L2_Visualization_filled

January 18, 2019

0.1 Import modules. Remember it is always good practice to do this at the beginning of a notebook.

If you don't have seaborn, you can install it with conda install seaborn

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
```

0.1.1 Use notebook magic to render matplotlib figures inline with notebook cells.

```
In [2]: %matplotlib inline
```

Let's begin!
We'll use pandas read_csv function to read in our data.

```
In [3]: df = pd.read_csv('HCEPDB/HCEPDB_moldata.csv')
```

Let's take a look at the data to make sure it looks right with head, and then look at the shape of the dataframe

```
In [4]: df.head()
```

```
Out[4]:         id                                         SMILES_str   stoich_str  \
        0   655365            C1C=CC=C1c1cc2[se]c3c4occc4c4nsnc4c3c2cn1  C18H9N3OSSe
        1  1245190  C1C=CC=C1c1cc2[se]c3c(ncc4ccccc34)c2c2=C[SiH2]...  C22H15NSeSi
        2    21847  C1C=c2ccc3c4c[nH]cc4c4c5[SiH2]C(=Cc5oc4c3c2=C1...   C24H17NOSi
        3    65553   [SiH2]1C=CC2=C1C=C([SiH2]2)C1=Cc2[se]ccc2[SiH]1  C12H12SeSi3
        4   720918        C1C=c2c3ccsc3c3[se]c4cc(oc4c3c2=C1)C1=CC=CC1   C20H12OSSe

              mass       pce       voc         jsc  e_homo_alpha  e_gap_alpha  \
        0  394.3151  5.161953  0.867601   91.567575     -5.467601     2.022944
        1  400.4135  5.261398  0.504824  160.401549     -5.104824     1.630750
        2  363.4903  0.000000  0.000000  197.474780     -4.539526     1.462158
        3  319.4448  6.138294  0.630274  149.887545     -5.230274     1.682250
        4  379.3398  1.991366  0.242119  126.581347     -4.842119     1.809439

           e_lumo_alpha                               tmp_smiles_str
        0     -3.444656    C1=CC=C(C1)c1cc2[se]c3c4occc4c4nsnc4c3c2cn1
```

```
1      -3.474074  C1=CC=C(C1)c1cc2[se]c3c(ncc4ccccc34)c2c2=C[SiH...
2      -3.077368  C1=CC=C(C1)C1=Cc2oc3c(c2[SiH2]1)c1c[nH]cc1c1cc...
3      -3.548025  C1=CC2=C([SiH2]1)C=C([SiH2]2)C1=Cc2[se]ccc2[Si...
4      -3.032680       C1=CC=C(C1)c1cc2[se]c3c4sccc4c4=CCC=c4c3c2o1
```

```python
In [5]: df.shape
```

```
Out[5]: (2322849, 11)
```

That's a lot of data. Let's take a random subsampling of the full dataframe to make playing with the data faster. This is something you may consider doing when you have large datasets and want to do data exploration. Pandas has a built-in method called sample that will do this for you.

```python
In [6]: df_sample = df.sample(frac=0.01)
```

```python
In [7]: df_sample.head()
```

```
Out[7]:               id                                         SMILES_str  \
        1071618  1533697           c1csc(n1)-c1cc2c3ccccc3c3ccoc3c2c2cscc12
        456904   1085872  [SiH2]1C=c2ccc3[nH]c-4c([SiH2]c5cc(sc-45)-c4cc...
        1879830   332597  c1cc2c(sc(-c3ncc(s3)-c3cncs3)c2s1)-c1scc2cc[se...
        396516   1262251                 C1C=Cc2csc(-c3cc4sc5C=CCc5c4[nH]3)c12
        423668    559687  c1cc([nH]c1-c1ccc(-c2ccccc2)c2c[nH]cc12)-c1ccc...

                  stoich_str      mass       pce       voc         jsc  e_homo_alpha  \
        1071618   C21H11NOS2  357.4559  3.388331  0.743792   70.110316     -5.343792
        456904   C19H14N2SSi2  358.5716  3.304591  0.356891  142.504759     -4.956891
        1879830   C18H8N2S5Se  491.5652  5.336153  0.584366  140.537136     -5.184366
        396516     C16H11NS2  281.4019  0.640926  0.242102   40.743345     -4.842102
        423668     C24H16N4S  392.4844  4.002914  0.280494  219.634163     -4.880494

                 e_gap_alpha  e_lumo_alpha  \
        1071618     2.192823     -3.150969
        456904      1.721582     -3.235309
        1879830     1.733909     -3.450457
        396516      2.498178     -2.343924
        423668      1.376497     -3.503998

                                                   tmp_smiles_str
        1071618         c1cc2c(o1)c1c3cscc3c(cc1c1ccccc21)-c1nccs1
        456904   [nH]1c2-c3sc(cc3[SiH2]c2c2c1ccc1=C[SiH2]C=c21)...
        1879830  c1ncc(s1)-c1cnc(s1)-c1sc(-c2scc3cc[se]c23)c2cc...
        396516                [nH]1c(cc2sc3C=CCc3c12)-c1scc2C=CCc12
        423668   [nH]1c(ccc1-c1cccc2nsnc12)-c1ccc(-c2ccccc2)c2c...
```

```python
In [8]: df_sample.shape
```

```
Out[8]: (23228, 11)
```

We can use this to try some of our plotting functions. We will start with two variables in the dataset, PCE and HOMO energy.

There are multiple packages you can use for plotting. Pandas has some built-in object-oriented methods we can try first.

```
In [ ]: df.plot.scatter('pce', 'e_homo_alpha')
```

Oops! We used the wrong dataset. The full dataset took a while to plot. We can use %%timeit to see how long that took.

```
In [ ]: %%timeit -n 1 -r 1
        df.plot.scatter('pce', 'e_homo_alpha')
```

Note that %%timeit repeats the function call a number of times and averages it. You can alter this behavior by changing the defaults. Let's see how long it takes to plot our subsample:

```
In [ ]: %%timeit -n 1 -r 7
        df_sample.plot.scatter('pce', 'e_homo_alpha')
```

That's a lot quicker! It doesn't scale perfectly with datasize (plotting took about 1/5 of the time with 1/10 of the data) likely due to code overhead.

But the default plot settings are pretty ugly. We can take advantage of the object-oriented nature of pandas plots to modify the output.

```
In [ ]: p_v_hplot = df_sample.plot.scatter('pce', 'e_homo_alpha')
        p_v_hplot.set_xlabel('PCE')
        p_v_hplot.set_ylabel('HOMO')
        p_v_hplot.set_title('Photoconversion Efficiency vs. HOMO energy')
```

That's a bit butter, but we can still make improvements, like adding gridlines, making the y-axis label more accurate, increasing size, and adjusting the aspect ratio.

```
In [ ]: p_v_hplot = df_sample.plot.scatter('pce', 'e_homo_alpha', figsize=(6, 6))
        p_v_hplot.set_xlabel('PCE')
        p_v_hplot.set_ylabel('$E_{HOMO}$')
        p_v_hplot.set_title('Photoconversion Efficiency vs. HOMO energy')
        p_v_hplot.grid()
```

Note that we used LaTeX notation to create the subscript text. LaTeX can be used to generate mathematical expressions, symbols, and Greek letters for figures. One reference guide is included here: https://www.overleaf.com/learn/latex/Subscripts_and_superscripts

Take a moment to try to figure out the following using the pandas documentation: * How to change the x range to be 2 to 10 * How to change the y range to be -6 to 2 * How to change the font size to 18 * how to change the colors and transparency.

You can access the documentation here.

```
In [ ]:
```

0.1.2 An aside: Matplotlib can also be used to plot datasets in a similar fashion

Pandas visualization toolbox is a convenience feature built on top of Matplotlib.

```
In [ ]: p_v_hplot = plt.figure(figsize=(6, 6))
        p_v_hplot.subplots_adjust(hspace=0.5)
        ax1, ax2 = p_v_hplot.add_subplot(211), p_v_hplot.add_subplot(212)
        ax1.scatter(df_sample['pce'], df_sample['e_homo_alpha'], alpha=0.1)
        ax2.scatter(df_sample['pce'], df_sample['e_gap_alpha'], alpha=0.1)

        ax1.set_xlabel('PCE')
        ax1.set_ylabel('$E_{HOMO}$')
        ax1.set_title('Photoconversion Efficiency vs. HOMO energy')
        ax1.grid()

        ax2.set_xlabel('PCE')
        ax2.set_ylabel('$E_{GAP}$')
        ax2.set_title('Photoconversion Efficiency vs. gap energy')
        ax2.grid()

        plt.show()
```

Note that pandas can also be used like matplotlib to create subplots. It just has a slightly different notation:

```
In [ ]: fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(6, 6))
        df_sample.plot(x='pce', y='e_homo_alpha', ax=axes[0], kind='scatter', alpha=0.1)
        df_sample.plot(x='pce', y='e_gap_alpha', ax=axes[1], kind='scatter', alpha=0.1)
        axes[0].grid()
        axes[1].grid()
        plt.show()
```

0.1.3 Back to pandas: Quick dataset exploration tools

A very useful tool for quickly exploring relationships between variables in a dataset is the built-in pandas scatterplot matrix:

```
In [ ]: from pandas.plotting import scatter_matrix
        scatter_matrix(df_sample, figsize=(10, 10), alpha=0.1)
```

That's a lot of information in one figure! Note the funky id plot at the left. IDs are the molecule ids and don't contain any useful information. Let's make that a column index before moving on.

```
In [ ]: df_sample.set_index('id', inplace=True)
```

```
In [ ]: df_sample.head()
```

OK, let's move on to density plots. These show the probability density of particular values for a variable. Notice how we used an alternate way of specifying plot type.

```
In [ ]: df_sample['pce'].plot(kind='kde')
```

We can plot two different visualizations on top of each other, for instance, the density plot and a histogram plot. Since the density plot has a different y axis than the density plot, make sure to use a secondary y axis

```
In [ ]: ax = df_sample['pce'].plot(kind='hist')
        df_sample['pce'].plot(kind='kde', ax=ax, secondary_y=True)
```

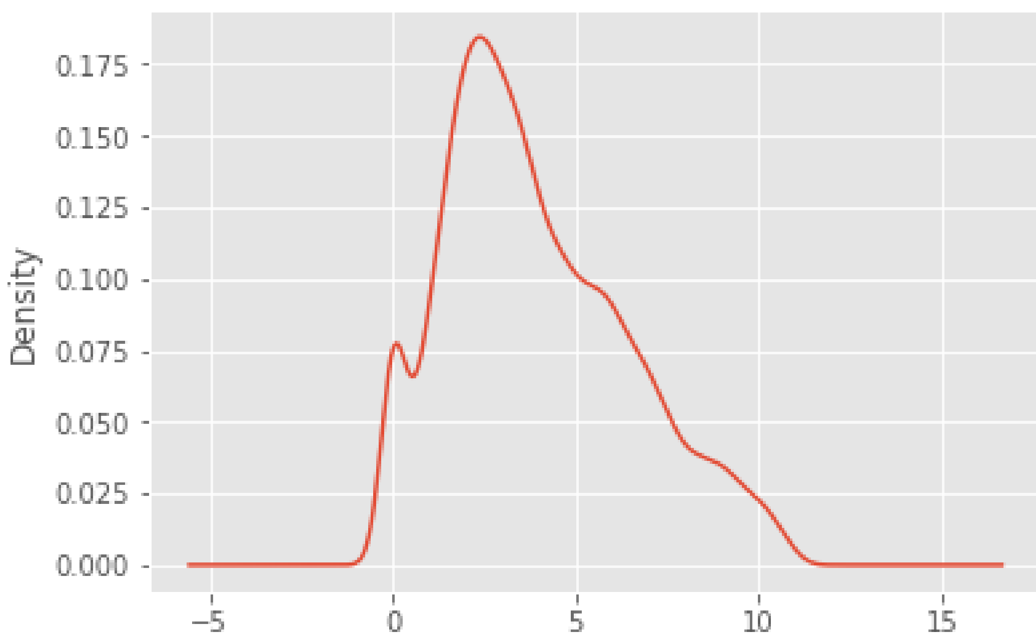
0.1.4 Alternate plot styles

As pandas is built on Matplotlib, you can use Matplotlib to alter then plot style. Styles are essentially a set of defaults for the plot appearance, so you don't have to modify them all yourselves. Let's try the ggplot style that mimics the ggplot2 style output from R.

```
In [9]: import matplotlib
        matplotlib.style.use('ggplot')
```

```
In [10]: df_sample['pce'].plot(kind='kde')
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x13a18a0d0f0>
```



You can find the list of matplotlib styles here

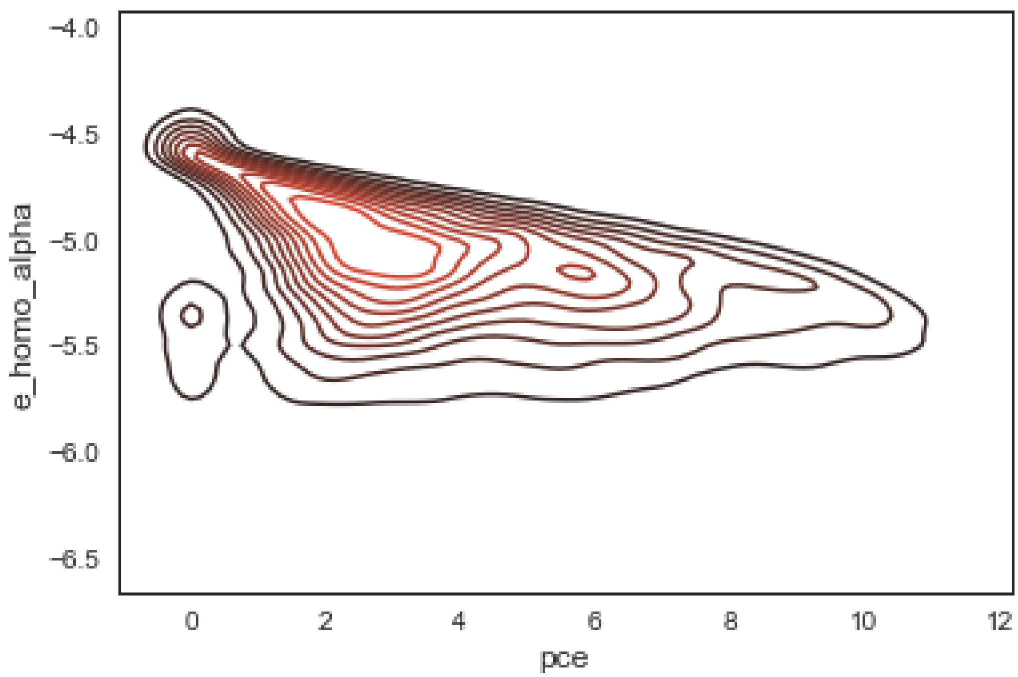
0.1.5 Seaborn improvements

Matplotlib can be used to create publication-quality images, but has some limitations-- including capabilities with 3D plots. There's another package Seaborn, that has a lot of built-in styles for very high-quality plots. Let's take a look at some of the options available:

```
In [11]: sns.set_style('white')
         sns.kdeplot(df_sample['pce'], df_sample['e_homo_alpha'])
```

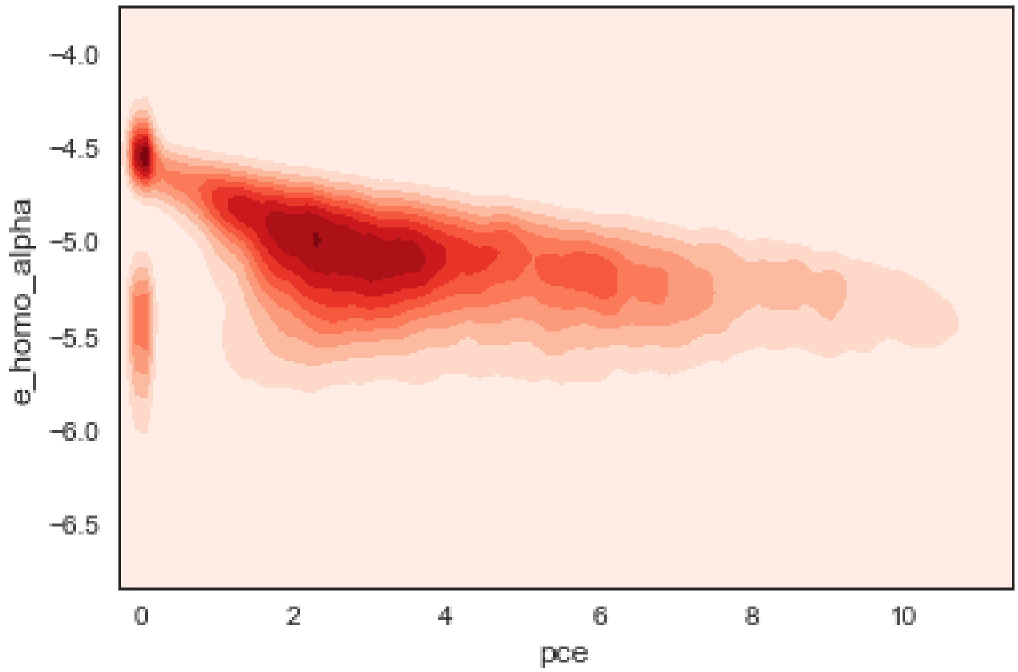
```
c:\users\koolk\anaconda3\envs\chad\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning:
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x13a18d05f28>
```




```
In [12]: sns.kdeplot(df_sample['pce'], df_sample['e_homo_alpha'], cmap='Reds',
                     shade=True, bw=0.1)
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x13a18e2f3c8>
```



0.1.6 In class exercise

Fix the above subplots so they aren't as shoddy. Add titles, increase font size, change colors and alpha, and change the margins and layout so they are side by side.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: