In this module

• Introduction to data repositories and APIs
• Hands on tutorial using nanoHUB: data querying for oxides
• Hands on tutorial using nanoHUB: performing your own tailored queries and data filtering
• Homework assignments
Learning objectives and prerequisites

After completing this lecture you will:

• Learn about and query data repositories
• Manage data through efficient Pandas dataframes
  • Parsing
  • Logic operations
  • Data visualization and selection
  • Type management
• Perform tailored queries specific to user application

Pre-requisites:
• Basic Python programming
Launching a Jupyter tool in nanoHUB

Querying Data Repositories

From your browser go to link: https://nanohub.org/tools/matdatarepo/

Click on Launch Tool to begin
Step 1: Landing Page – Notebook: Querying Data

- Read through overview
- Navigate to “Introduction to materials databases and their accessibility” first
  - This will provide you information on how to get access to each of these repositories
STEP 1: get API keys for MP

Follow the links included in steps 3.1 to get an API key for Materials Project

Keep these API keys private! They are linked to your personal accounts and should not be distributed freely. Account access may be revoked by client if terms of use are broken.
Querying and managing data

Tasks

• Perform large queries to database repository
  • Filter results
  • Convert between types of data

• Plot results & find correlations
  • Assess the scale of your data
  • Advanced images with Plotly
Run through the cells in the Jupyter notebook (Shift+Enter)

Filter and parse data cell by cell to reduce load of queried data

Load API keys – press ‘Enter’ after. A message displays if your key has been stored
Querying the Materials Project & managing data

1. Query all oxides in MP database

```python
# Query every structure in the MP database that has oxygen atom within compound.
# Grab set of properties from database.
data = rester.query({"elements": "O", "nelements": {"gte": 2}},
                  ["task_id", "pretty_formula", "formula", "volume", "density", "elements",
                   "e_above_hull", "elasticity", "unit_cell_formula"])
```

You can change the number of properties queried by editing the property requests in the rester.query().

Problem 1: Add piezoelectric data to the oxide query

- Filter and parse data cell by cell to reduce load of queried data

See Pymatgen documentation for more info: https://pymatgen.org/usage.html
Querying the Materials Project & managing data

Filter database based on structure stability
1. Only structures with energy within 1 meV of the ground state are kept

Convert data into accessible pandas dataframe

```
<table>
<thead>
<tr>
<th>density</th>
<th>e_above_hull</th>
<th>elasticity</th>
<th>elements</th>
<th>formula</th>
<th>pretty_formula</th>
<th>task_id</th>
<th>unit_cell_formula</th>
<th>volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7125813</td>
<td>0.038578</td>
<td>None</td>
<td>[Eu, Fe, O]</td>
<td>EuFeO3</td>
<td>mp-1011260</td>
<td>119.222279</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.374443</td>
<td>0.218318</td>
<td>None</td>
<td>[Ge, Li, Ni, O]</td>
<td>Li3Ni2(GeO4)3</td>
<td>mp-1013756</td>
<td>832.269582</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.573133</td>
<td>0.130555</td>
<td>None</td>
<td>[Li, Mo, O, P]</td>
<td>LiMo3P3O13</td>
<td>mp-1013856</td>
<td>553.654637</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.087849</td>
<td>0.000000</td>
<td>None</td>
<td>[Ba, Eu, O]</td>
<td>BaEu2O4</td>
<td>mp-1019738</td>
<td>473.481865</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
def oxide = pd.DataFrame.from_dict(data)
display(oxide)
```
Perform our own calculations on the data

For each oxide:
- calculate the ionic packing fraction
- append result to dictionary

```python
# Create temporary list arrays for data calculation and manipulation.
IPF = []

for k in range(0, len(df_oxide)):
    if str(df_oxide['e_above_hull'].iloc[k]) == 'None':
        continue

    # Grab values from data query
    oxide_volume = df_oxide['volume'].iloc[k]
    atom_type_count = len(df_oxide['elements'].iloc[k])
    unit_cell_form = df_oxide['unit_cell_formula'].iloc[k]
    atom_count = []
    atom_sample = []

    # Pull atom count for each atom type. Will be used to find summed atomic volumes and summed masses
    atoms = list(unit_cell_form.keys())
    atom_count = list(unit_cell_form.values())

    # Do a quick query of each element type for ionic radius and atomic mass from Pymatgen
    for item in atoms:
        element_object = pymatgen.Element(item)
        radii_values = element_object.average_ionic_radius
        atom_volume.append((4/3)*np.pi*radii_values**3)

    atom_volume_total = []
    for i in np.arange(0, len(atom_count)):
        v_total = atom_count[i]*atom_volume[i]
        atom_volume_total.append(v_total)

    # Sum the volumes of all atoms in the oxide
    summed_volumes = sum(atom_volume_total)

    # Calculate packing fraction. Append values to dictionary with the Materials Project ID as key value
    packing_fra = summed_volumes/oxide_volume
    IPF.append(packing_fra)
```
Plot the query results with Plotly
### 3. Filtering data for elasticity

```python
1 df_oxide = df_oxide.dropna(subset=['Elasticity'])
```

3.1 Observe the reduction from all oxide species to the limited set for stable structures, and ones with mechanical property data

```python
1 print('Number of oxide structures available on the HP database: %s' % len(data))
2 print('Number of oxide structures below 1.0 meV and with elasticity data: %s' % (energy_cutoff_value, len(df_oxide)))
```

Number of oxide structures available on the HP database: 70702
Number of oxide structures below 1.0 meV and with elasticity data: 856

### 3.2 Full dataframe column with dictionary of elastic constants. Create separate columns for each value instead of just in dictionary.

```python
1 #Pull dataframe column with dictionary of elastic constants
2 elastic = df_oxide['Elasticity']
3 G_VHN = []
4 C_VHN = []
5 Elastic_Aneisotropy = []
6 poisson_ratio = []
7 for k in elastic.keys():
8     G_VHN.append(elastic[k].get('G_VHN'))
9     C_VHN.append(elastic[k].get('C_VHN'))
10     Elastic_Aneisotropy.append(elastic[k].get('Elastic_Aneisotropy'))
11     poisson_ratio.append(elastic[k].get('poisson_ratio'))
12 elasticity_data = pd.DataFrame({
13     'G_VHN': G_VHN,
14     'C_VHN': C_VHN,
15     'Elastic_Aneisotropy': Elastic_Aneisotropy,
16     'poisson_ratio': poisson_ratio
17     }, index=elastic.keys())
18 df_oxide = df_oxide.reset_index(drop=True)
19 df_oxide['Elasticity'] = df_oxide['Elasticity'].reset_index(drop=True)
20 df_oxide = df_oxide.drop(['Elasticity', 'Elasticity'], axis=1)
21 display(df_oxide)
22 #Remove erroneous poisson ratios
23 valid_test = []
24 row_delete = []
25 for k in range(len(df_oxide)):
26     if df_oxide['poisson_ratio'].iloc[k] < 0:
27         row_delete.append(k)
28         df_oxide.drop(index=row_delete, axis=0, inplace=True)
```

- Filter results that do not have elastic constants
- Expand elastic constant dictionary
Querying the Materials Project & managing data

Stable oxides

Stable oxides + elasticity
data filtered

Oxide Query IPF vs. Density

Oxide Query IPF vs. Density

Zachary McClure - Repositories and data management
Querying the Materials Project & managing data

3.3.2 Plot material properties from dataframe using 3rd property as color key

```python
import pandas as pd
import plotly.express as px

# Extract dataframe
data = pd.read_csv('materials.csv')

# Plotting
fig = px.scatter(data, x='Bulk Modulus [GPa]', y='Shear Modulus [GPa]', color='Poisson Ratio', title='Oxide Query Bulk Modulus vs. Shear Modulus', hover_data=['material', 'formula'], size='density', size_max=50)
fig.show()
```

Advanced plotting & property correlation
Summary

• Easily queryable databases offer researches access to data that can be useful for initial material searches, or for creating rapid sets of descriptors for machine learning models.

• Management of these queried datasets can be easily done using pandas dataframes.

• The code in https://nanohub.org/tools/matdatarepo can be easily modified to meet needs in a wide range of problems.
Next steps

Homework assignment: to reinforce concepts and help students modify the workflow and adapt it for their needs