An Interactive SDF Viewer

RDKit UGM 3

23-Oct-2014

Axel Pahl

Dr. Axel Pahl
Medicinal Chemistry
AVIRU
Technische Universität München
Tel.: 089-289-13822
pahl@aviru.de
axel.pahl@tum.de

whoami

- Medicinal Chemist (12+ yrs. experience)
  - still working in the lab

- Linux / OpenSource Enthusiast
- Programming Skills
  - Python, Postgresql, RDKit, HTML
  - Pipeline Pilot, KNIME

- Married, two children (10 & 12 yrs. old)

CV Summary

- 1998 PhD in Organic Chemistry
- 1998 - 2012 Research at Solvay Pharmaceuticals / Abbott Products (Hannover)
  - 2 yrs. Chemical Pilot Plant (GMP)
  - 9+ yrs. Medicinal Chemistry
    - Metalloprotease Inhibitors (ECE, NEP)
    - HCV Polymerase Inhibitors (NS5B palm II site)
- 2013 - today AVIRU Project (TU Munich)
  - Virulence Inhibitors (a.o. ClpP) for MRSA

SDF Viewer

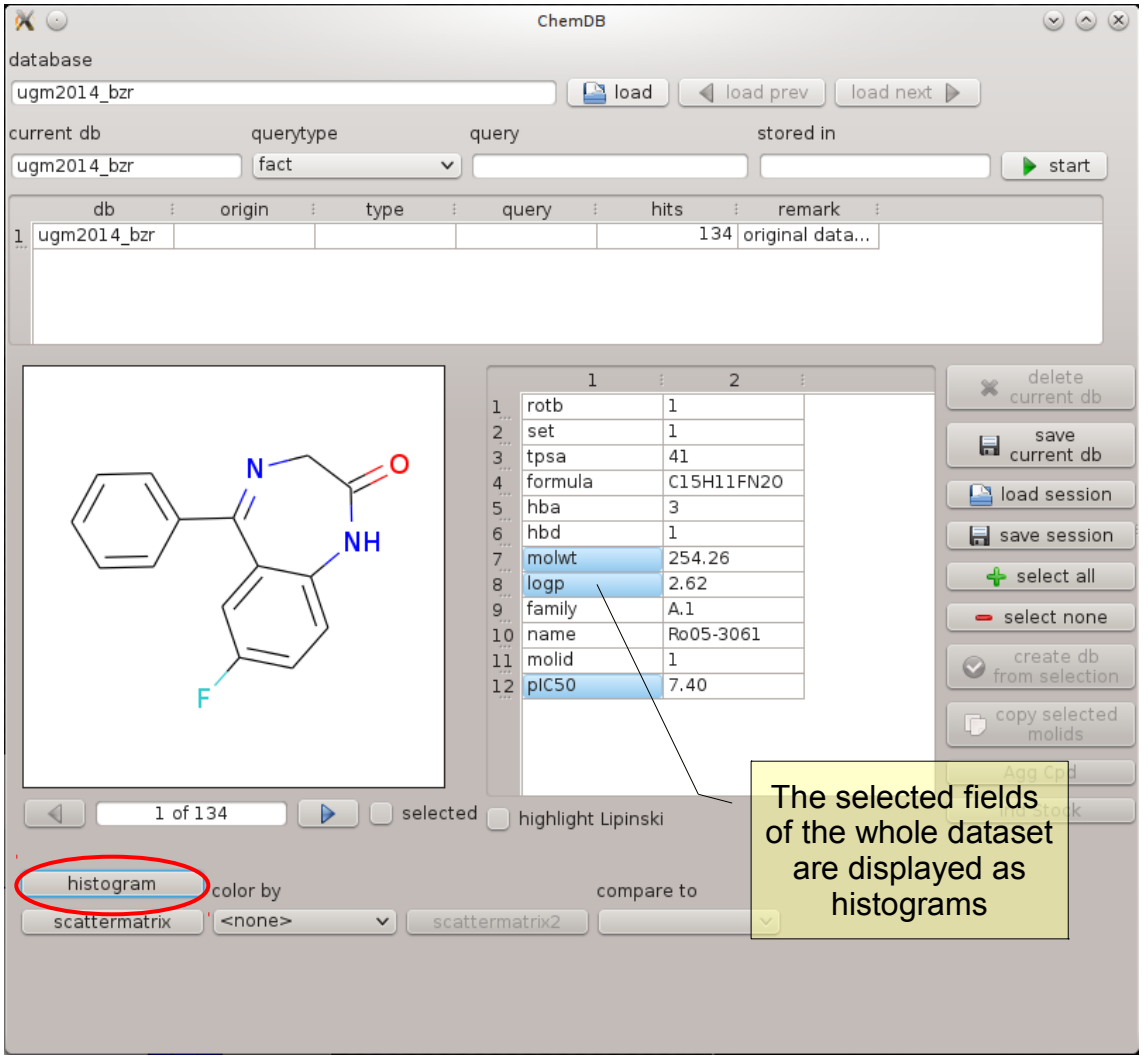
- SD File Viewer
- Display properties that are stored in the SD file
- Interactive visualization of the properties
  - color data points by category properties (e.g. compound family) (slide 8)
  - compare a subset (e.g. from a substructure search) to the whole SD file graphically (slide 9)
  - display molecules from data points
    - clicking a data point once displays the record in the SDF Viewer
- Create subsets / new SD files
  - fact or substructure (SMILES) searches
  - manual selection of molecules, then <create db from selection>
    - in the SDF Viewer directly or
    - by clicking data points twice in the scatter plots
  - subsets can always be saved as new SD file
  - program sessions can be stored
  - special behaviour for SD files saved from Aldrich or Chemspider (slide 10)
- Written in Python (2.7), RDKit (2014_03_1), PyQt (4.11.2), Matplotlib (1.4.0)
  - platform-independent (but written on and for Linux)
  - runs on Linux and Windows (Mac not tested)

Example Data Set

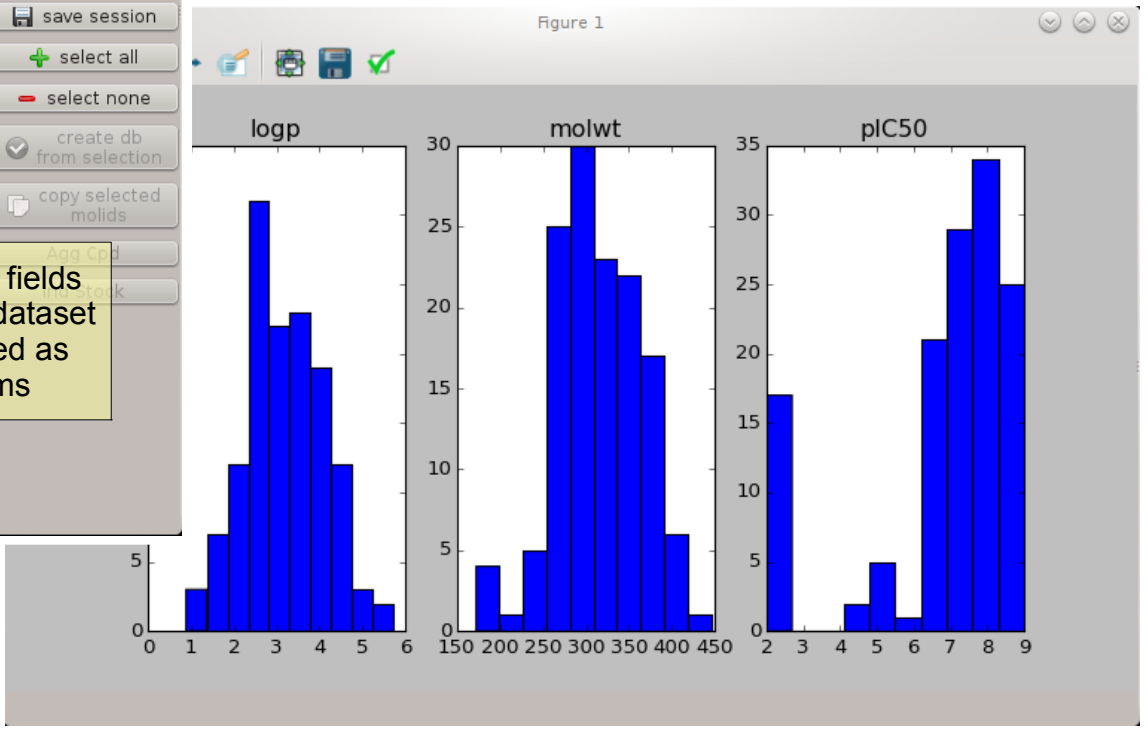
- Excerpt from the Sutherland Data Set
  J. Chem. Inf. Comput. Sci. 2003 (43) 1906 - 1915

  (as found on http://www.cheminformatics.org/datasets/#qsar)

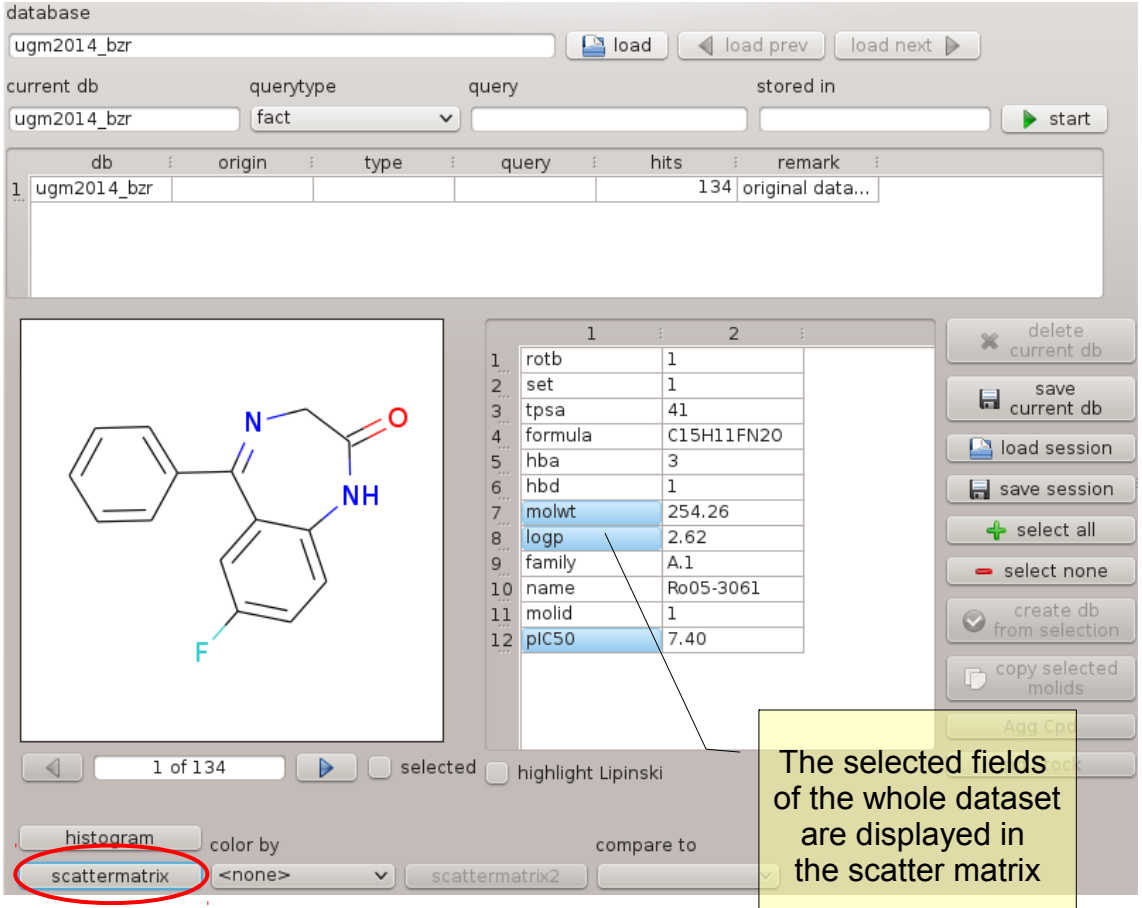
Histograms - For a Quick Overview



Histograms

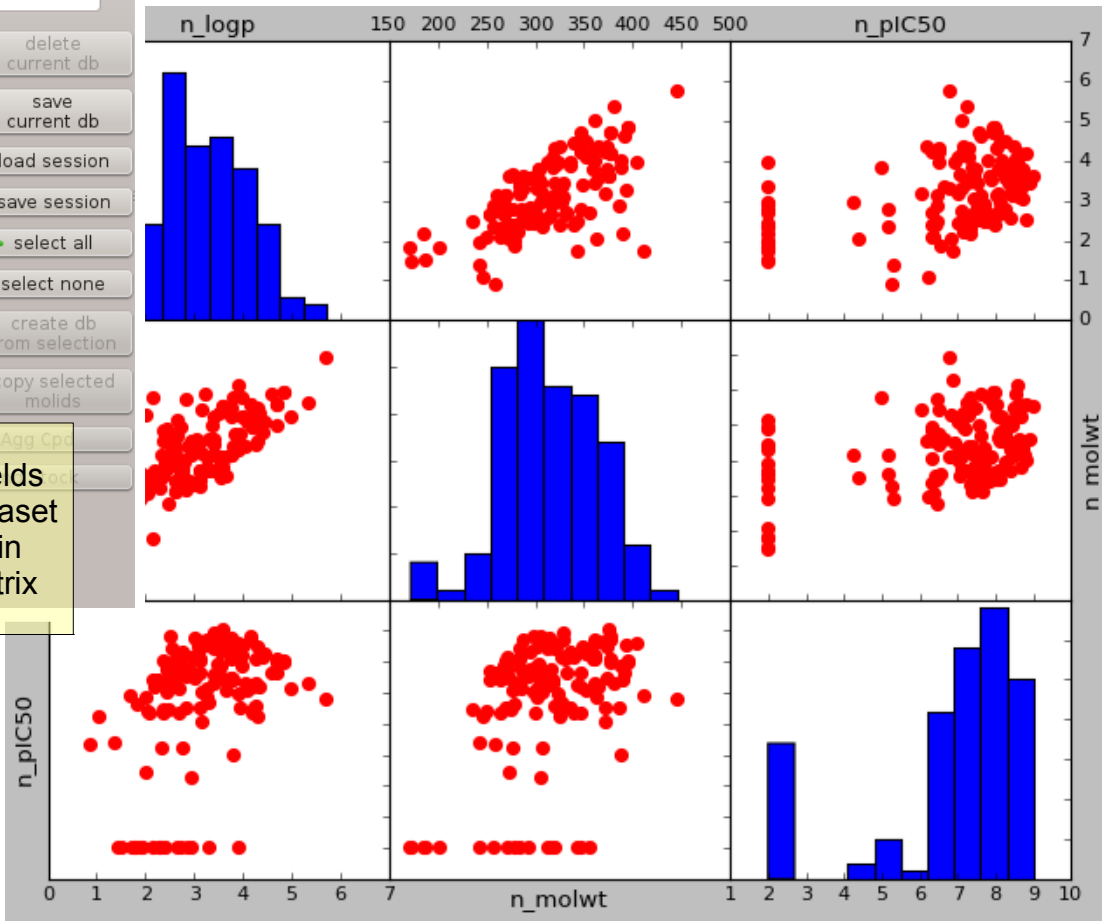


Scatter Matrix



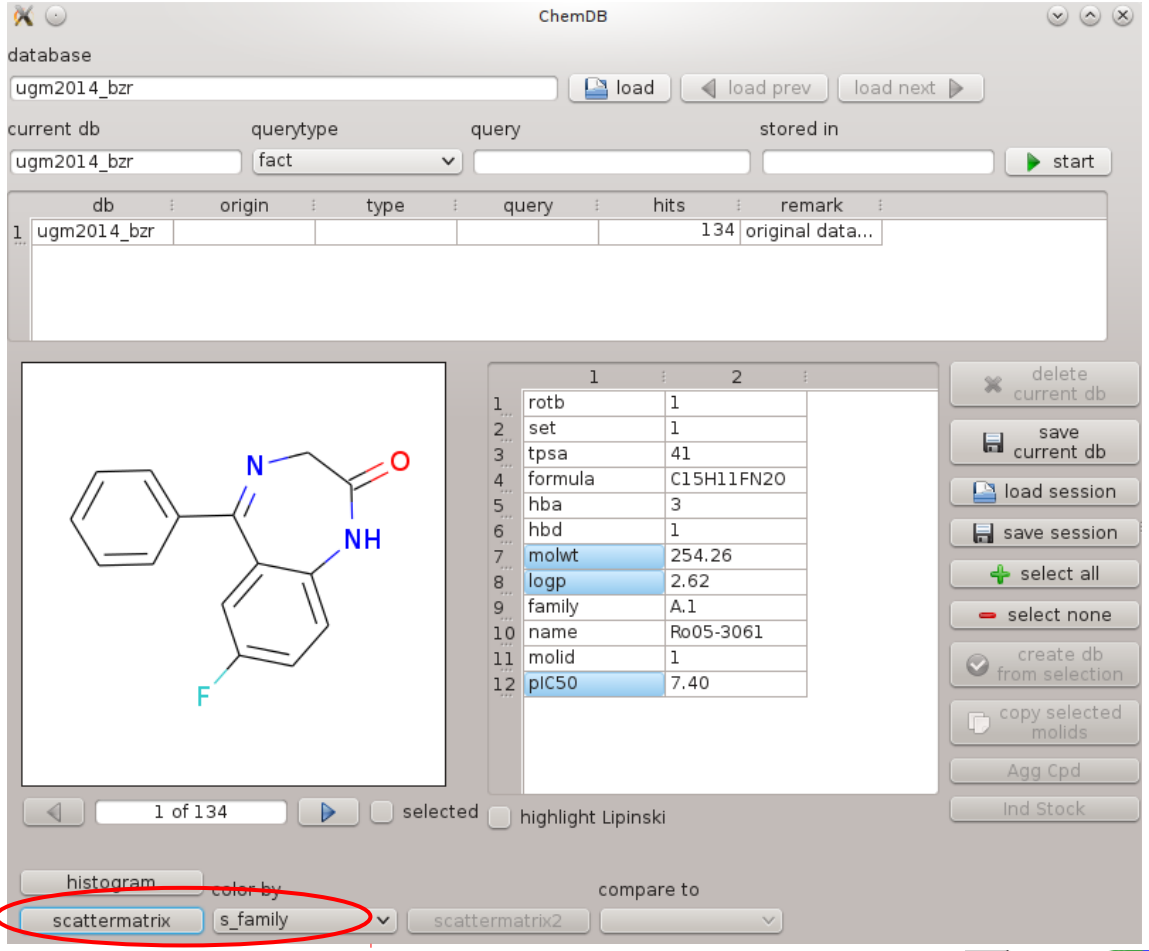
The selected fields of the whole dataset are displayed in the scatter matrix

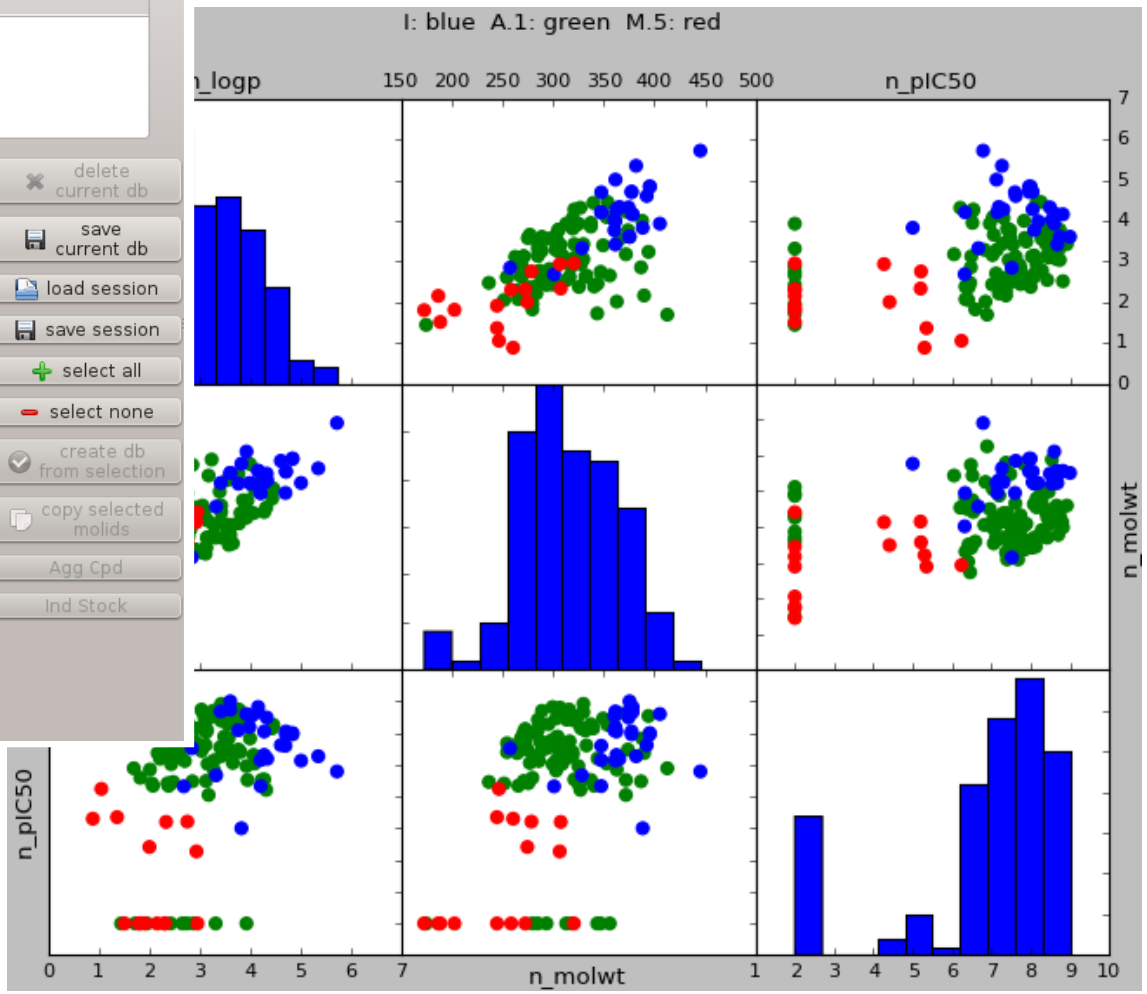
Scatter Matrix: all selected fields are plotted against each other, with histograms on the diagonale



Scatter Matrix - Color by Property

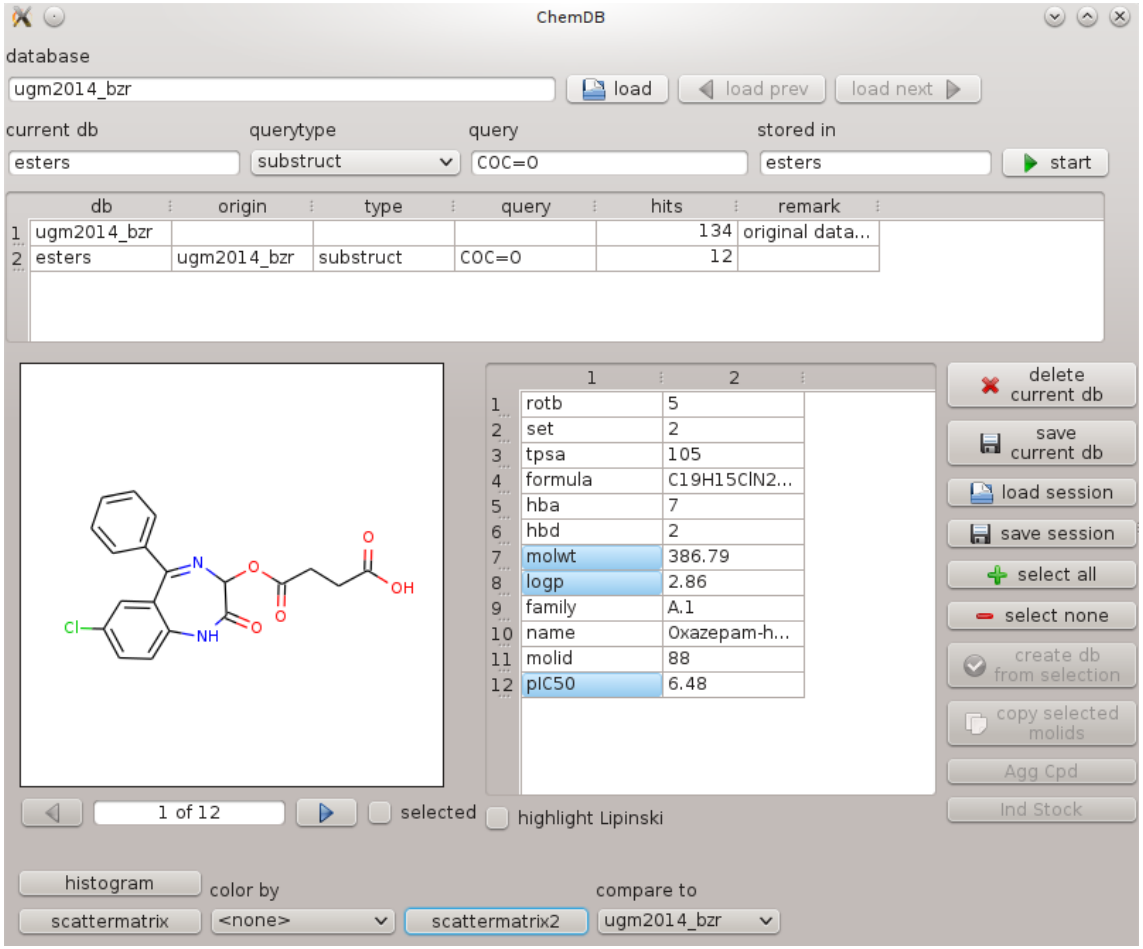
- color data points by category properties (e.g. compound family)

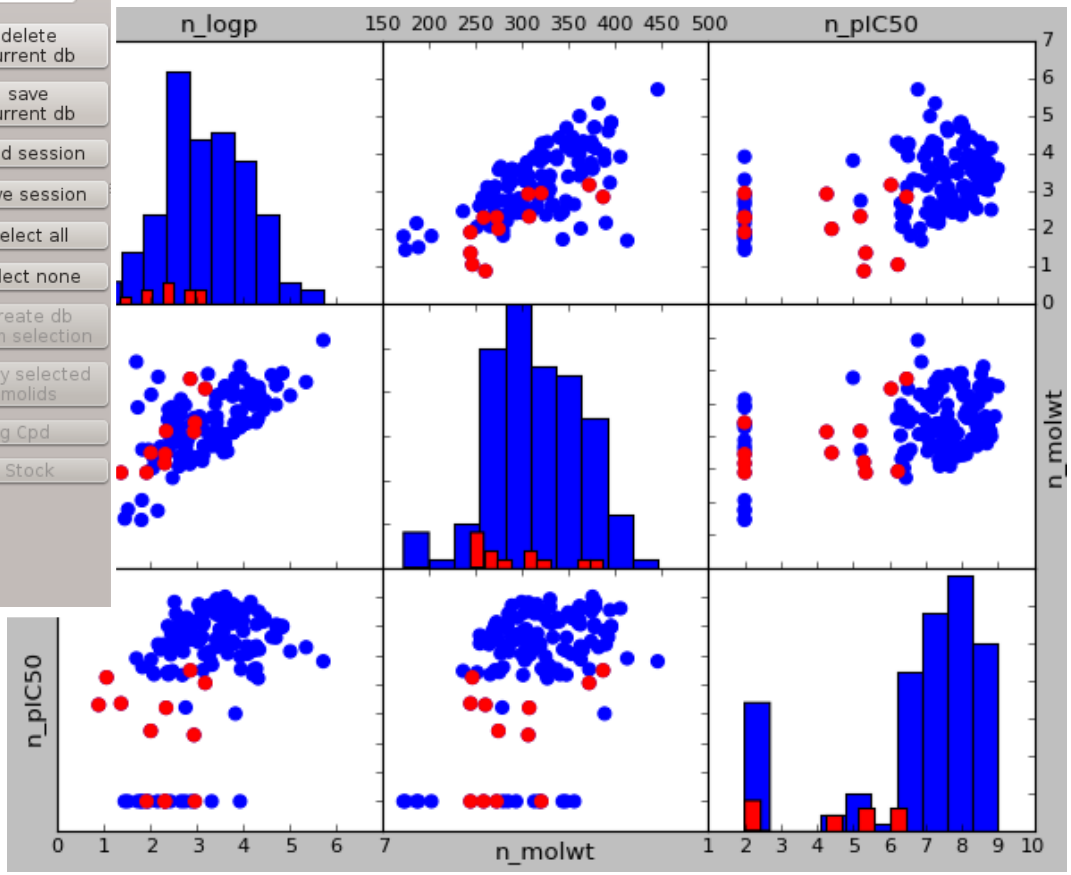




Compare a Subset to the Original SD File

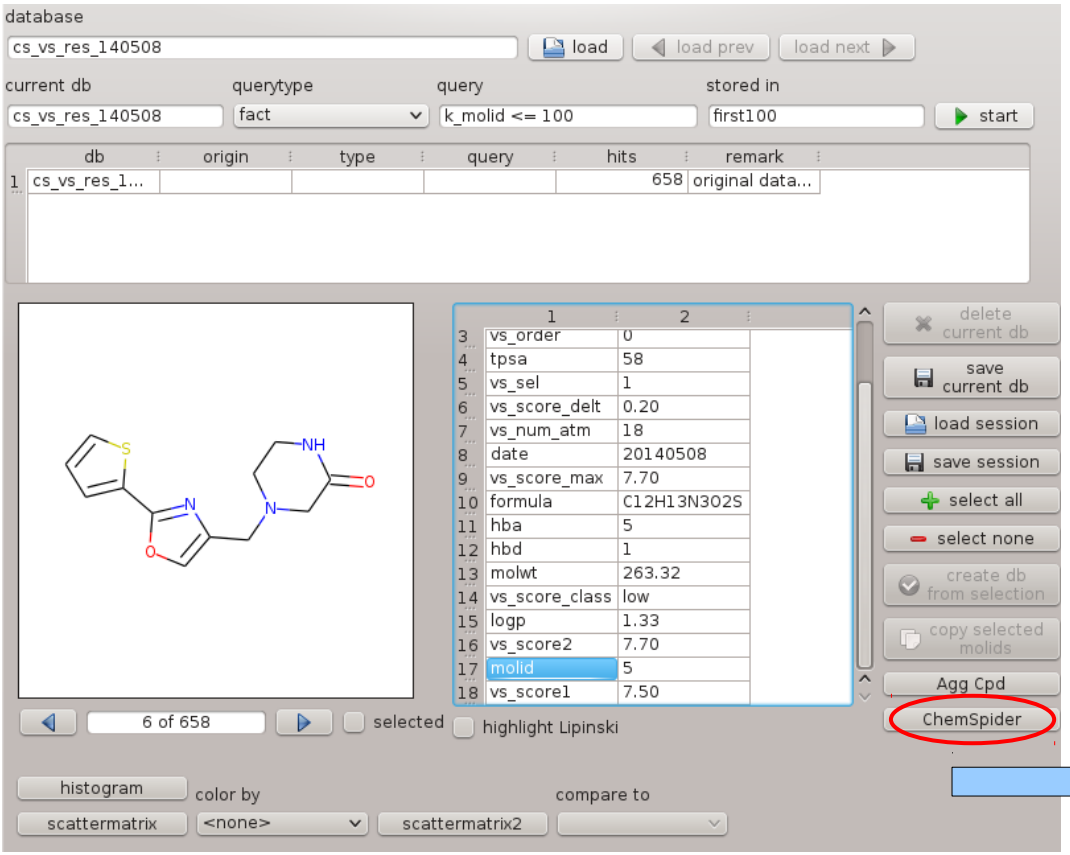
- compare a subset (e.g. from a substructure search) to the whole SD file graphically

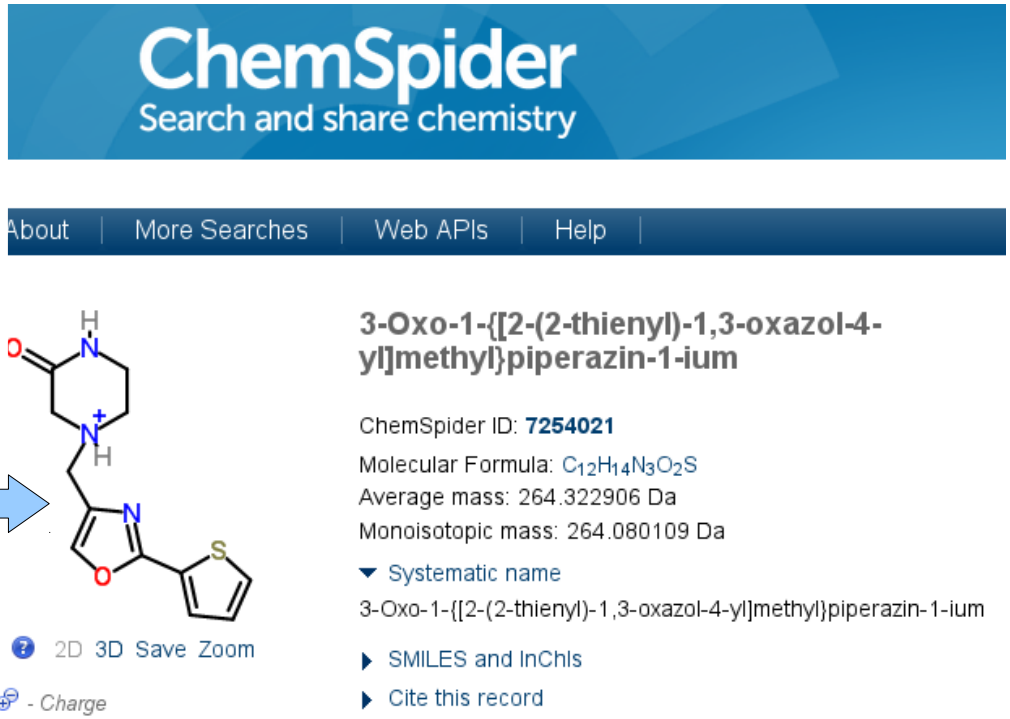




Link to Online Records

- special behaviour for SD files saved from Aldrich or Chemspider
- clicking the button opens the Chemspider (or Aldrich) record in a browser





sdf_tools Provides a Set of Convenience Functions

sdf_tools.py:

```
def load_db(fn="testset.sdf", large_db=False):
```

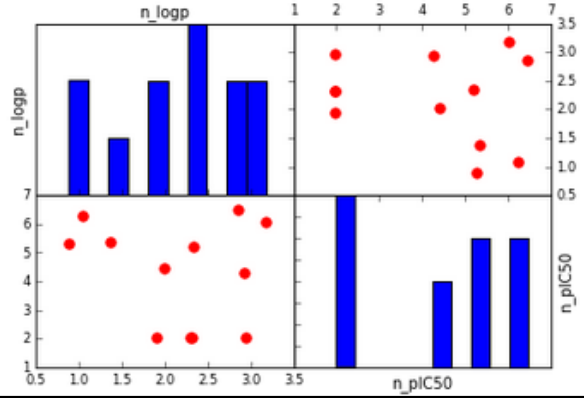
- returns the SD file as a list of RDKit Mol objects
- large SD files can be returned as file objects
- most other functions in the module can work with both so that arbitrarily large SD files can be handled
  a typical scenario is to perform a search on a large SD file as a file object and handle the smaller search result
  as list in memory. A substructure search in 120 000 compounds (315MB file) takes about 70s on my laptop.

```
def write_db(db_list, fn, conf_id=-1):
def list_fields(db_list_or_file):
def show_db(db_list, force=False):
def merge_prop_from_file(db_list, fn, prop):
def remove_props(mol_or_db, props):
def calc_props(mol_or_db, counterprop="k_molid", dateprop="k_date", include_date=True, force2d=False):
def sort_db(db_list, field, reverse=True):
def factsearch(db_list_or_file, query, invert=False, max_hits=2000, count_only=False, sorted=True,
               reverse=True):
def substruct_search(db_list_or_file, smarts, invert=False, max_hits=5000, count_only=False, add_h=False):
def similarity_search(db_list_or_file, smarts, similarity=0.8, max_hits=2000, count_only=False):

… and more
```

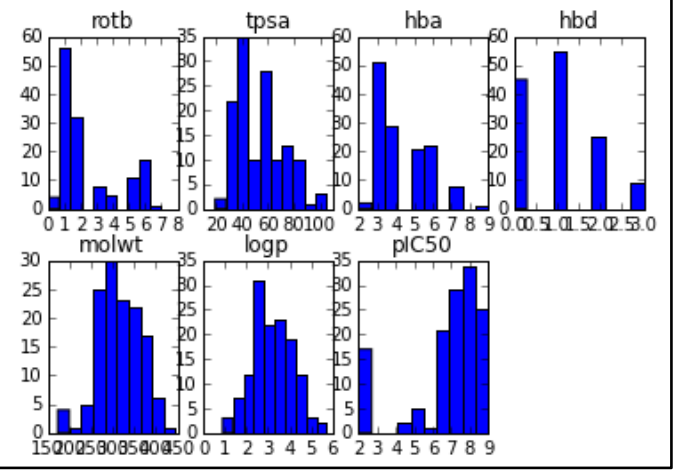
- Can be used e.g. in an IPython Session
- The viewer uses a subset of these functions
- Functions that return the same number of records (sorting, renaming fields, …) modify the list in place
- Functions that modify the number of records (e.g. substructure search) return a new list
- Plotting of properties is possible just like in the viewer (but without interactivity)

Example of Plotting With sdf_tools in the IPython Notebook

```
In [1]: from __future__ import absolute_import, division, print_function

        from rdkit.Chem import AllChem as Chem
        from rdkit.Chem import Draw
        from rdkit.Chem.Draw import IPythonConsole
        Draw.DrawingOptions.atomLabelFontSize = 18

In [2]: import sdf_tools as sdft

          > interactive session, trying to import display_png from IPython... success!
          > found environment var /home/apl/aviru/db_reports

In [3]: %pylab inline

        Populating the interactive namespace from numpy and matplotlib

In [4]: db = sdft.load_db("sdf/testset.sdf")

          > db sdf/testset loaded with 134 records.

In [5]: esters = sdft.substruct_search(db, "COC=O")

          > searching...
            processed:     134   Found:     12
            done.

In [7]: sdft.show_scattermatrix(esters, ["n_pIC50", "n_logp"])
```




```
In [8]: sdft.show_hist(db)
```



Interesting Bits and Caveats

- The images of the structures are generated on-the-fly
  - stored as a temporary png file
  - and displayed in a QtGui.QLabel as a QtGui.QPixmap

```
with tempfile.NamedTemporaryFile() as f:
    Draw.MolToFile(self.curr_db[self.curr_db_mol_index],
                   f.name, imageType="png")
    self.label_molimage.setPixmap(QtGui.QPixmap(f.name))
```

Interesting Bits and Caveats

- The Matplotlib data point picker and selector (*onpick*, next slide)
  - for the data picker to work in the viewer, it has to be taken care of that usually not every compound is tested in every assay
    - for this a dict of molids is kept per individual plot in the scatter matrix (self.axes_molindex_dict)

The Matplotlib Data Point Picker and Selector

```
class App(QtGui.QMainWindow, Ui_MainWindow):
…
    def onpick(self, event):
        x = event.mouseevent.xdata
        y = event.mouseevent.ydata
        ind = event.ind
        ax = event.mouseevent.inaxes
        if ind:
            ind = ind[0]
            if self.curr_db_index != 0:
                # use root database to display structures
                self.init_curr_db(self.db_name_order[0])

            molid = self.axes_molindex_dict[ax][ind]
            molindex = self.db_mol_index[molid]

            if self.curr_db_mol_index != self.db_mol_index[molid]:
                self.curr_db_mol_index = molindex
                print "x= %s, y= %s" % (x, y)
                print "index %2d: molid %s" %(molindex, molid)
                self.display_mol()

            else:
                # toggle the selection checkbox only on the second click on the data point:
                # self.curr_db_mol_index == self.db_mol_index[molid]
                # (this automagically triggers on_check_rec_selected_stateChanged)
                self.check_rec_selected.setChecked(not self.check_rec_selected.isChecked())

    @QtCore.pyqtSlot()
    def on_btn_scatter_clicked(self):
        colorby = str(self.combo_colorby.currentText())
        print "colorby:", colorby
        if colorby == "<none>":
            colorby = None
        selected_fields = self.get_selected_fields()
        self.statusbar.showMessage("generating scatter matrix...", 2000)
        self.fig, self.axes_molindex_dict = qdb.show_scattermatrix(self.curr_db,
                                    fields=selected_fields, colorby=colorby, mode="gui")
        self.statusbar.showMessage("scatter matrix generated.", 2000)
        self.fig.canvas.mpl_connect('pick_event', self.onpick)
        self.fig.show()
```

Thanks

Questions ?

- Code will be uploaded to GitHub in the coming days (as soon as I have figured out how to do that)