

# Hands-On Tutorial: Data-Mining the QM9 Dataset

Raghunathan Ramakrishnan\*

<sup>1</sup> Tata Institute of Fundamental Research, Hyderabad 500046, India  
(Dated: November 22, 2024)

This tutorial introduces the application of the QM9 dataset using Python. We explore basic aspects of data retrieval, statistics, and analysis using the `qm9pack` Python module.

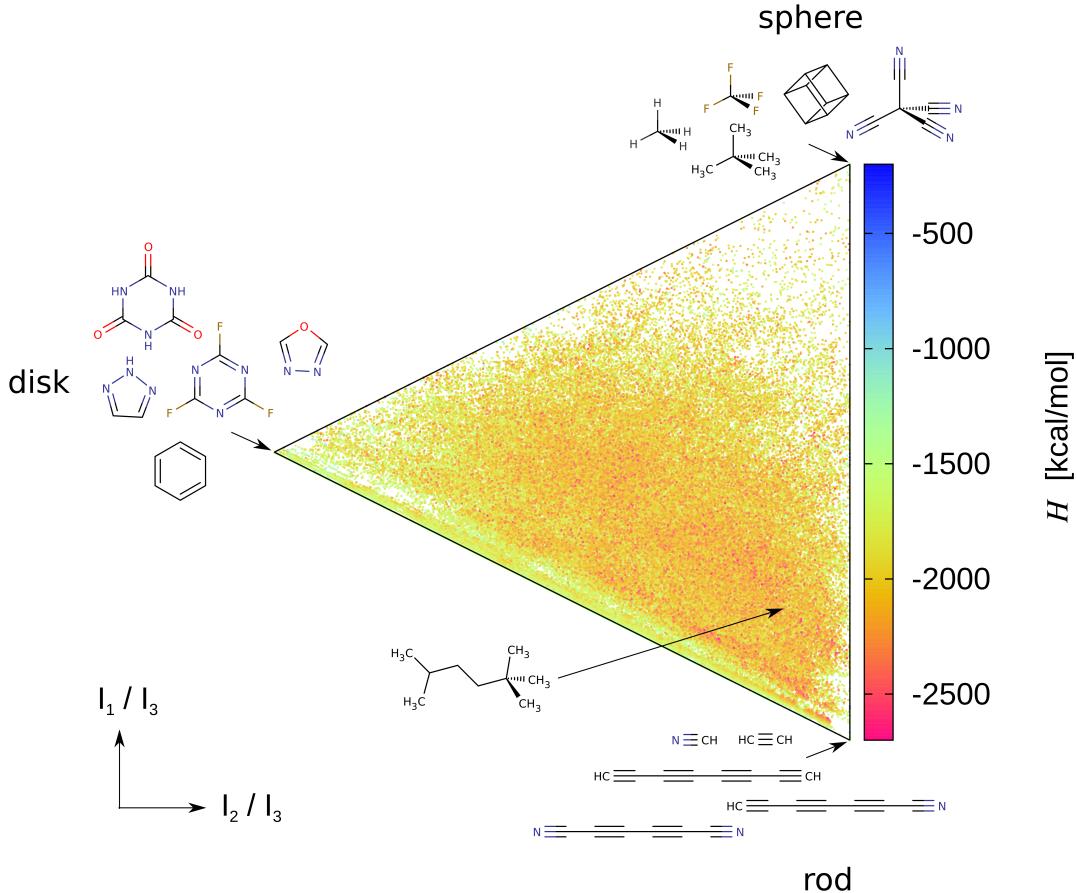


FIG. 1. Distribution of QM9 molecules as a function of their shape and stability. The  $x$  and  $y$  axes are normalized moments of inertia ( $I_1/I_3$  and  $I_2/I_3$ ), and the atomization enthalpy ( $H$  in kcal/mol) from density functional theory calculations is color-coded. Also shown are the representative 1D (rod-shaped), 2D (disc-shaped), and 3D (near-spherical) molecules at the corners. Each single dot corresponds to one or many QM9 molecules.

\* [ramakrishnan@tifrh.res.in](mailto:ramakrishnan@tifrh.res.in)

**CONTENTS**

I. Introduction	3
1. Energies in the QM9 dataset	3
2. A Word on Python	4
II. Basic Exercises	4
A. Loading the QM9 Dataset	4
Exercise A.1: Load the dataset	4
B. Exploring the Content of QM9 Dataset	4
Exercise B.1: Overview of <code>columns</code>	4
Exercise B.2: <code>helper</code>	5
Exercise B.3: <code>helper</code> for all <code>columns</code>	6
Exercise B.4: <code>Cartesian coordinates</code> using <code>makexyz</code>	7
Exercise B.5: <code>Collect coordinates</code> of all molecules	7
Exercise B.6: Filter molecules using <code>Stoichiometry</code>	8
III. Advanced Exercises	8
Exercise C.1: All stoichiometries in QM9	8
Exercise C.2: Atomization energy	9
Exercise C.3: Multiproperty query	10
Exercise C.4: Extract all constitutional isomers for a given stoichiometry	11
IV. Acknowledgments	12
V. Data Availability	12
References	12

## I. INTRODUCTION

The QM9 dataset [1] contains minimum energy geometries and various properties of 133,885 molecules, calculated using the density functional theory method B3LYP and the 6-31G(2df,*p*) basis set. Among these, 3,054 molecules faced convergence issues during structure optimization and were excluded, resulting in a final dataset of 130,831 molecules. The 3,054 ‘uncharacterized’ molecules have been analyzed separately, highlighting the critical role of quantum chemistry approximations in establishing the correspondence between a molecular graph and its three-dimensional structure [2]. For each of the 130,831 molecules, the dataset contains the SMILES representation[3] collected from the GDB17 database[4] containing 166.4 billion molecular graphs. In this tutorial, we will use the Python modules Pandas[5, 6], Matplotlib[7], NumPy[8], and SciPy[9]. The QM9 dataset is shared through the module `qmlpack`, which uses Pandas and Numpy.

### 1. Energies in the QM9 dataset

The QM9 dataset reports four thermochemistry energies defined according to Ref. 10: internal energy at zero kelvin (0 K), denoted customarily as  $U_0$ ; internal energy at 298.15 K (room temperature), denoted customarily as  $U_T$ ; enthalpy at 298.15 K, denoted customarily as  $H_T$ ; Gibbs free energy at 298.15 K, denoted customarily as  $G_T$ . These four energies can be queried using the keys (or column names): `InternalEnergy_0K_au`, `InternalEnergy_298K_au`, `Enthalpy_298K_au`, and `GibbsFreeEnergy_298K_au`. Here are the formal definitions of these energies:

1.  $U_0$  is the energy of the molecule at 0 K, which is the sum of its electronic energy (including the classical electron-electron repulsion term),  $E_{\text{electronic}}$ , and the zero-point vibrational energy (ZPVE)

$$U_0 = E_{\text{electronic}} + \text{ZPVE}. \quad (1)$$

2.  $U_T$  is the room temperature internal energy, which is the sum of the  $E_{\text{electronic}}$  and the total thermal corrections to energy along all degrees of freedom (translational, rotational, vibrational, and electronic),  $E_{\text{tot}}$

$$U_T = E_{\text{electronic}} + E_{\text{tot}}. \quad (2)$$

Note that  $E_{\text{tot}}$  includes ZPVE.

3.  $H_T$  is the room temperature enthalpy, which is the sum of  $E_{\text{electronic}}$ ,  $E_{\text{tot}}$ , and the energy due to the volume of the system, which under the ideal gas assumption is  $k_B T$ .

$$H_T = E_{\text{electronic}} + E_{\text{tot}} + k_B T. \quad (3)$$

The enthalpy correction  $E_{\text{tot}} + k_B T$  is often denoted as  $H_{\text{corr}}$ .

4.  $G_T$  is the room temperature Gibbs free energy, which is the sum of  $E_{\text{electronic}}$ , free energy correction defined as  $G_{\text{corr}} = H_{\text{corr}} - TS_{\text{tot}}$ , where  $S_{\text{tot}}$  is the total entropy along all degrees of freedom (translational, rotational, vibrational, and electronic).

$$G_T = E_{\text{electronic}} + G_{\text{corr}}. \quad (4)$$

The absolute values of  $U_0$ ,  $U_T$ ,  $H_T$ , and  $G_T$  of molecules can be used to calculate the change in the corresponding property during a reaction. For example, for a reaction  $A \rightarrow B$ , one can calculate the change in the free energy as  $\Delta G = G_B - G_A$ .

For most applications, one might prefer energies that quantify the stability of the molecule. Hence, one has to consider the relative values of the energies compared to a reference, such as free atoms. The molecular atomization energy is the sum of all bond energies, defined as

$$E_{\text{atomization}} = U_0^{\text{molecule}} - \sum_{A \in \text{atoms}} U_0^A \quad (5)$$

According to this definition,  $E_{\text{atomization}}$  is a negative number, implying all QM9 molecules are thermodynamically more stable than the free atoms. As stated above,  $U_0^{\text{molecule}}$  includes the ZPVE. For the free atoms, there are no

vibrational degrees of freedom; hence,  $U_0^A$  is the same  $E_{\text{electronic}}$ . TABLE I contains the energies of the CHONF atoms calculated using the DFT method, B3LYP, using the basis set, 6-31G(2df, p), as reported in [1] and can be accessed from the original data repository[11].

TABLE I. Reference thermochemical energies of H, C, N, O, F atoms in hartree, collected from Ref. 11.

Element	$U_0$	$U_T$	$H_T$	$G_T$
H	-0.500273	-0.498857	-0.497912	-0.510927
C	-37.846772	-37.845355	-37.844411	-37.861317
N	-54.583861	-54.582445	-54.581501	-54.598897
O	-75.064579	-75.063163	-75.062219	-75.079532
F	-99.718730	-99.717314	-99.716370	-99.733544

## 2. A Word on Python

To complete the exercises, you will need a `python3.x` compiler. I have tried these exercises using Python versions 3.9 and 3.11. All the code examples are available in the folder <https://github.com/raghurama123/qm9pack/tree/main/tutorials>. If you wish to develop your own exercises or workflows, it is beneficial to have some understanding of working with the DataFrame object in Pandas.

## II. BASIC EXERCISES

### A. Loading the QM9 Dataset

*Exercise A.1: Load the dataset*

In the first exercise, we will load the `qm9pack` module, fetch the QM9 dataset (`qm9`), and perform initial exploratory analysis.

---

```
import qm9pack # Import the qm9pack module

# Fetch the QM9 dataset and store it in a DataFrame
df = qm9pack.get_data('qm9')

# Print the summary statistics of the dataset
print(df.describe())

# Print the first 5 rows of the dataset
print(df.head())

# Print the last 5 rows of the dataset
print(df.tail())
```

---

<OUTPUT>  
Long output, not shown.

---

### B. Exploring the Content of QM9 Dataset

*Exercise B.1: Overview of columns*

Let us find the names of all columns (properties) in `qm9`.

---

```

import qm9pack # Import the qm9pack module

# Fetch the QM9 dataset and store it in a DataFrame
df = qm9pack.get_data('qm9')

# Iterate over each column in the DataFrame and print the column name
for key in df.columns:
    print(key)

```

---

```
<OUTPUT>
XYZ_file
Index
SMILES
InChi
N_atoms
Stoichiometry
Elements
XYZ_Ang
Mulliken_pop
Harmonic_Freq_cmi
RotA_GHz
RotB_GHz
RotC_GHz
Dipole_debye
Polarizability_bohr3
HOMO_au
LUMO_au
HOMO_LUMO_gap_au
R2_bohr2
ZPVE_au
InternalEnergy_0K_au
InternalEnergy_298K_au
Enthalphy_298K_au
GibbsFreeEnergy_298K_au
Heatcapacity_Cv_cal_mol_K
```

---

#### *Exercise B.2: helper*

Get more details about one of the columns in `qm9`.

```

import qm9pack # Import the qm9pack module

# Fetch the QM9 dataset and store it in a DataFrame
df = qm9pack.get_data('qm9')

# Define a specific column name for which to print more details
key = 'RotC_GHz'

# Use the helper function from qm9pack to print more details for the
# specified column
qm9pack.helper(key)

```

---

```
<OUTPUT>
'RotC_GHz': Rotational constant C in GHz.
```

---

*Exercise B.3: helper for all columns*

Get more details about all the columns in qm9.

---

```
import qm9pack # Import the qm9pack module

# Fetch the QM9 dataset and store it in a DataFrame
df = qm9pack.get_data('qm9')

# Iterate over each column in the DataFrame and print more details for
# each column
for key in df.columns:
    qm9pack.helper(key)
```

---

<OUTPUT>

'XYZ\_file': The file containing the XYZ coordinates of the molecule as submitted in the original QM9 article (<https://doi.org/10.1038/sdata.2014.22>). The name 'dsgdb9nsd\_000001.xyz' contain data for molecule-1 and so on. Overall, qm9 contains 130831 molecules

'Index': A unique identifier for each molecule in the dataset which ranges from 1 to 133885 of which 3054 will be missing as the corresponding molecular structures failed to converge with the DFT method, B3LYP, and the basis set, 6-31G(2df,p)

'SMILES': Simplified Molecular Input Line Entry System representation of the molecule.

'InChi': IUPAC International Chemical Identifier for the molecule.

'N\_atoms': The number of atoms in the molecule.

'Stoichiometry': The stoichiometric formula of the molecule encoded as the array containing numbers of H, C, N, O, and F atoms. For CH4, the list corresponds to [4,1,0,0,0]

'Elements': List of elements present in the molecule. For CH4, the list is ['C', 'H', 'H', 'H', 'H']

'XYZ\_Ang': The XYZ coordinates of the molecule in Angstroms calculated using the DFT method, B3LYP, and the basis set, 6-31G(2df,p).

'Mulliken\_pop': Mulliken population analysis data for the molecule in units of charge of an electron, e.

'Harmonic\_Freq\_cmi': Harmonic vibrational frequencies of the molecule in cm<sup>-1</sup>.

'RotA\_GHz': Rotational constant A in GHz.

'RotB\_GHz': Rotational constant B in GHz.

'RotC\_GHz': Rotational constant C in GHz.

'Dipole\_debye': Dipole moment of the molecule in debye.

'Polarizability\_bohr3': Polarizability of the molecule in bohr<sup>3</sup>.

'HOMO\_au': Highest Occupied Molecular Orbital energy in atomic units, hartree.

'LUMO\_au': Lowest Unoccupied Molecular Orbital energy in atomic units, hartree.

'HOMO\_LUMO\_gap\_au': Energy gap between HOMO and LUMO in atomic units, hartree.

'R2\_bohr2': Spread of the electron density calculated as the expectation value of the operator, R<sup>2</sup>, in bohr<sup>2</sup>

'ZPVE\_au': No help content available for this key.

'InternalEnergy\_0K\_au': Internal energy of the molecule at 0 kelvin.

'InternalEnergy\_298K\_au': Internal energy of the molecule at 298.15 kelvin.

'Enthalphy\_298K\_au': Enthalpy of the molecule at 298.15 kelvin.

'GibbsFreeEnergy\_298K\_au': Gibbs free energy of the molecule at 298.15 kelvin.

'Heatcapacity\_Cv\_cal\_mol\_K': Heat capacity at constant volume in cal/(mol K).

---

#### Exercise B.4: Cartesian coordinates using makexyz

Store the Cartesian coordinates of a molecule in qm9 corresponding to an index.

---

```
import qm9pack # Import the qm9pack module

# Fetch the QM9 dataset and store it in a DataFrame
df = qm9pack.get_data('qm9')

# Save XYZ for a molecule using its index
index = 0
filename = 'Mol_0.xyz'
qm9pack.makexyz(index, df, filename)

index = 1
filename = 'Mol_1.xyz'
qm9pack.makexyz(index, df, filename)
```

---

```
<OUTPUT>
5
Mol_0.xyz
C   -0.01269814    1.08580416    0.00800100
H    0.00215042   -0.00603132    0.00197612
H    1.01173084    1.46375116    0.00027657
H    -0.54081507    1.44752661   -0.87664372
H    -0.52381363    1.43793264    0.90639729
4
Mol_1.xyz
N   -0.04042605    1.02410775    0.06256380
H    0.01725746    0.01254521   -0.02737716
H    0.91578937    1.35874519   -0.02875776
H    -0.52027774    1.34353213   -0.77554261
```

---

#### Exercise B.5: Collect coordinates of all molecules

Store the Cartesian coordinates of all molecules in qm9 in the file qm9\_130831\_molecules.xyz. WARNING: The code will run for a while and will create an approximately 108 MB file.

---

```
import qm9pack          # Import the qm9pack module
import os               # Import the os module

df = qm9pack.get_data('qm9')          # Fetch the QM9 dataset

filename = 'qm9_130831_molecules.xyz' # Output filename

# Create the output file
with open(filename, 'w') as f:
    pass

# Loop through each molecule in the dataset
for index in range(len(df)):
    tmpfile = 'mol_' + str(index) + '.xyz' # Temporary filename
    qm9pack.makexyz(index, df, tmpfile)      # Create the XYZ file
```

```
# Append contents to the main outputfile
with open(tmpfile, 'r') as temp_file:
    contents = temp_file.read()

with open(filename, 'a') as final_file:
    final_file.write(contents)

os.remove(tmpfile)                                # Remove temporary file
```

---

<OUTPUT>  
Long output, not shown.

---

#### *Exercise B.6: Filter molecules using Stoichiometry*

Let us find all molecules in `qm9` matching a stoichiometry. Let us use  $C_2H_6O$  corresponding to two constitutional isomers, ethanol ( $CH_3CH_2OH$ ) and dimethylether ( $CH_3OCH_3$ ). We will filter the dataset to make a filtered DataFrame. Then, we will go over each molecule in this subset and print SMILES and InChi strings. Note that `_` (underscore) is a placeholder for the index of the DataFrame (which is not used).

NOTE: You can modify this exercise and retrieve any other properties for a set of constitutional isomers. You can use the names of the columns from II B Exercise B.1. For example, if you want to print SMILES and dipole moment, you can modify the last line of the code as

```
print(mol['SMILES'], mol['Dipole_debye'])
```

---

```
import qm9pack

df = qm9pack.get_data('qm9')

# stoichiometry is a string of list with number of H, C, N, O, and F atoms
stoi = "[6,2,0,1,0]" # Six H, two C, zero N, one O, and zero F

# Filter the DataFrame to match the stoichiometry
filtered_df = df[df['Stoichiometry'].apply(lambda x: x == stoi)]

# Iterate over each row in the filtered DataFrame
for _, mol in filtered_df.iterrows():
    # Print SMILES and InChi for each molecule
    print(mol['SMILES'], mol['InChi'])
```

---

<OUTPUT>  
CCO InChI=1S/C2H6O/c1-2-3/h3H,2H2,1H3  
CCOC InChI=1S/C2H6O/c1-3-2/h1-2H3

---

### III. ADVANCED EXERCISES

#### *Exercise C.1: All stoichiometries in QM9*

Let us find all the unique stoichiometries in `qm9` and find their counts. We will find that the most populated stoichiometry corresponds to  $C_7H_{10}O_2$ , containing 6095 molecules, which has been extensively studied in Ref. 12. NOTE: Remember, Python counts from zero.

---

```

import qm9pack

df = qm9pack.get_data('qm9')

# Get counts of unique values in the 'Stoichiometry' column, in descending order
value_counts = df['Stoichiometry'].value_counts(ascending=False)

# Iterating over each unique value and its count
for index, count in value_counts.items():
    print(f"{index}: {count}") # Printing each unique value and its count

```

---

```
<OUTPUT>
[10,7,0,2,0]: 6094
[11,7,1,1,0]: 5858
[9,6,1,2,0]: 5630
[9,7,1,1,0]: 5215
[12,8,0,1,0]: 4918
[12,7,0,2,0]: 4612
[7,6,1,2,0]: 3872
[8,6,2,1,0]: 3556
[10,8,0,1,0]: 3171
[14,8,0,1,0]: 3154
[8,7,0,2,0]: 3009
...
...
```

---

### *Exercise C.2: Atomization energy*

Let us calculate the atomization energy of a molecule using its index. As an example we will set `index=0` corresponding to methane. We will use  $U_0$  of free atoms collected in TABLE. I.

NOTE: You can also consider atomization energy as a positive number by defining it as the energy of the reaction Molecule $\rightarrow$ atoms. In that case, you can change the line

```
U0_atomization = U0_mol - np.dot(stoi, U0_atoms)
to
U0_atomization = np.dot(stoi, U0_atoms) - U0_mol
```

---

```

import numpy as np
import qm9pack

df = qm9pack.get_data('qm9')

# Conversion constants from Hartree to other units
hartree2ev = 27.211386245
hartree2kcm = 627.5094740631
hartree2kJm = 2625.4996394799

# U0 for H, C, N, O, and F atoms from the manual (in atomic units)
U0_atoms = np.array([-0.500273, -37.846772, -54.583861, -75.064579, -99.718730])

# Selecting the molecule at index 0
index = 0
mol = df.loc[index]

# Extracting the internal energy of the molecule at 0 K in atomic units (au)

```

```

U0_mol = mol['InternalEnergy_0K_au']

# Extracting the stoichiometry of the molecule
stoi = mol['Stoichiometry']

# Convert the stoichiometry string to an array of float
str_list = stoi.strip('[]').split(',')
stoi = [float(num) for num in str_list]

# Calculating the atomization energy
U0_atomization = U0_mol - np.dot(stoi, U0_atoms)

print(f'Atomization energy is {U0_atomization:.8f} hartree')
print(f'Atomization energy is {U0_atomization * hartree2ev:.6f} eV')
print(f'Atomization energy is {U0_atomization * hartree2kcm:.5f} kcal/mol')
print(f'Atomization energy is {U0_atomization * hartree2kJm:.4f} kJ/mol')



---


<OUTPUT>
Atomization energy is -0.63106600 hartree
Atomization energy is -17.172181 eV
Atomization energy is -395.99989 kcal/mol
Atomization energy is -1656.8636 kJ/mol

```

### *Exercise C.3: Multiproperty query*

Let us do a multiproperty query and get the SMILES of all molecules in qm9 with dipole moment in the range 5–8 debye and HOMO-LUMO gap in the range 2–5 eV. Let's print the output in a file `multiquery.txt` with molecules arranged in ascending order of dipole moment.

```

import qm9pack

df = qm9pack.get_data('qm9')

# Conversion factor from hartree to eV
hartree2ev = 27.211386245

# Define minimum and maximum values for Dipole moment and HOMO-LUMO gap (in eV)
min_dipole = 5.0
max_dipole = 8.0

min_gap = 2.0 / hartree2ev # Minimum HOMO-LUMO gap converted to eV
max_gap = 5.0 / hartree2ev # Maximum HOMO-LUMO gap converted to eV

# Combined condition to filter rows based on two criteria
combined_condition = (
    (df['Dipole_debye'] >= min_dipole) & (df['Dipole_debye'] <= max_dipole) &
    (df['HOMO_LUMO_gap_au'] >= min_gap) & (df['HOMO_LUMO_gap_au'] <= max_gap)
)

# Apply the combined condition to filter the DataFrame
filtered_df = df[combined_condition]

# Sort filtered DataFrame by 'Dipole_debye' column in ascending order
sorted_df = filtered_df.sort_values(by='Dipole_debye', ascending=True)

```

```
# Write SMILES, HOMO-LUMO gap (in eV), and Dipole moment to file
with open('multiquery.txt', 'w') as f:
    for _, mol in sorted_df.iterrows():
        f.write(f'{mol['SMILES']} {mol['HOMO_LUMO_gap_au']} * hartree2ev {mol['Dipole_debye']}\n')

<OUTPUT>
printed in the file 'multiquery.txt'
```

---

*Exercise C.4: Extract all constitutional isomers for a given stoichiometry*

This example is based on the content from the supplementary information of a graph neural network study[13] that used qm9pack to extract the subset of qm9 molecules with a given stoichiometry.

In this example, we define the number of H, C, N, O, and F atoms as `stoi = "[4,6,1,1,1]"` which corresponds to H4 C6 N1 O1 F1 or C6H4NOF. We will print the SMILES and a property of interest (e.g. dipole moment), and collect all the B3LYP/6-31G(2df,p)-level coordinates reported in the qm9 dataset in a file '`QM9_subset_C6H4NOF.xyz`'.

---

```
import qm9pack
import os

df = qm9pack.get_data('qm9')

# Define the stoichiometry of interest in the order NH, NC, NN, NO, NF
stoi = "[4,6,1,1,1]" # corresponds to H4 C6 N1 O1 F1 or C6H4NOF

# Collect coordinates in a file
filename='QM9_subset_C6H4NOF.xyz'

# Extract subset of QM9 with the stoichiometry defined above
filtered_df = df[df['Stoichiometry'].apply(lambda x: x == stoi)]

Nmols=len(filtered_df)
print(f'{Nmols} molecules are found in QM9 for the defined stoichiometry')

# Print SMILES and the property of interest such as dipole moment
for _, mol in filtered_df.iterrows():
    print(f'{mol['SMILES']},{mol['Dipole_debye']}')

with open(filename, 'w') as f:
    pass

# Loop through each molecule in the subset
for index in range(len(filtered_df)):

    tmpfile = 'mol_' + str(index) + '.xyz' # Temporary file

    qm9pack.makexyz(index, filtered_df, tmpfile)      # Create the XYZ

    with open(tmpfile, 'r') as temp_file:
        contents = temp_file.read()

    with open(filename, 'a') as final_file: # copy in filename
        final_file.write(contents)

    os.remove(tmpfile)                      # Remove temporary file
```

---

```
<OUTPUT>
10 molecules are found in QM9 for the defined stoichiometry
FC1=CC(C=O)=CC=N1,0.7098
FC1=CC(C=O)=CN=C1,2.5274
FC1=CC(C=O)=NC=C1,3.6415
FC1=NC(C=O)=CC=C1,5.3096
FC1=CN=C(C=O)C=C1,4.2053
FC1=NC=C(C=O)C=C1,3.4893
FC1=CC=CN=C1C=0,4.2773
FC1=CC=NC=C1C=0,3.2381
FC1=CN=CC=C1C=0,2.5198
FC1=NC=CC=C1C=0,4.7799
```

The coordinates are collected in the file 'QM9\_subset\_C6H4NOF.xyz'

---

#### IV. ACKNOWLEDGMENTS

I am grateful to everyone who has reached out with inquiries regarding the QM9 data, which served as the primary inspiration for developing this package. Big thanks to OAvL for letting me dive into the QM9 project and for treating me to an unforgettable dinner on the evening we completed the QM9 article. I acknowledge the support of the Department of Atomic Energy, Government of India, under Project Identification No. RTI 4007.

#### V. DATA AVAILABILITY

The qm9pack is maintained here: <https://github.com/raghurama123/qm9pack>

---

- [1] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, Quantum chemistry structures and properties of 134 kilo molecules, *Sci. Data* **1**, 1 (2014).
- [2] S. Senthil, S. Chakraborty, and R. Ramakrishnan, Troubleshooting unstable molecules in chemical space, *Chem. Sci.* **12**, 5566 (2021).
- [3] D. Weininger, Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules, *J. Chem. Inf. Comp. Sci.* **28**, 31 (1988).
- [4] L. Ruddigkeit, R. Van Deursen, L. C. Blum, and J.-L. Reymond, Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17, *J. Chem. Inf. Model.* **52**, 2864 (2012).
- [5] J. Reback, W. McKinney, J. Van Den Bossche, T. Augspurger, P. Cloud, A. Klein, S. Hawkins, M. Roeschke, J. Tratner, C. She, *et al.*, pandas-dev/pandas: Pandas 1.0. 5, Zenodo (2020).
- [6] Wes McKinney, Data Structures for Statistical Computing in Python, in *Proceedings of the 9th Python in Science Conference*, edited by Stéfan van der Walt and Jarrod Millman (2010) pp. 56 – 61.
- [7] J. D. Hunter, Matplotlib: A 2d graphics environment, *Computing in Science & Engineering* **9**, 90 (2007).
- [8] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, Array programming with NumPy, *Nature* **585**, 357 (2020).
- [9] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, *Nature Methods* **17**, 261 (2020).
- [10] J. W. Ochterski, Thermochemistry in Gaussian (2000).
- [11] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, Dataset: Quantum chemistry structures and properties of 134 kilo molecules (2014).
- [12] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, Big data meets quantum chemistry approximations: the  $\Delta$ -machine learning approach, *J. Chem. Theory Comput.* **11**, 2087 (2015).
- [13] S. A. R. Sangala and S. Raghunathan, Graph neural network for 3-dimensional structures including dihedral angles for molecular property prediction, (2024).