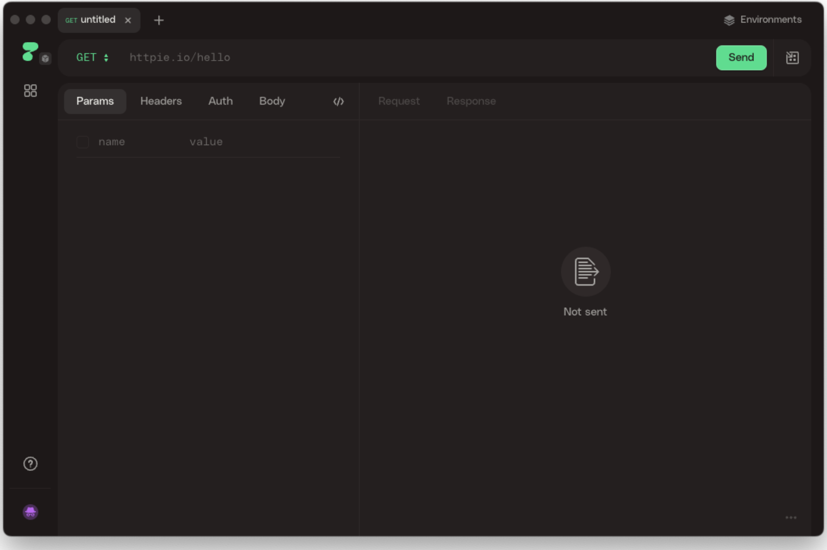
Entrez text query. All special characters must be URL encoded. Spaces may be replaced by '+' signs. For very long queries (more than several hundred characters long), consider using an HTTP POST call. See the PubMed or Entrez help for information about search field descriptions and tags. Search fields and tags are database specific.

```
esearch.fcgi?db=pubmed&term=asthma
```

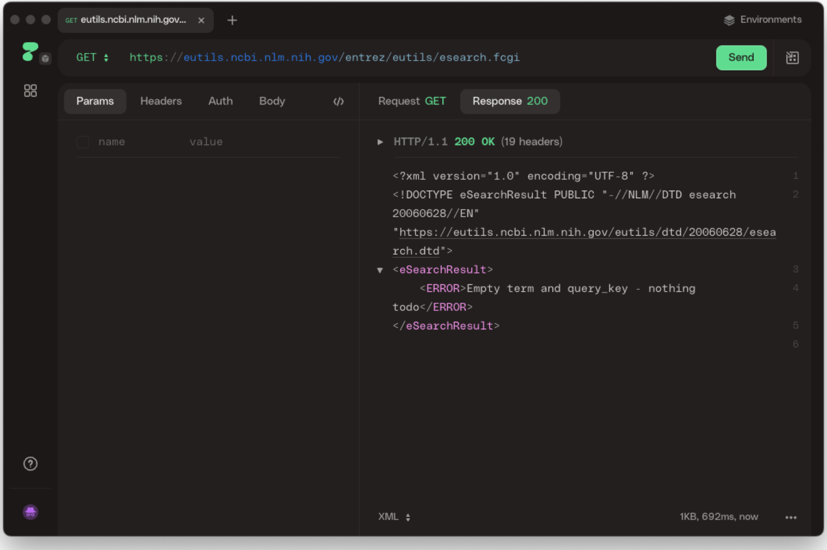
PubMed also offers "proximity searching" for multiple terms appearing in any order within a specified number of words from one another in the [Title] or [Title/Abstract] fields.

```
esearch.fcgi?db=pubmed&term="asthma treatment"[Title:~3]
```

Using APIs via HTTPie



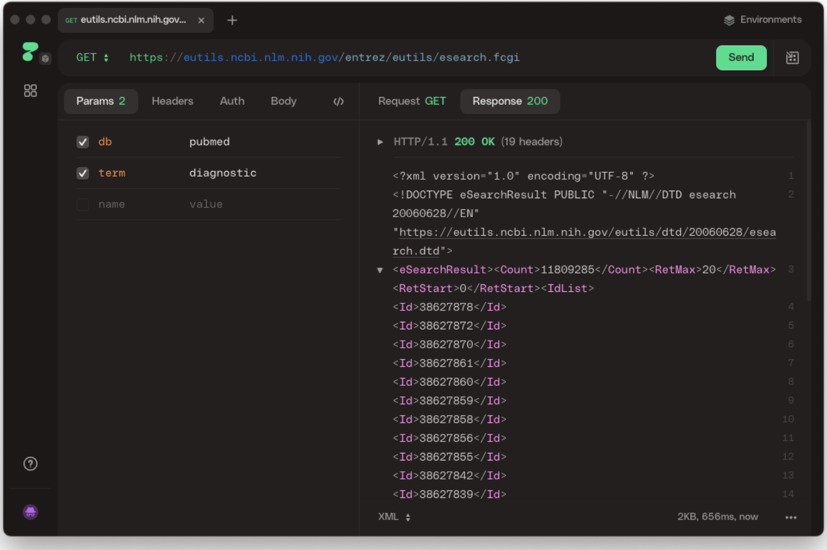
Using APIs via HTTPie



```
▸ HTTP/1.1 200 OK (19 headers)
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE eSearchResult PUBLIC "-//NLM//DTD esearch 20060628//EN" "https://eutils.ncbi.nlm.nih.gov/eutils/dtd/20060628/esearch.dtd">
<eSearchResult>
    <ERROR>Empty term and query_key - nothing todo</ERROR>
</eSearchResult>
```

Using APIs via HTTPie



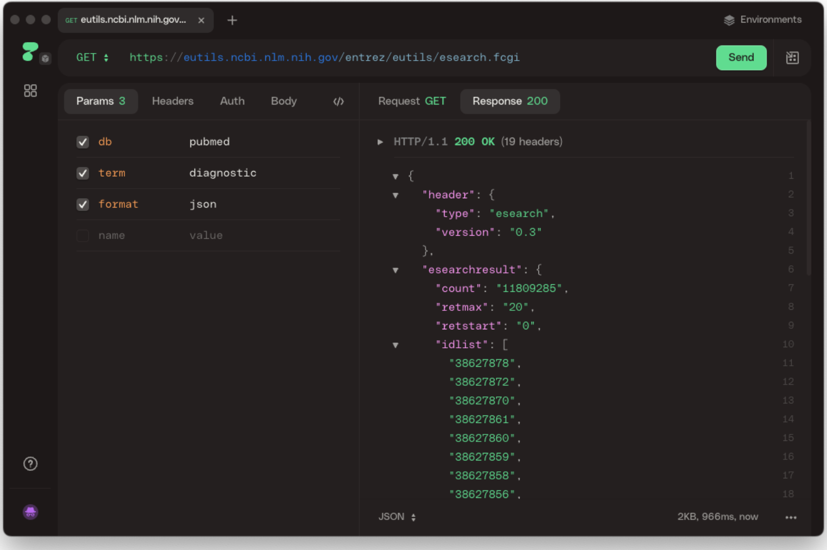
GET https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi

☑	db	pubmed
☑	term	diagnostic
□	name	value

```
▸ HTTP/1.1 200 OK (19 headers)

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE eSearchResult PUBLIC "-//NLM//DTD esearch
20060628//EN"
"https://eutils.ncbi.nlm.nih.gov/eutils/dtd/20060628/esea
rch.dtd">
▾ <eSearchResult><Count>11809285</Count><RetMax>20</RetMax>
<RetStart>0</RetStart><IdList>
<Id>38627878</Id>
<Id>38627872</Id>
<Id>38627870</Id>
<Id>38627861</Id>
<Id>38627860</Id>
<Id>38627859</Id>
<Id>38627858</Id>
<Id>38627856</Id>
<Id>38627855</Id>
<Id>38627842</Id>
<Id>38627839</Id>
```

Using APIs via HTTPie



```
GET https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi
```

☑	db	pubmed
☑	term	diagnostic
☑	format	json
□	name	value

```
▶ HTTP/1.1 200 OK (19 headers)

▼ {
▼   "header": {
      "type": "esearch",
      "version": "0.3"
    },
▼   "esearchresult": {
      "count": "11809285",
      "retmax": "20",
      "retstart": "0",
▼     "idlist": [
        "38627878",
        "38627872",
        "38627870",
        "38627861",
        "38627860",
        "38627859",
        "38627858",
        "38627856",
```

Using APIs via Python

We can also get the exact same result if we use Python

Using APIs via Python

We can also get the exact same result if we use Python

In 0:

```
requests.get(  # GET request
    # URL of the API
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={  # Parameters of the request
        "db": "pubmed",
        "term": "diagnostic",
        "format": "json"
    }
).json()  # Parse the response as JSON
```

Using APIs via Python

We can also get the exact same result if we use Python

In 0:

```
requests.get(  # GET request
    # URL of the API
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={  # Parameters of the request
        "db": "pubmed",
        "term": "diagnostic",
        "format": "json"
    }
).json()  # Parse the response as JSON
```

Out 0:

```
{'header': {'type': 'esearch', 'version': '0.3'},
 'esearchresult': {'count': '11808236',
  'retmax': '20',
  'retstart': '0',
  'idlist': ['38622011',
   '38621987',
   '38621906',
       ...
   '38621734',
   '38621722'],
  'translationset': [{'from': 'diagnostic',
    'to': '"diagnosis"[MeSH Terms] OR "diagnosis"[All Fields] OR
         "diagnostic"[All Fields] OR "diagnostical"[All Fields] OR
         "diagnostically"[All Fields] OR "diagnostics"[All Fields]'}],
  'querytranslation': '"diagnosis"[MeSH Terms] OR "diagnosis"[All Fields]
      OR "diagnostic"[All Fields] OR "diagnostical"[All Fields] OR
      "diagnostically"[All Fields] OR "diagnostics"[All Fields]'}}
```

Using APIs via Python

Python gives us many more tools than applications like HTTPie

In 0:

```
response = requests.get(  # GET request
    # URL of the API
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={  # Parameters of the request
        "db": "pubmed",
        "term": "diagnostic",
        "format": "json"
    }
).json()  # Parse the response as JSON
```

Using APIs via Python

Python gives us many more tools than applications like HTTPie

In 0:

```
response = requests.get(  # GET request
    # URL of the API
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={  # Parameters of the request
        "db": "pubmed",
        "term": "diagnostic",
        "format": "json"
    }
).json()  # Parse the response as JSON
```

In 1:

```
response["esearchresult"]["count"]
```

Out 1:

```
11808236
```

Using APIs via Python

Python gives us many more tools than applications like HTTPie

In 0:

```
response = requests.get(  # GET request
    # URL of the API
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={  # Parameters of the request
        "db": "pubmed",
        "term": "diagnostic",
        "format": "json"
    }
).json()  # Parse the response as JSON
```

In 1:

```
response["esearchresult"]["count"]
```

Out 1:

```
11808236
```

In 2:

```
response["esearchresult"]["idlist"]
```

Out 2:

```
['38622011',
 '38621987',
 '38621906',
     ...
 '38621761',
 '38621754',
 '38621722']
```

Using APIs via Python

We can also search other APIs with the same code, like ClinicalTrials.gov

In 0:

```python
response = requests.get(  # GET request
    url="https://clinicaltrials.gov/api/v2/studies",  # URL of the API
    params={  # Parameters of the request
        "query.cond": "heart+attack",
        "pageSize": 20,
        "format": "json",
        "countTotal": "true"
    }
).json()  # Parse the response as JSON
```

Using APIs via Python

We can also search other APIs with the same code, like ClinicalTrials.gov

In 0:

```python
response = requests.get(  # GET request
    url="https://clinicaltrials.gov/api/v2/studies",  # URL of the API
    params={  # Parameters of the request
        "query.cond": "heart+attack",
        "pageSize": 20,
        "format": "json",
        "countTotal": "true"
    }
).json()  # Parse the response as JSON
```

In 1:

```python
response["totalCount"]
```

Using APIs via Python

We can also search other APIs with the same code, like ClinicalTrials.gov

In 0:

```
response = requests.get(  # GET request
    url="https://clinicaltrials.gov/api/v2/studies",  # URL of the API
    params={  # Parameters of the request
        "query.cond": "heart+attack",
        "pageSize": 20,
        "format": "json",
        "countTotal": "true"
    }
).json()  # Parse the response as JSON
```

In 1:

```
response["totalCount"]
```

Out 1:

```
3299
```

Using APIs via Python

We can also search other APIs with the same code, like ClinicalTrials.gov

In 0:

```python
response = requests.get(  # GET request
    url="https://clinicaltrials.gov/api/v2/studies",  # URL of the API
    params={  # Parameters of the request
        "query.cond": "heart+attack",
        "pageSize": 20,
        "format": "json",
        "countTotal": "true"
    }
).json()  # Parse the response as JSON
```

In 1:

```python
response["totalCount"]
```

Out 1:

```
3299
```

In 2:

```python
for study in response["studies"]:
    print(study["protocolSection"]["identificationModule"]["nctId"],
          study["protocolSection"]["identificationModule"]["officialTitle"])
```

Using APIs via Python

We can also search other APIs with the same code, like ClinicalTrials.gov

In 0:

```
response = requests.get(  # GET request
    url="https://clinicaltrials.gov/api/v2/studies",  # URL of the API
    params={  # Parameters of the request
        "query.cond": "heart+attack",
        "pageSize": 20,
        "format": "json",
        "countTotal": "true"
    }
).json()  # Parse the response as JSON
```

In 1:

```
response["totalCount"]
```

Out 1:

```
3299
```

In 2:

```
for study in response["studies"]:
    print(study["protocolSection"]["identificationModule"]["nctId"],
          study["protocolSection"]["identificationModule"]["officialTitle"])
```

Out 2:

```
NCT02137980 Registry of Non-primary Angioplasty at Hospitals Without
     Surgery On-site
NCT02762162 Online Assistance for Stent Thrombosis
NCT04335162 Cardiovascular Complications in Patients With COVID-19
...
```

Using APIs via Python

We can also search other APIs with the same code, like ClinicalTrials.gov

In 3:

```python
# Stores the number of studies per country
countries = Counter()

# Loop over the studies
for study in response["studies"]:
    locations = study["protocolSection"]["contactsLocationsModule"]
    if "locations" in locations:  # Some studies don't have a location
        location =
            study["protocolSection"]["contactsLocationsModule"]["locations"]
        for loc in location:  # Some studies have multiple locations
            if "country" in loc:
                # +1 to this location
                countries.update([loc["country"]])

# Display the result as a table
pd.DataFrame.from_records(countries.most_common(5),
                          columns=["Country", "Count"])
```

Using APIs via Python

We can also search other APIs with the same code, like ClinicalTrials.gov

```
In 3:
```

```python
# Stores the number of studies per country
countries = Counter()

# Loop over the studies
for study in response["studies"]:
    locations = study["protocolSection"]["contactsLocationsModule"]
    if "locations" in locations:  # Some studies don't have a location
        location =
             study["protocolSection"]["contactsLocationsModule"]["locations"]
        for loc in location:  # Some studies have multiple locations
            if "country" in loc:
                # +1 to this location
                countries.update([loc["country"]])

# Display the result as a table
pd.DataFrame.from_records(countries.most_common(5),
                          columns=["Country", "Count"])
```

```
Out 3:
```

Country	Count
United States	396
Japan	73
Germany	64
Canada	50
Netherlands	41

What are APIs?

How to use APIs?

Use cases for APIs

Summary

Validating Search Queries

Let’s use the Entrez API to validate a search string with seed studies

Validating Search Queries

Let’s use the Entrez API to validate a search string with seed studies

In 0:

```
search_string = '''("Acne Vulgaris"[Mesh] OR Acne[tiab] OR
    Blackheads[tiab] OR Whiteheads[tiab] OR Pimples[tiab]) AND
    ("Phototherapy"[Mesh] OR "Blue light"[tiab] OR Phototherapy[tiab] OR
    Phototherapies[tiab] OR "Photoradiation therapy"[tiab] OR
    "Photoradiation Therapies"[tiab] OR "Light Therapy"[tiab] OR "Light
    Therapies"[tiab]) AND (Randomized controlled trial[pt] OR controlled
    clinical trial[pt] OR randomized[tiab] OR randomised[tiab] OR
    placebo[tiab] OR "drug therapy"[sh] OR randomly[tiab] OR trial[tiab]
    OR groups[tiab]) NOT (Animals[Mesh] not (Animals[Mesh] and
    Humans[Mesh]))'''
seed_studies = ["27575854", "25594129", "20098847", "22091799",
    "23278295", "24313686", "29152718", "10809858", "18664153", "15379878"]
```

Validating Search Queries

Let’s use the Entrez API to validate a search string with seed studies

In 0:

```
search_string = '''("Acne Vulgaris"[Mesh] OR Acne[tiab] OR
     Blackheads[tiab] OR Whiteheads[tiab] OR Pimples[tiab]) AND
     ("Phototherapy"[Mesh] OR "Blue light"[tiab] OR Phototherapy[tiab] OR
     Phototherapies[tiab] OR "Photoradiation therapy"[tiab] OR
     "Photoradiation Therapies"[tiab] OR "Light Therapy"[tiab] OR "Light
     Therapies"[tiab]) AND (Randomized controlled trial[pt] OR controlled
     clinical trial[pt] OR randomized[tiab] OR randomised[tiab] OR
     placebo[tiab] OR "drug therapy"[sh] OR randomly[tiab] OR trial[tiab]
     OR groups[tiab]) NOT (Animals[Mesh] not (Animals[Mesh] and
     Humans[Mesh]))'''
seed_studies = ["27575854", "25594129", "20098847", "22091799",
     "23278295", "24313686", "29152718", "10809858", "18664153", "15379878"]
```

In 1:

```
response = requests.get(
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={  # Parameters of the request
        "db": "pubmed",
        "term": search_string,
        "retmax": 10_000,  # We can retrieve up to 10,000 studies at a time
        "format": "json"
    }
).json()
```

Validating Search Queries

In 2:

```python
def validate(seed_studies, response):
    total_studies = int(response["esearchresult"]["count"])
    retrieved_studies = response["esearchresult"]["idlist"]
    retrieved_seeds = []
    for study in seed_studies:
        if study in retrieved_studies:
            retrieved_seeds.append(study)
    return total_studies, len(retrieved_seeds)
```

Validating Search Queries

In 2:

```
def validate(seed_studies, response):
    total_studies = int(response["esearchresult"]["count"])
    retrieved_studies = response["esearchresult"]["idlist"]
    retrieved_seeds = []
    for study in seed_studies:
        if study in retrieved_studies:
            retrieved_seeds.append(study)
    return total_studies, len(retrieved_seeds)
```

In 3:

```
retrieved_studies, retrieved_seeds = validate(seed_studies, response)
print(f"Total studies retrieved: {retrieved_studies}")
print(f"{retrieved_seeds} of {len(seed_studies)} seed studies.")
```

Validating Search Queries

In 2:

```
def validate(seed_studies, response):
    total_studies = int(response["esearchresult"]["count"])
    retrieved_studies = response["esearchresult"]["idlist"]
    retrieved_seeds = []
    for study in seed_studies:
        if study in retrieved_studies:
            retrieved_seeds.append(study)
    return total_studies, len(retrieved_seeds)
```

In 3:

```
retrieved_studies, retrieved_seeds = validate(seed_studies, response)
print(f"Total studies retrieved: {retrieved_studies}")
print(f"{retrieved_seeds} of {len(seed_studies)} seed studies.")
```

Out 3:

```
Total studies retrieved: 495
10 of 10 seed studies.
```

Validating Search Queries

In 2:

```python
def validate(seed_studies, response):
    total_studies = int(response["esearchresult"]["count"])
    retrieved_studies = response["esearchresult"]["idlist"]
    retrieved_seeds = []
    for study in seed_studies:
        if study in retrieved_studies:
            retrieved_seeds.append(study)
    return total_studies, len(retrieved_seeds)
```

In 3:

```python
retrieved_studies, retrieved_seeds = validate(seed_studies, response)
print(f"Total studies retrieved: {retrieved_studies}")
print(f"{retrieved_seeds} of {len(seed_studies)} seed studies.")
```

Out 3:

```
Total studies retrieved: 495
10 of 10 seed studies.
```

In 4:

```python
print("Precision:", retrieved_seeds / retrieved_studies)
print("Recall:", retrieved_seeds / len(seed_studies))
```

Validating Search Queries

In 2:

```
def validate(seed_studies, response):
    total_studies = int(response["esearchresult"]["count"])
    retrieved_studies = response["esearchresult"]["idlist"]
    retrieved_seeds = []
    for study in seed_studies:
        if study in retrieved_studies:
            retrieved_seeds.append(study)
    return total_studies, len(retrieved_seeds)
```

In 3:

```
retrieved_studies, retrieved_seeds = validate(seed_studies, response)
print(f"Total studies retrieved: {retrieved_studies}")
print(f"{retrieved_seeds} of {len(seed_studies)} seed studies.")
```

Out 3:

```
Total studies retrieved: 495
10 of 10 seed studies.
```

In 4:

```
print("Precision:", retrieved_seeds / retrieved_studies)
print("Recall:", retrieved_seeds / len(seed_studies))
```

Out 4:

```
Precision: 0.02
Recall: 1.0
```

Generating Search Queries with LLMs

Let’s use a local LLM to generate a search and validate it

Generating Search Queries with LLMs

Let’s use a local LLM to generate a search and validate it

In 0:

```python
class ChatAssistant:
    def __init__(self):
        self.pipe = pipeline("text-generation",
                             # Name of the open LLM
                             model="BioMistral/BioMistral-7B-SLERP")
        self.context = []

    def chat(self, message):
        # Append all the previous interactions as context
        prompt = self.pipe.tokenizer.apply_chat_template(
                     self.context + [{"role": "user", "content": message}],
                     tokenize=False,
                     add_generation_prompt=True)
        # Generate a response to the current message
        outputs = self.pipe(prompt,
                            max_new_tokens=256,
                            do_sample=True,
                            temperature=0.7,
                            top_k=50,
                            top_p=0.95)
        # Append the two turns to the context history
        self.context.append({"role": "user",
                             "content": message})
        self.context.append({"role": "assistant",
                             "content": outputs[0]["generated_text"]})
        # Show the text the LLM generated
        print(outputs[0]["generated_text"].split("[/INST]")[-1])

assistant = ChatAssistant()
```

Generating Search Queries with LLMs

In 1:

```
statement = "Blue-Light Therapy for Acne Vulgaris"
title = "A randomized controlled study for the treatment of acne vulgaris using high-intensity 414 nm solid state diode arrays."
```

Generating Search Queries with LLMs

In 1:

```
statement = "Blue-Light Therapy for Acne Vulgaris"
title = "A randomized controlled study for the treatment of acne vulgaris
    using high-intensity 414 nm solid state diode arrays."
```

In 2:

```
assistant.chat(f"Follow my instructions precisely to develop a highly
    effective Boolean query for a medical systematic review literature
    search. Do not explain or elaborate. First, Given the following
    statement and title from a relevant study, identify 10 terms or
    phrases that are relevant. The terms you identify will be used to
    retrieve more relevant studies. statement: {statement} title: {title}")
```

Generating Search Queries with LLMs

In 1:

```
statement = "Blue-Light Therapy for Acne Vulgaris"
title = "A randomized controlled study for the treatment of acne vulgaris
     using high-intensity 414 nm solid state diode arrays."
```

In 2:

```
assistant.chat(f"Follow my instructions precisely to develop a highly
     effective Boolean query for a medical systematic review literature
     search. Do not explain or elaborate. First, Given the following
     statement and title from a relevant study, identify 10 terms or
     phrases that are relevant. The terms you identify will be used to
     retrieve more relevant studies. statement: {statement} title: {title}")
```

Out 2:

```
1. Blue-light therapy
2. Acne vulgaris
3. High-intensity
4. 414 nm
5. Solid state diode arrays
6. Randomized controlled study
7. Treatment
8. Acne
9. Solid state
10. Diode
```

Generating Search Queries with LLMs

In 3:

```
assistant.chat("For each item in step 1, classify it as of three
     categories: terms relating to health conditions (A), terms relating to
     a treatment (B), terms relating to types of study design (C). When an
     item does not fit one of these categories, mark it as (N/A). Do not
     explain or elaborate.")
```

Generating Search Queries with LLMs

In 3:

```
assistant.chat("For each item in step 1, classify it as of three
     categories: terms relating to health conditions (A), terms relating to
     a treatment (B), terms relating to types of study design (C). When an
     item does not fit one of these categories, mark it as (N/A). Do not
     explain or elaborate.")
```

Out 3:

```
1. Blue-light therapy (B)
2. Acne vulgaris (A)
3. High-intensity (N/A)
4. 414 nm (N/A)
5. Solid state diode arrays (B)
6. Randomized controlled study (C)
7. Treatment (B)
8. Acne (A)
9. Solid state (N/A)
10. Diode (B)
```

Generating Search Queries with LLMs

In 3:

```
assistant.chat("For each item in step 1, classify it as of three
     categories: terms relating to health conditions (A), terms relating to
     a treatment (B), terms relating to types of study design (C). When an
     item does not fit one of these categories, mark it as (N/A). Do not
     explain or elaborate.")
```

Out 3:

```
1. Blue-light therapy (B)
2. Acne vulgaris (A)
3. High-intensity (N/A)
4. 414 nm (N/A)
5. Solid state diode arrays (B)
6. Randomized controlled study (C)
7. Treatment (B)
8. Acne (A)
9. Solid state (N/A)
10. Diode (B)
```

In 4:

```
assistant.chat("Using the list in step 2, use your expert knowledge to
     create a valid Boolean query that can be submitted to PubMed which
     groups together items from each category. Also add relevant MeSH terms
     into the query where necessary. Each main clause of the query must
     correspond to a PICO element. Do not explain or elaborate.")
```

Generating Search Queries with LLMs

In 3:

```
assistant.chat("For each item in step 1, classify it as of three
     categories: terms relating to health conditions (A), terms relating to
     a treatment (B), terms relating to types of study design (C). When an
     item does not fit one of these categories, mark it as (N/A). Do not
     explain or elaborate.")
```

Out 3:

```
1. Blue-light therapy (B)
2. Acne vulgaris (A)
3. High-intensity (N/A)
4. 414 nm (N/A)
5. Solid state diode arrays (B)
6. Randomized controlled study (C)
7. Treatment (B)
8. Acne (A)
9. Solid state (N/A)
10. Diode (B)
```

In 4:

```
assistant.chat("Using the list in step 2, use your expert knowledge to
     create a valid Boolean query that can be submitted to PubMed which
     groups together items from each category. Also add relevant MeSH terms
     into the query where necessary. Each main clause of the query must
     correspond to a PICO element. Do not explain or elaborate.")
```

Out 4:

```
((blue-light therapy[Mesh] OR blue light therapy) AND (acne vulgaris[Mesh]
     OR acne vulgaris)) AND ((randomized controlled trial[Mesh]) OR
     randomized controlled study) AND (treatment OR therapy)
```

Generating Search Queries with LLMs

In 5:

```
gen_search = assistant.context[-1]["content"]
```

Generating Search Queries with LLMs

In 5:

```python
gen_search = assistant.context[-1]["content"]
```

In 6:

```python
response = requests.get(
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={"db": "pubmed", "term": gen_search, # Search we generated
            "retmax": 10_000, "format": "json"}
).json()
gen_retrieved, gen_retrieved_seeds = validate(seed_studies, response)
```

Generating Search Queries with LLMs

In 5:

```
gen_search = assistant.context[-1]["content"]
```

In 6:

```
response = requests.get(
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={"db": "pubmed", "term": gen_search, # Search we generated
            "retmax": 10_000, "format": "json"}
).json()
gen_retrieved, gen_retrieved_seeds = validate(seed_studies, response)
```

In 7:

```
print(f"Total studies retrieved: {gen_retrieved}")
print(f"{gen_retrieved_seeds} of {len(gen_retrieved_seeds)} seed studies.")
```

Generating Search Queries with LLMs

In 5:

```
gen_search = assistant.context[-1]["content"]
```

In 6:

```
response = requests.get(
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={"db": "pubmed", "term": gen_search, # Search we generated
            "retmax": 10_000, "format": "json"}
).json()
gen_retrieved, gen_retrieved_seeds = validate(seed_studies, response)
```

In 7:

```
print(f"Total studies retrieved: {gen_retrieved}")
print(f"{gen_retrieved_seeds} of {len(gen_retrieved_seeds)} seed studies.")
```

Out 7:

```
Total studies retrieved: 141
9 of 10 seed studies.
```

Generating Search Queries with LLMs

In 5:

```python
gen_search = assistant.context[-1]["content"]
```

In 6:

```python
response = requests.get(
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={"db": "pubmed", "term": gen_search, # Search we generated
            "retmax": 10_000, "format": "json"}
).json()
gen_retrieved, gen_retrieved_seeds = validate(seed_studies, response)
```

In 7:

```python
print(f"Total studies retrieved: {gen_retrieved}")
print(f"{gen_retrieved_seeds} of {len(gen_retrieved_seeds)} seed studies.")
```

Out 7:

```
Total studies retrieved: 141
9 of 10 seed studies.
```

In 8:

```python
print("Precision:", retrieved_seeds / gen_retrieved)
print("Recall:", gen_retrieved_seeds / len(gen_retrieved_seeds))
```

Generating Search Queries with LLMs

In 5:

```
gen_search = assistant.context[-1]["content"]
```

In 6:

```
response = requests.get(
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={"db": "pubmed", "term": gen_search, # Search we generated
            "retmax": 10_000, "format": "json"}
).json()
gen_retrieved, gen_retrieved_seeds = validate(seed_studies, response)
```

In 7:

```
print(f"Total studies retrieved: {gen_retrieved}")
print(f"{gen_retrieved_seeds} of {len(gen_retrieved_seeds)} seed studies.")
```

Out 7:

```
Total studies retrieved: 141
9 of 10 seed studies.
```

In 8:

```
print("Precision:", retrieved_seeds / gen_retrieved)
print("Recall:", gen_retrieved_seeds / len(gen_retrieved_seeds))
```

Out 8:

```
Precision: 0.063
Recall: 0.9
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

Let’s run queries on PubMed & ClinicalTrials.gov and de-duplicate them

Searching & De-duplicating Pubmed and ClinicalTrials.gov

Let’s run queries on PubMed & ClinicalTrials.gov and de-duplicate them

In 0:

```
pubmed_search_string = """("Acne Vulgaris"[Mesh] OR Acne[tiab] OR
    Blackheads[tiab] OR Whiteheads[tiab] OR Pimples[tiab]) AND
    ("Phototherapy"[Mesh] OR "Blue light"[tiab] OR Phototherapy[tiab] OR
    Phototherapies[tiab] OR "Photoradiation therapy"[tiab] OR
    "Photoradiation Therapies"[tiab] OR "Light Therapy"[tiab] OR "Light
    Therapies"[tiab]) AND (Randomized controlled trial[pt] OR controlled
    clinical trial[pt] OR randomized[tiab] OR randomised[tiab] OR
    placebo[tiab] OR "drug therapy"[sh] OR randomly[tiab] OR trial[tiab]
    OR groups[tiab]) NOT (Animals[Mesh] not (Animals[Mesh] and
    Humans[Mesh]))"""
ct_search_string = "(Acne AND (Phototherapy OR light))"
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

Let’s run queries on PubMed & ClinicalTrials.gov and de-duplicate them

In 0:

```
pubmed_search_string = """("Acne Vulgaris"[Mesh] OR Acne[tiab] OR
    Blackheads[tiab] OR Whiteheads[tiab] OR Pimples[tiab]) AND
    ("Phototherapy"[Mesh] OR "Blue light"[tiab] OR Phototherapy[tiab] OR
    Phototherapies[tiab] OR "Photoradiation therapy"[tiab] OR
    "Photoradiation Therapies"[tiab] OR "Light Therapy"[tiab] OR "Light
    Therapies"[tiab]) AND (Randomized controlled trial[pt] OR controlled
    clinical trial[pt] OR randomized[tiab] OR randomised[tiab] OR
    placebo[tiab] OR "drug therapy"[sh] OR randomly[tiab] OR trial[tiab]
    OR groups[tiab]) NOT (Animals[Mesh] not (Animals[Mesh] and
    Humans[Mesh]))"""
ct_search_string = "(Acne AND (Phototherapy OR light))"
```

In 1:

```
pubmed_pmids = get_pmids_from_pubmed(pubmed_search_string)
len(pubmed_pmids)
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

Let’s run queries on PubMed & ClinicalTrials.gov and de-duplicate them

In 0:

```
pubmed_search_string = """("Acne Vulgaris"[Mesh] OR Acne[tiab] OR
    Blackheads[tiab] OR Whiteheads[tiab] OR Pimples[tiab]) AND
    ("Phototherapy"[Mesh] OR "Blue light"[tiab] OR Phototherapy[tiab] OR
    Phototherapies[tiab] OR "Photoradiation therapy"[tiab] OR
    "Photoradiation Therapies"[tiab] OR "Light Therapy"[tiab] OR "Light
    Therapies"[tiab]) AND (Randomized controlled trial[pt] OR controlled
    clinical trial[pt] OR randomized[tiab] OR randomised[tiab] OR
    placebo[tiab] OR "drug therapy"[sh] OR randomly[tiab] OR trial[tiab]
    OR groups[tiab]) NOT (Animals[Mesh] not (Animals[Mesh] and
    Humans[Mesh]))"""
ct_search_string = "(Acne AND (Phototherapy OR light))"
```

In 1:

```
pubmed_pmids = get_pmids_from_pubmed(pubmed_search_string)
len(pubmed_pmids)
```

Out 1:

```
496
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

Let’s run queries on PubMed & ClinicalTrials.gov and de-duplicate them

In 0:

```
pubmed_search_string = """("Acne Vulgaris"[Mesh] OR Acne[tiab] OR
    Blackheads[tiab] OR Whiteheads[tiab] OR Pimples[tiab]) AND
    ("Phototherapy"[Mesh] OR "Blue light"[tiab] OR Phototherapy[tiab] OR
    Phototherapies[tiab] OR "Photoradiation therapy"[tiab] OR
    "Photoradiation Therapies"[tiab] OR "Light Therapy"[tiab] OR "Light
    Therapies"[tiab]) AND (Randomized controlled trial[pt] OR controlled
    clinical trial[pt] OR randomized[tiab] OR randomised[tiab] OR
    placebo[tiab] OR "drug therapy"[sh] OR randomly[tiab] OR trial[tiab]
    OR groups[tiab]) NOT (Animals[Mesh] not (Animals[Mesh] and
    Humans[Mesh]))"""
ct_search_string = "(Acne AND (Phototherapy OR light))"
```

In 1:

```
pubmed_pmids = get_pmids_from_pubmed(pubmed_search_string)
len(pubmed_pmids)
```

Out 1:

```
496
```

In 2:

```
ct_nctids = get_ntcids_from_clinicaltrials(ct_search_string)
len(ct_nctids)
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

Let’s run queries on PubMed & ClinicalTrials.gov and de-duplicate them

In 0:

```
pubmed_search_string = """("Acne Vulgaris"[Mesh] OR Acne[tiab] OR
    Blackheads[tiab] OR Whiteheads[tiab] OR Pimples[tiab]) AND
    ("Phototherapy"[Mesh] OR "Blue light"[tiab] OR Phototherapy[tiab] OR
    Phototherapies[tiab] OR "Photoradiation therapy"[tiab] OR
    "Photoradiation Therapies"[tiab] OR "Light Therapy"[tiab] OR "Light
    Therapies"[tiab]) AND (Randomized controlled trial[pt] OR controlled
    clinical trial[pt] OR randomized[tiab] OR randomised[tiab] OR
    placebo[tiab] OR "drug therapy"[sh] OR randomly[tiab] OR trial[tiab]
    OR groups[tiab]) NOT (Animals[Mesh] not (Animals[Mesh] and
    Humans[Mesh]))"""
ct_search_string = "(Acne AND (Phototherapy OR light))"
```

In 1:

```
pubmed_pmids = get_pmids_from_pubmed(pubmed_search_string)
len(pubmed_pmids)
```

Out 1:

```
496
```

In 2:

```
ct_nctids = get_ntcids_from_clinicaltrials(ct_search_string)
len(ct_nctids)
```

Out 2:

```
106
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

In 3:

```
pubmed_ntcid_search_string = " OR ".join(
    [f"{nct_id}[SI]" for nct_id in ct_nctids])
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

In 3:

```python
pubmed_ntcid_search_string = " OR ".join(
    [f"{nct_id}[SI]" for nct_id in ct_nctids])
```

In 4:

```python
pubmed_ntcid_response = requests.get(
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={
        "db": "pubmed",
        "term": pubmed_ntcid_search_string,
        "retmax": 10_000,
        "format": "json"
    }
).json()

pubmed_ntcid_pmids = pubmed_ntcid_response["esearchresult"]["idlist"]
len(pubmed_ntcid_pmids)
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

In 3:

```python
pubmed_ntcid_search_string = " OR ".join(
    [f"{nct_id}[SI]" for nct_id in ct_nctids])
```

In 4:

```python
pubmed_ntcid_response = requests.get(
    url="https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi",
    params={
        "db": "pubmed",
        "term": pubmed_ntcid_search_string,
        "retmax": 10_000,
        "format": "json"
    }
).json()

pubmed_ntcid_pmids = pubmed_ntcid_response["esearchresult"]["idlist"]
len(pubmed_ntcid_pmids)
```

Out 4:

```
5
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

In 5:

```
deduplicated_pmids = list(set(pubmed_pmids).union(set(pubmed_ntcid_pmids)))
len(deduplicated_pmids)
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

In 5:

```
deduplicated_pmids = list(set(pubmed_pmids).union(set(pubmed_ntcid_pmids)))
len(deduplicated_pmids)
```

Out 5:

```
500
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

In 5:

```
deduplicated_pmids = list(set(pubmed_pmids).union(set(pubmed_ntcid_pmids)))
len(deduplicated_pmids)
```

Out 5:

```
500
```

In 6:

```
missing_nctids =
    pubmed_ntcid_response["esearchresult"]["errorlist"]["phrasesnotfound"]
len(missing_nctids)
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

In 5:

```
deduplicated_pmids = list(set(pubmed_pmids).union(set(pubmed_ntcid_pmids)))
len(deduplicated_pmids)
```

Out 5:

```
500
```

In 6:

```
missing_nctids =
    pubmed_ntcid_response["esearchresult"]["errorlist"]["phrasesnotfound"]
len(missing_nctids)
```

Out 6:

```
101
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

In 7:

```
pubmed_studies = get_pubmed_studies(deduplicated_pmids)
ct_studies = get_ct_studies(missing_nctids)
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

In 7:

```
pubmed_studies = get_pubmed_studies(deduplicated_pmids)
ct_studies = get_ct_studies(missing_nctids)
```

In 8:

```
pd.DataFrame(pubmed_studies + ct_studies)
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

In 7:

```
pubmed_studies = get_pubmed_studies(deduplicated_pmids)
ct_studies = get_ct_studies(missing_nctids)
```

In 8:

```
pd.DataFrame(pubmed_studies + ct_studies)
```

Out 8:

title	abstract	pmid	nctid
Topical methyl aminolevulinate photodynamic th...	Photodynamic therapy (PDT) has been found to b...	17598868	None
...	...	...	...
A Study to Evaluate the Potential of Tazaroten...	The purpose of this study is to evaluate the p...	None	NCT01119651

Searching & De-duplicating Pubmed and ClinicalTrials.gov

In 7:

```
pubmed_studies = get_pubmed_studies(deduplicated_pmids)
ct_studies = get_ct_studies(missing_nctids)
```

In 8:

```
pd.DataFrame(pubmed_studies + ct_studies)
```

Out 8:

title	abstract	pmid	nctid
Topical methyl aminolevulinate photodynamic th...	Photodynamic therapy (PDT) has been found to b...	17598868	None
...	...	...	...
A Study to Evaluate the Potential of Tazaroten...	The purpose of this study is to evaluate the p...	None	NCT01119651

In 9:

```
len(pubmed_studies + ct_studies)
```

Searching & De-duplicating Pubmed and ClinicalTrials.gov

In 7:

```
pubmed_studies = get_pubmed_studies(deduplicated_pmids)
ct_studies = get_ct_studies(missing_nctids)
```

In 8:

```
pd.DataFrame(pubmed_studies + ct_studies)
```

Out 8:

title	abstract	pmid	nctid
Topical methyl aminolevulinate photodynamic th...	Photodynamic therapy (PDT) has been found to b...	17598868	None
...	...	...	...
A Study to Evaluate the Potential of Tazaroten...	The purpose of this study is to evaluate the p...	None	NCT01119651

In 9:

```
len(pubmed_studies + ct_studies)
```

Out 9:

```
601
```

Frequency Analysis

Let’s create a term frequency analysis tool using Entrez

Frequency Analysis

Let’s create a term frequency analysis tool using Entrez

```
In 0:
```

```
seed_studies = ["27575854", "25594129", "20098847", "22091799", "23278295",
                "24313686", "29152718", "10809858", "18664153", "15379878"]
```

Frequency Analysis

Let’s create a term frequency analysis tool using Entrez

In 0:

```
seed_studies = ["27575854", "25594129", "20098847", "22091799", "23278295",
                "24313686", "29152718", "10809858", "18664153", "15379878"]
```

In 1:

```
pubmed_studies = get_pubmed_studies(seed_studies)
```

Frequency Analysis

Let’s create a term frequency analysis tool using Entrez

In 0:

```
seed_studies = ["27575854", "25594129", "20098847", "22091799", "23278295",
                "24313686", "29152718", "10809858", "18664153", "15379878"]
```

In 1:

```
pubmed_studies = get_pubmed_studies(seed_studies)
```

In 2:

```
# Combine the title and abstracts
study_data = [study["TI"] for study in pubmed_studies] +
             [study["AB"] for study in pubmed_studies]
```

Frequency Analysis

In 3:

```
# Contains all the punctuation we want to remove
table = str.maketrans(string.punctuation, ' ' * len(string.punctuation))

# Count the frequency of each term
term_frequency = Counter()
for s in study_data:
    term_frequency.update(s.lower().translate(table).split())

# Show the results of the top 100 most commonly occuring terms
pd.DataFrame(term_frequency.most_common(100),
             columns=["Term", "Frequency"])
```

Frequency Analysis

In 3:

```
# Contains all the punctuation we want to remove
table = str.maketrans(string.punctuation, ' ' * len(string.punctuation))

# Count the frequency of each term
term_frequency = Counter()
for s in study_data:
    term_frequency.update(s.lower().translate(table).split())

# Show the results of the top 100 most commonly occuring terms
pd.DataFrame(term_frequency.most_common(100),
             columns=["Term", "Frequency"])
```

Out 3:

Term	Frequency
the	160
of	128
and	101
a	65
...	...
conclusions	6
use	6
their	6
treatments	6

Frequency Analysis

In 4:

```
# Count the frequency of each term
phrase_frequency = Counter()
for s in study_data:
    s = s.lower().translate(table).split()
    for i in range(len(s) - 1):
        phrase_frequency.update([" ".join(s[i:i + 2])])
    for i in range(len(s) - 2):
        phrase_frequency.update([" ".join(s[i:i + 3])])

# Show the results of the top 100 most commonly occuring phrases
pd.DataFrame(phrase_frequency.most_common(100),
             columns=["Phrase", "Frequency"])
```

Frequency Analysis

In 4:

```
# Count the frequency of each term
phrase_frequency = Counter()
for s in study_data:
    s = s.lower().translate(table).split()
    for i in range(len(s) - 1):
        phrase_frequency.update([" ".join(s[i:i + 2])])
    for i in range(len(s) - 2):
        phrase_frequency.update([" ".join(s[i:i + 3])])

# Show the results of the top 100 most commonly occuring phrases
pd.DataFrame(phrase_frequency.most_common(100),
             columns=["Phrase", "Frequency"])
```

Out 4:

Term	Frequency
blue light	32
in the	27
the treatment	25
acne vulgaris	18
...	...
light device	4
severe acne	4
6 weeks	4
the led	4

Frequency Analysis

In 5:

```
no_stopwords_frequency = Counter()
for term, count in (phrase_frequency + term_frequency).items():
    if all([word not in stopwords and str.isalpha(word) for word in
         term.split()]):
        no_stopwords_frequency[term] = count
pd.DataFrame(no_stopwords_frequency.most_common(100),
             columns=["Phrase", "Frequency"])
```

Frequency Analysis

In 5:

```
no_stopwords_frequency = Counter()
for term, count in (phrase_frequency + term_frequency).items():
    if all([word not in stopwords and str.isalpha(word) for word in
         term.split()]):
        no_stopwords_frequency[term] = count
pd.DataFrame(no_stopwords_frequency.most_common(100),
             columns=["Phrase", "Frequency"])
```

Out 5:

Term	Frequency
treatment	55
blue light	32
phototherapy	22
acne vulgaris	18
...	...
blue light irradiation	5
blue red	5
extension	5
long	5

Finding MeSH Terms

Let’s use the extracted terms and phrases to find MeSH terms

Finding MeSH Terms

Let’s use the extracted terms and phrases to find MeSH terms

In 0:

```
mesh_response = requests.get(
    url="https://meshmate.ielab.io:8443/api/v1/resources/mesh",
    params={
        "term": "$".join([item[0] for item in
            no_stopwords_frequency.most_common(100)]),
        "type": "Semantic"
    }
).json()
```

Finding MeSH Terms

Let’s use the extracted terms and phrases to find MeSH terms

In 0:

```
mesh_response = requests.get(
    url="https://meshmate.ielab.io:8443/api/v1/resources/mesh",
    params={
        "term": "$".join([item[0] for item in
            no_stopwords_frequency.most_common(100)]),
        "type": "Semantic"
    }
).json()
```

In 1:

```
suggested_mesh_terms = []
for suggestions in mesh_response["Data"]:
    suggested_mesh_terms.append((suggestions["Keywords"],
        list(suggestions["MeSH_Terms"].values())))
pd.DataFrame(suggested_mesh_terms, columns=["Terms", "MeSH Terms"])
```

Finding MeSH Terms

Let’s use the extracted terms and phrases to find MeSH terms

```
In 0:
```

```
mesh_response = requests.get(
    url="https://meshmate.ielab.io:8443/api/v1/resources/mesh",
    params={
        "term": "$".join([item[0] for item in
            no_stopwords_frequency.most_common(100)]),
        "type": "Semantic"
    }
).json()
```

```
In 1:
```

```
suggested_mesh_terms = []
for suggestions in mesh_response["Data"]:
    suggested_mesh_terms.append((suggestions["Keywords"],
        list(suggestions["MeSH_Terms"].values())))
pd.DataFrame(suggested_mesh_terms, columns=["Terms", "MeSH Terms"])
```

```
Out 1:
```

Terms	MeSH Terms
acne, acne vulgaris, inflammatory acne	Acne Vulgaris, Administration, Topical
light, blue light, light phototherapy	Light, Photochemotherapy, Ultraviolet Rays
trial, main trial, randomized	Randomized Controlled Trials as Topic
...	...

What are APIs?

How to use APIs?

Use cases for APIs

Summary

Summary

APIs can be used to automate many tasks in evidence identification

- Retrieval and deduplication of studies
- Statistical text analysis
- Reimplementing and extending tools

Requires a new set of skills

- Familiarisation with API fundamentals
- Learning a programming language
- Data science and data management

This workshop and the supplementary material provided the basics

---

Let’s get in touch!

`https://scells.me/irm24` `@hscells` `harry.scells@uni-leipzig.de`