Numerical Methods - Assignment 1

In [ ]:

1. Python and MATLAB

List at least 3 differences between Python and MATLAB.

Python is free and open source, meaning that you can manipulate other people's code to do what you want, and it's free so you do not need to spend money on licences like you need to with Matlab.

Python is not as fast as Matlab when working with computationally expensive numerical methods, which is something Matlab is used for, and why it was most likely a good option for the Numerical Methods ChBE course.

Matlab automatically comes with an IDE (integrated development environment) which is really helpful to those new to it to be able to debug and see what variables are valued at in certain points, for Python you would have to download the Spyder IDE that allows you to do this. Which is really cool because earlier last year I tried picking up Python and it would have been a lot better if I knew there was this IDE feature and I wouldn't just have to work in something that looks like a command prompt window! The fact that I couldn't find this I guess also goes to show the different versions of Python that are hard to manage, like referenced in the notes.

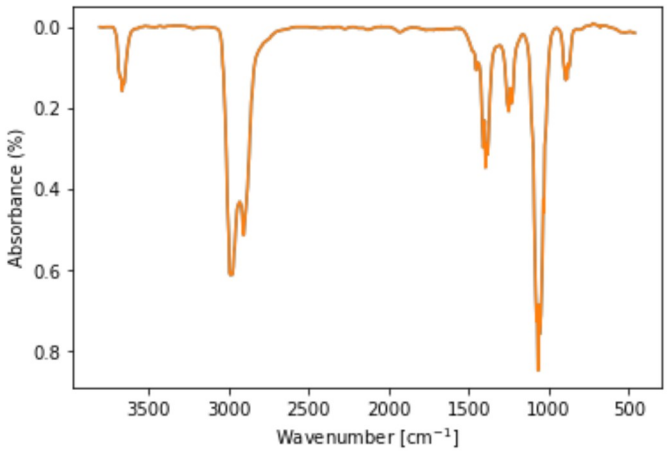
2. Plot Data

Read the data and create a plot.

- Import `matplotlib` and `pandas` packages.
- Read in `data/ethanol_IR.csv` file and create a plot of IR spectra data.

In [1]:
```
import pylab as plt
import pandas as pd
%matplotlib inline
etoh_IR = pd.read_csv('data/ethanol_IR.csv')

wavenum = pd.DataFrame(etoh_IR, columns= ['wavenumber [cm^-1]'])
absorb = pd.DataFrame(etoh_IR, columns = ['absorbance'])

fig, ax = plt.subplots(); ax.plot(wavenum,absorb)
ax.invert_xaxis()
ax.invert_yaxis()
ax.set_xlabel('Wavenumber [cm$^{-1}$]')
ax.set_ylabel('Absorbance (%)')
plt.plot(wavenum,absorb)
```

Out[1]:
```
[<matplotlib.lines.Line2D at 0x1baeaae7760>]
```



Briefly describe the most prominent peaks in the dataset.

The peaks around 3000 cm^-1 show Carbon-Hydrogen stretching, the one slightly above 3000 cm^-1 is indicative of sp2 hybridization and the part of the peak below 3000^-1 is indicative of sp3 hybridization. The strong peak near 1000 cm^-1 is indicative of Carbon-Oxygen stretching, so this structure could have an ether or be classified just as an ether.

3. Matrix-vector Multiplication

Write a funcion that uses `for` loops.

This function should multiply an arbitrary matrix and vector.

In [2]:
```
import numpy as np
def mulMatVec(matrix, vector):
    result = []
    for row in matrix:
        dotprod = np.dot(row,vector)
        result.append(dotprod)
    return result
```

You can use the matrix and vector given below.

In [3]:
```
A = np.array([[1, 2], [-4, 5]])
B = np.array([-2, 3])
print(mulMatVec(A,B))
```

```
[4, 23]
```

Or create an arbitrary set of matrix and vector using `numpy.random.rand` .

In [4]:
```
#This was just for practice, please grade the one above!

numrows = np.random.randint(1,high = 5)
numcols = np.random.randint(1,high = 5)
matrix = []
for i in range(numrows):
    row = np.random.randint(1,high = 7,size = numcols)
    matrix.append(row)
vector = np.random.randint(1, high = 8, size = numcols)
print(matrix)
print(vector)
print(mulMatVec(matrix, vector))
```

```
[array([6, 6]), array([2, 5]), array([1, 3]), array([1, 4])]
[1 7]
[48, 37, 22, 29]
```

Show that your function is correct using `numpy.isclose` .

In [5]:
```
print(np.isclose(mulMatVec(A,B),[4,23]))
```

```
[ True  True]
```

4. Vandermonde Matrix

Use `numpy.hstack` to construct a 4th-order Vandermonde matrix.

Range should be from -1 to 1 with a resolution of 25 (i.e. the number of rows should be 25).

In [6]:
```
import numpy as np
resolution = 25
xi = np.linspace(-1,1,resolution)
xi = xi.reshape(-1,1)
xvdm = np.hstack((xi**0,xi**1,xi**2,xi**3,xi**4))
print(xvdm)
```

```
[[ 1.00000000e+00 -1.00000000e+00  1.00000000e+00 -1.00000000e+00
   1.00000000e+00]
 [ 1.00000000e+00 -9.16666667e-01  8.40277778e-01 -7.70254630e-01
   7.06066744e-01]
 [ 1.00000000e+00 -8.33333333e-01  6.94444444e-01 -5.78703704e-01
   4.82253086e-01]
 [ 1.00000000e+00 -7.50000000e-01  5.62500000e-01 -4.21875000e-01
   3.16406250e-01]
 [ 1.00000000e+00 -6.66666667e-01  4.44444444e-01 -2.96296296e-01
   1.97530864e-01]
 [ 1.00000000e+00 -5.83333333e-01  3.40277778e-01 -1.98495370e-01
   1.15788966e-01]
 [ 1.00000000e+00 -5.00000000e-01  2.50000000e-01 -1.25000000e-01
   6.25000000e-02]
 [ 1.00000000e+00 -4.16666667e-01  1.73611111e-01 -7.23379630e-02
   3.01408179e-02]
 [ 1.00000000e+00 -3.33333333e-01  1.11111111e-01 -3.70370370e-02
   1.23456790e-02]
 [ 1.00000000e+00 -2.50000000e-01  6.25000000e-02 -1.56250000e-02
   3.90625000e-03]
 [ 1.00000000e+00 -1.66666667e-01  2.77777778e-02 -4.62962963e-03
   7.71604938e-04]
 [ 1.00000000e+00 -8.33333333e-02  6.94444444e-03 -5.78703704e-04
   4.82253086e-05]
 [ 1.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00]
 [ 1.00000000e+00  8.33333333e-02  6.94444444e-03  5.78703704e-04
   4.82253086e-05]
 [ 1.00000000e+00  1.66666667e-01  2.77777778e-02  4.62962963e-03
   7.71604938e-04]
 [ 1.00000000e+00  2.50000000e-01  6.25000000e-02  1.56250000e-02
   3.90625000e-03]
 [ 1.00000000e+00  3.33333333e-01  1.11111111e-01  3.70370370e-02
   1.23456790e-02]
 [ 1.00000000e+00  4.16666667e-01  1.73611111e-01  7.23379630e-02
   3.01408179e-02]
 [ 1.00000000e+00  5.00000000e-01  2.50000000e-01  1.25000000e-01
   6.25000000e-02]
 [ 1.00000000e+00  5.83333333e-01  3.40277778e-01  1.98495370e-01
   1.15788966e-01]
 [ 1.00000000e+00  6.66666667e-01  4.44444444e-01  2.96296296e-01
   1.97530864e-01]
 [ 1.00000000e+00  7.50000000e-01  5.62500000e-01  4.21875000e-01
   3.16406250e-01]
 [ 1.00000000e+00  8.33333333e-01  6.94444444e-01  5.78703704e-01
   4.82253086e-01]
 [ 1.00000000e+00  9.16666667e-01  8.40277778e-01  7.70254630e-01
   7.06066744e-01]
 [ 1.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00
   1.00000000e+00]]
```

Create an orthonormal version of the Vandermonde matrix.

Orthonormal means:

- the L_2 norm of each column is 1.
- the inner product between any 2 columns is 0.

Print the orthonormalized Vandermonde matrix.

In [7]:
```
numcols = np.size(xvdm[0,:])
vec = list(range(0,5))

for i in vec:
    if i==0:
        xvdm[:,i] = (xvdm[:,i])/(np.linalg.norm(xvdm[:,i]))
    j = i-i
    while j != i:
        ortho = xvdm[:,i] - np.dot(xvdm[:,j],xvdm[:,i])*xvdm[:,j]
        xvdm[:,i] = ortho
        j = j+1
    xvdm[:,i] = xvdm[:,i]/(np.linalg.norm(xvdm[:,i],2))
print(xvdm)
```

```
[[ 2.00000000e-01 -3.32820118e-01  3.96566460e-01 -4.15922412e-01
   4.01324069e-01]
 [ 2.00000000e-01 -3.05085108e-01  2.97424845e-01 -2.07961206e-01
   6.68873448e-02]
 [ 2.00000000e-01 -2.77350098e-01  2.06904240e-01 -4.52089579e-02
  -1.36682835e-01]
 [ 2.00000000e-01 -2.49615088e-01  1.25004645e-01  7.64442378e-02
  -2.37146040e-01]
 [ 2.00000000e-01 -2.21880078e-01  5.17260600e-02  1.61108286e-01
  -2.59618073e-01]
 [ 2.00000000e-01 -1.94145069e-01 -1.29315150e-02  2.12893092e-01
  -2.26570966e-01]
 [ 2.00000000e-01 -1.66410059e-01 -6.89680800e-02  2.35908562e-01
  -1.57832983e-01]
 [ 2.00000000e-01 -1.38675049e-01 -1.16383635e-01  2.34264600e-01
  -7.05886207e-02]
 [ 2.00000000e-01 -1.10940039e-01 -1.55178180e-01  2.12071111e-01
   2.06213948e-02]
 [ 2.00000000e-01 -8.32050294e-02 -1.85351715e-01  1.73438002e-01
   1.03900105e-01]
 [ 2.00000000e-01 -5.54700196e-02 -2.06904240e-01  1.22475177e-01
   1.69994319e-01]
 [ 2.00000000e-01 -2.77350098e-02 -2.19835755e-01  6.32925410e-02
   2.12294616e-01]
 [ 2.00000000e-01  1.66277051e-17 -2.24146260e-01 -3.18152729e-17
   2.26835343e-01]
 [ 2.00000000e-01  2.77350098e-02 -2.19835755e-01 -6.32925410e-02
   2.12294616e-01]
 [ 2.00000000e-01  5.54700196e-02 -2.06904240e-01 -1.22475177e-01
   1.69994319e-01]
 [ 2.00000000e-01  8.32050294e-02 -1.85351715e-01 -1.73438002e-01
   1.03900105e-01]
 [ 2.00000000e-01  1.10940039e-01 -1.55178180e-01 -2.12071111e-01
   2.06213948e-02]
 [ 2.00000000e-01  1.38675049e-01 -1.16383635e-01 -2.34264600e-01
  -7.05886207e-02]
 [ 2.00000000e-01  1.66410059e-01 -6.89680800e-02 -2.35908562e-01
  -1.57832983e-01]
 [ 2.00000000e-01  1.94145069e-01 -1.29315150e-02 -2.12893092e-01
  -2.26570966e-01]
 [ 2.00000000e-01  2.21880078e-01  5.17260600e-02 -1.61108286e-01
  -2.59618073e-01]
 [ 2.00000000e-01  2.49615088e-01  1.25004645e-01 -7.64442378e-02
  -2.37146040e-01]
 [ 2.00000000e-01  2.77350098e-01  2.06904240e-01  4.52089579e-02
  -1.36682835e-01]
 [ 2.00000000e-01  3.05085108e-01  2.97424845e-01  2.07961206e-01
   6.68873448e-02]
 [ 2.00000000e-01  3.32820118e-01  3.96566460e-01  4.15922412e-01
   4.01324069e-01]]
```

Show that the L_2 of 5th column is 1.

In [8]:
```
print(np.isclose(np.linalg.norm(xvdm[:,4],2),1))
```

```
True
```

Show that the inner product between 1st column & 4th column is 0.

In [9]:
```
print(np.isclose(np.dot(xvdm[:,0],xvdm[:,3]),0))
```

```
True
```

Compute the rank of the orthonormalized Vandermonde matrix.

In [10]:
```
xvdmrank = np.linalg.matrix_rank(xvdm)
print(xvdmrank)
```

```
5
```

Show that the rank is equal to the number of columns.

In [11]:
```
numcols = np.size(xvdm[0,:])
print(numcols == xvdmrank)
```

```
True
```

Change the resolution to 30 and show that the rank is independent of the number of rows.

In [12]:
```
resolution = 30
xi = np.linspace(-1,1,resolution)
xi = xi.reshape(-1,1)
xvdm = np.hstack((xi**0,xi**1,xi**2,xi**3,xi**4))

numcols = np.size(xvdm[0,:])
vec = list(range(0,5))

for i in vec:
    if i==0:
        xvdm[:,i] = (xvdm[:,i])/(np.linalg.norm(xvdm[:,i]))
    j = i-i
    while j != i:
        ortho = xvdm[:,i] - np.dot(xvdm[:,j],xvdm[:,i])*xvdm[:,j]
        xvdm[:,i] = ortho
        j = j+1
    xvdm[:,i] = xvdm[:,i]/(np.linalg.norm(xvdm[:,i],2))
xvdmrank = np.linalg.matrix_rank(xvdm)
numcols = np.size(xvdm[0,:])
print(numcols == xvdmrank)
```

```
True
```

Numerical Methods - Assignment 2

1. Gaussian Features

Write a function that creates a set of evenly-spaced Gaussian functions.

The input should be an vector x, a number of Gaussians N, and a fixed width σ.

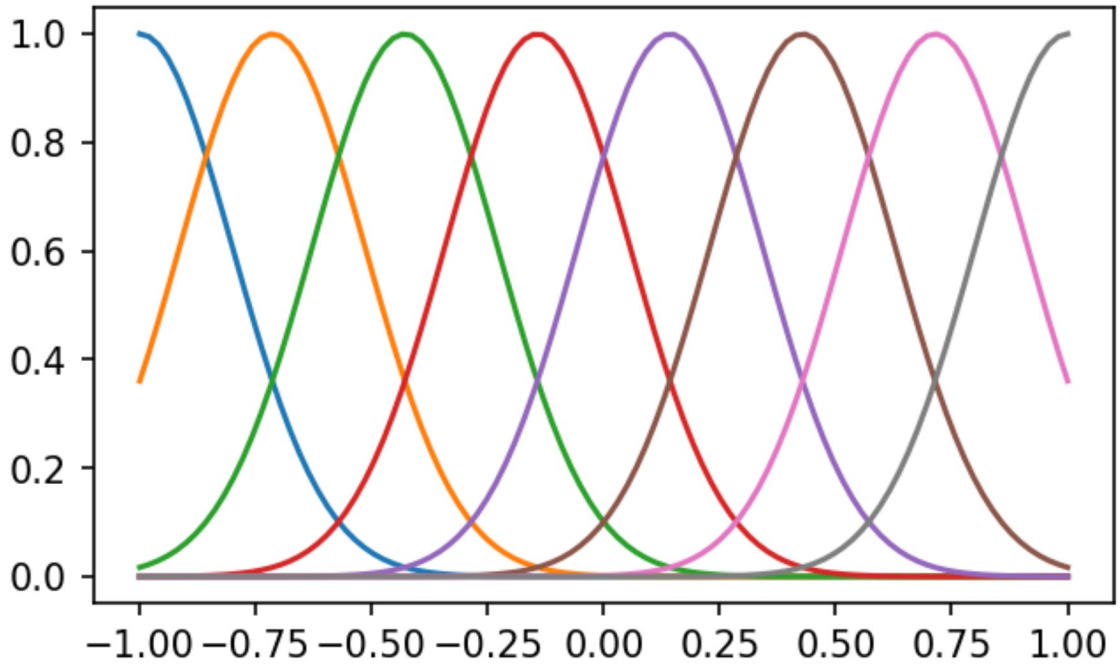
In [62]:
```
#making a function that will create as manuy gaussians as I need
#and will use this function later
import numpy as np
def gaussian_features(x, N, sigma):
    #this function assumes that the x input is a row vec
    #so we will reshape it into a col vec
    x = x.reshape(-1)
    #do not want to assign a mean for each peak, find evenly spaced gauss distrib
    #xvec is now the means of each of our gaussians
    xmeans = np.linspace(min(x),max(x),N) #do this for N number of gaussians
    features = []
    #do a for loop to do the gaussian eqn for each col of x and for N number gauss
    for xnow in xmeans:
        features.append(np.exp(-((x-xnow)**2)/(2*sigma**2)))
```

Use this function to plot 8 evenly-spaced Gaussians from -1 to 1 with a width of 0.2.

You can randomly define the resolution of the range.

In [63]:
```
import matplotlib.pyplot as plt
x = np.linspace(-1,1,100)#changing the number of the points to 100 made the curves
mygauss = gaussian_features(x,8,0.2)
fig, ax = plt.subplots(figsize = (5, 3), dpi = 150)
```

Out[63]:
```
[<matplotlib.lines.Line2D at 0x18d2fcb6640>,
 <matplotlib.lines.Line2D at 0x18d2f538fa0>,
 <matplotlib.lines.Line2D at 0x18d2f5383d0>,
 <matplotlib.lines.Line2D at 0x18d2f538400>,
 <matplotlib.lines.Line2D at 0x18d2f538d00>,
 <matplotlib.lines.Line2D at 0x18d2f538d90>,
 <matplotlib.lines.Line2D at 0x18d2f538610>,
 <matplotlib.lines.Line2D at 0x18d2f5380a0>]
```



2. General Linear Regression

Determine the best-fit of the peaks below using general linear regression.

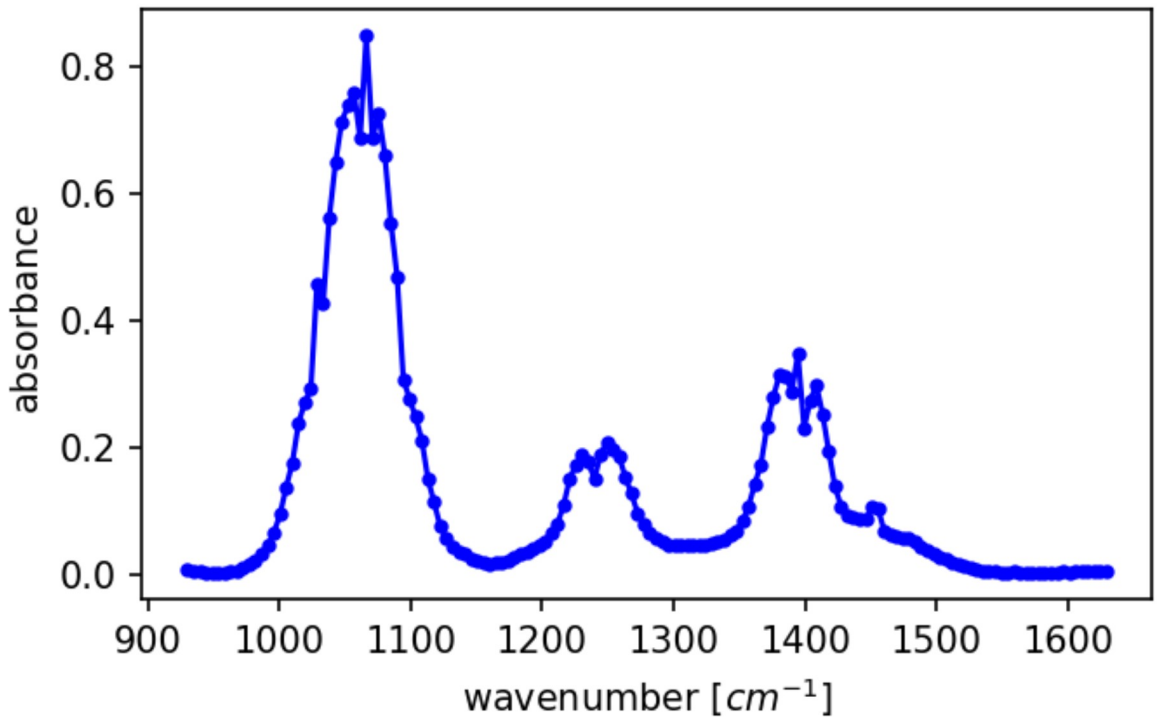
Plot the result of your regression model along with the original data.

You may assume that:

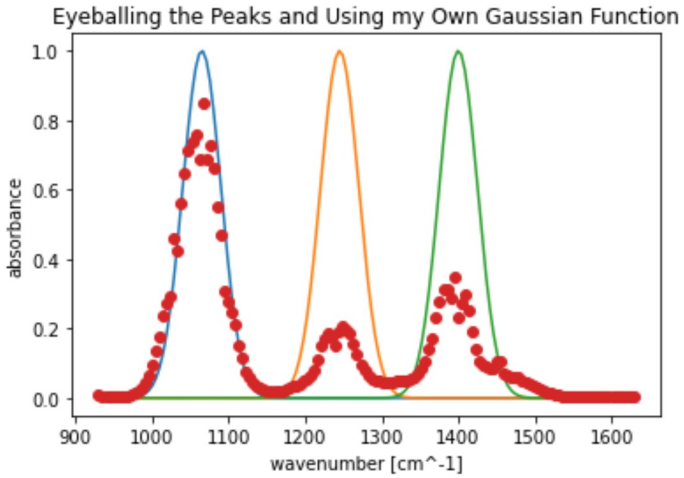
- The peaks follow a Gaussian distribution.
- There are 3 peaks of the **same width** in this region of the spectra below.

In [65]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data/ethanol_IR.csv')
x_all = df['wavenumber [cm^-1]'].values
y_all = df['absorbance'].values

x_peak = x_all[100:250]#just so they can the peaks for that range of the ethanol s
y_peak = y_all[100:250]

fig, ax = plt.subplots(figsize = (5, 3), dpi = 150)
```

```
ax.plot(x_peak, y_peak, '-b', marker = '.')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
ax.set_ylabel('absorbance');

#Follow Gaussian Distribution, so you have a gaussian basis matrix
#knowing they're the same width
```



In [66]:
```
import numpy as np
#1st thing I will do is eyeball the means and make my own gaussian
#using my own made gaussian functions
#mean guesses
mu1 = 1065
mu2 = 1245
mu3 = 1400
sig = 25
xgauss = np.zeros((len(x_peak),3)) #3 cols bc 3 peaks
#change the first col to be the gaussian eqn
xgauss[:,0] = np.exp(-(x_peak - mu1)**2/(2*(sig**2)))
#change the second col
xgauss[:,1] = np.exp(-(x_peak - mu2)**2/(2*(sig**2)))
#change the 3rd col
xgauss[:,2] = np.exp(-(x_peak - mu3)**2/(2*(sig**2)))

fig, ax = plt.subplots()
ax.plot(x_peak, xgauss[:,0])
ax.plot(x_peak, xgauss[:,1])
ax.plot(x_peak, xgauss[:,2])
ax.plot(x_peak, y_peak, 'o')
ax.set_title('Eyeballing the Peaks and Using my Own Gaussian Function')
ax.set_xlabel('wavenumber [cm^-1]')
ax.set_ylabel('absorbance');
```



Briefly describe the result.

The peaks shown here were made by eyeballing the means and the standard deviation, an then manually creating the gaussian vector myself and putting it together as one matrix, then I plotted each column together, where each column represents a Gaussian curve, and I made 3 of them as shown. These curves are at the same x values as the data and nearly the same width as I could get them, but I cannot get the heights of

Continue working on general linear regression.

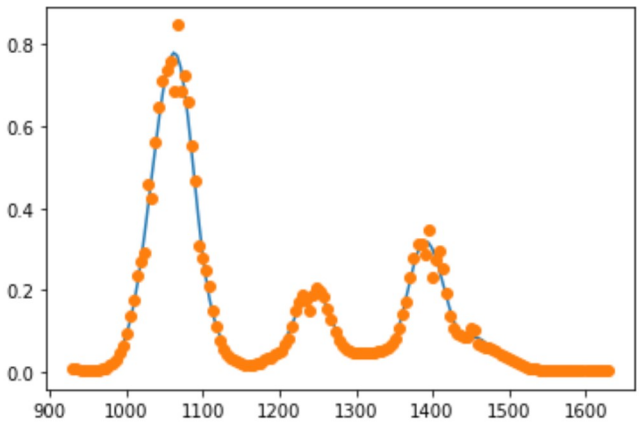
Now the second assumption is gone. You do not know how many peaks there are, or the widths of the peaks. However, you do know that they follow Gaussian distributions.

- Use your intuition and trial-and-error along to find a model that describes the data.
- Also plot the result along with the original data.

- This is not a spectroscopy class. There is no right answer to this question.

In [67]:
```
#since that first gaussian I made was really off from the data
#we will use the function I made for the gaussian function to make
#a better fit


m = 35 #this is the N number of Gaussians, arbitrarily guessed
#and checked to get tii this number 35
mygauss = gaussian_features(x_peak,m,25)
#need to get it again in Aw = b form, so create those components
A = np.dot(mygauss.T, mygauss)
b = np.dot(mygauss.T,y_peak)
w = np.linalg.solve(A,b)#solve the system
#now do the predictions
yhat = np.dot(mygauss,w)

#fig, ax = plt.subplots(figsize = (5, 3), dpi = 150)
fig, ax = plt.subplots()
ax.plot(x_peak, yhat)
ax.plot(x_peak, y_peak, 'o')
```

Out[67]:
```
[<matplotlib.lines.Line2D at 0x18d3313e160>]
```



3. Non-linear Regression

Write a loss function.

You want to solve the same problem above using non-linear regression to find the optimal positions and widths of the peaks.

The inputs of the loss function should be:

- a parameter vector $\vec{\lambda} = [\vec{w}, \vec{\mu}, \vec{\sigma}]$
- an input vector x
- an output vector y
- a number of Gaussians n

The function should return a root-mean-squared error of the estimation.

In [68]:
```
import numpy as np
def gaussian_loss(lamda, x, y, m):
    yhat = np.zeros(len(y))
    for i in range(m):
        w = lamda[i] #making a lambda that contains all the parameters w,mu,sigma
```

```
        mu = lamda[m+i]
        sigma = lamda[2*m+i]
        yhat = yhat + w*np.exp(-(x-mu)**2/(2*(sigma**2)))
    RMSE = np.sqrt(np.sum(((y-yhat)**2)/len(y)))
    return RMSE
lamda = np.array([10., 10., 10., 1080., 1300., 1500., 40., 40., 40.])
y = 10*np.exp(-(x-1080)**2/(2*(40**2)))
y += 10*np.exp(-(x-1300)**2/(2*(40**2)))
y += 10*np.exp(-(x-1500)**2/(2*(40**2)))
gloss = gaussian_loss(lamda,x,y,3)
```

Use `autograd` to compute the derivative of the loss function.

Find the derivative of the loss function when all of the parameters are 1.

In [71]:
```
! pip install autograd
import autograd.numpy as np
from autograd import grad
#need to make a lamda later so this function can be used


def lossfunc(lamda, x = x_peak, y = y_peak, m = 3):
    return gaussian_loss(lamda,x,y,m)
lamda = np.array([10., 10., 10., 1000., 1250., 1500., 30., 30., 30.])

diffg = grad(lossfunc)
print(lossfunc(lamda))
print(diffg(lamda))
```

```
Requirement already satisfied: autograd in c:\users\inara\anaconda3\lib\site-pac
kages (1.3)4.688091040865237

Requirement already satisfied: future>=0.15.2 in c:\users\inara\anaconda3\lib\si
te-packages (from autograd) (0.18.2)
Requirement already satisfied: numpy>=1.12 in c:\users\inara\anaconda3\lib\site-
packages (from autograd) (1.18.5)
[ 1.57005748e-01  1.58308313e-01  1.60464500e-01 -1.40645451e-03
  8.40456069e-05  2.05938443e-04  2.45406270e-02  2.63444151e-02
  2.65071027e-02]
```

Out[71]:
```
<function autograd.wrap_util.unary_to_nary.<locals>.nary_operator.<locals>.nary_
f(*args, **kwargs)>
```

Implement gradient descent method.

Write a function for an iteration of gradient descent that returns the optimal parameters.

The inputs are:

- a parameter vector $\vec{\lambda}$
- a function g
- a step size
- a tolerance

In [ ]:
```
lamda0 = np.array([10., 10., 10., 1000., 1250., 1500., 30., 30., 30.])
h = 0.2 #step size
tol = 0.01 #tolerance
N = 500 #iterations
def grad_descent(lamda, lossfunc, h, tol):
    for i in range(N):
        new_lamda = lamda - h*np.array(diffg(lamda))
```

```
    return new_lamda
lamdaFinal = grad_descent(lamda0,lossfunc,h,tol)
print('Initial Loss: {:4f}'.format(lossfunc(lamda0)))
print('Final Loss: {:4f}'.format(loss(lamdaFinal)))
print(lamdaFinal)
```

Find the optimal parameters.

Plot the result of non-linear regression along with the original data. Set the number of Gaussians as 5.

In [ ]:

```
n = 5
lamda2 = np.array([10., 10., 10., 1000., 1250., 1500., 30., 30., 30.])
lamFin = grad_descent(lamda2,lossfunc,h,tol)
diffg = grad(g)
```

Print the weights $\vec{w}$.

In [ ]:

Constrain the weights.

Modify the loss function to constrain the weights to be positive. You can write this in code, or you can write an analytical version of the loss function.

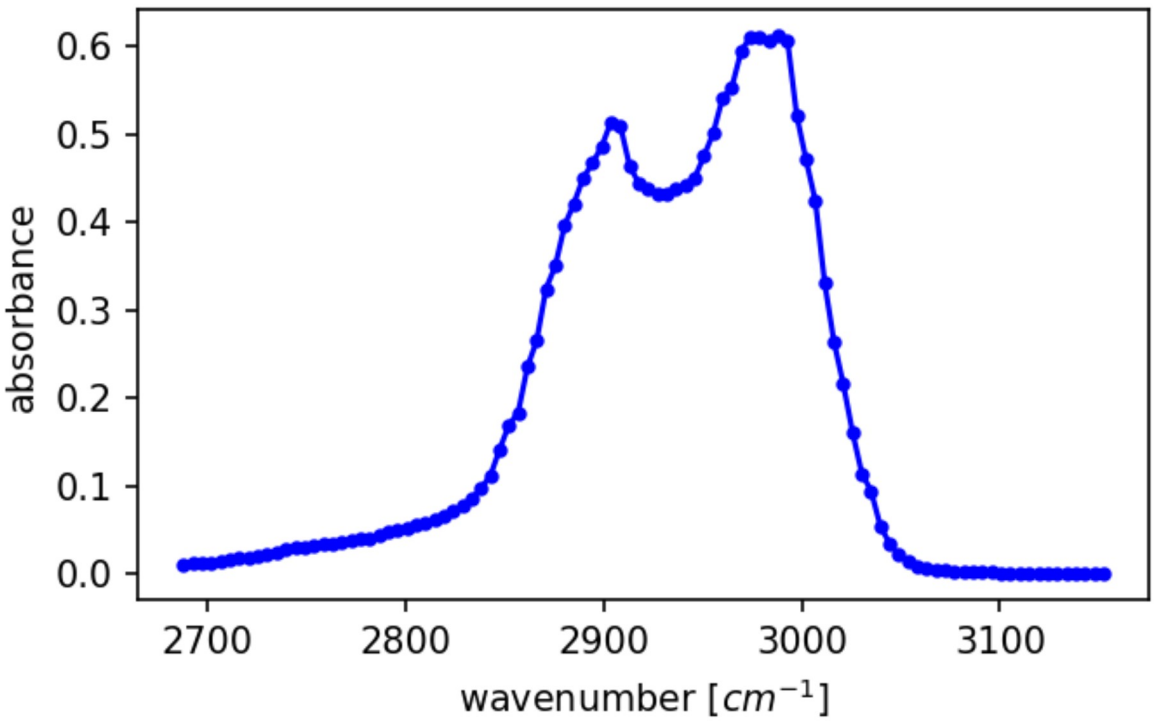
In [ ]:

Regression - Assignment 1

Data and Package Import

In [77]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import pylab as plt
```

In [78]:
```
df = pd.read_csv('data/ethanol_IR.csv')
x_all = df['wavenumber [cm^-1]'].values
y_all = df['absorbance'].values

x_peak = x_all[475:575]
y_peak = y_all[475:575]

fig, ax = plt.subplots(figsize = (5, 3), dpi = 150)
ax.plot(x_peak, y_peak, '-b', marker = '.')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
ax.set_ylabel('absorbance');
```



1. Linear Interpolation

Select every third datapoint from `x_peak` and `y_peak` dataset.

In [79]:
```
x_train = x_peak[0::3]
y_train = y_peak[0::3]
```

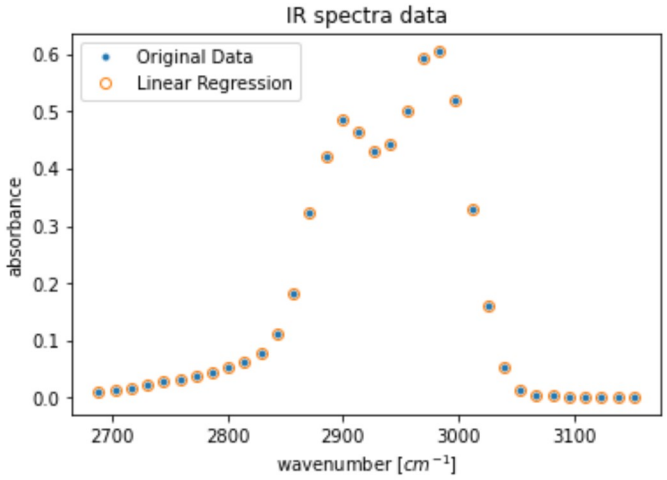
Use these datapoints to train a linear interpolation model.

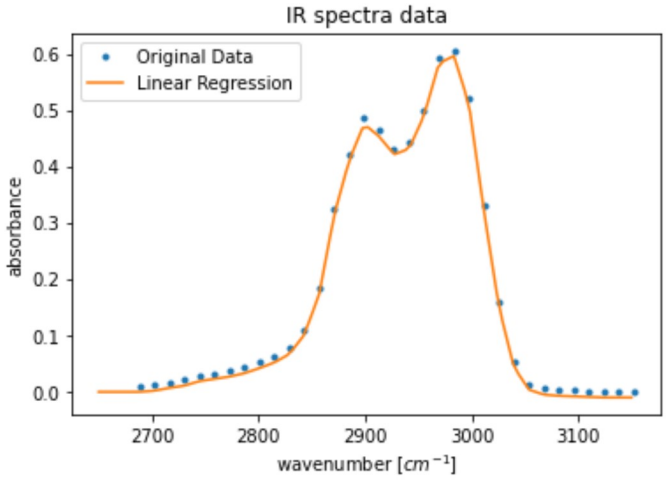
Predict the full dataset using the model and plot the result along with the original dataset.

In [80]:
```
#Define a piecewise funciton, this one has one parameter
def piecewise_lin(x):
    N = len(x)
    Xmat = np.zeros((N,N)) #make a square matrix of 0s
    for i in range(N): #for ith row in N rows
        for j in range(N): #for jth row in N cols
            Xmat[i,j] = max(0, x[i] - x[j])#piecewise
    return Xmat



Xmat = piecewise_lin(x_train)
#Need to make the last col in Xmat 1s, since the last one is still 0
#because that's how the piecewise function works
Xmat[:,-1] += 1

#the piecewise basis has features each shown as lines
#with slope 1 originating at each data point
#Do Linear Interp by solving the Gen Linear Reg Problem, using skikit-learn

from sklearn.linear_model import LinearRegression

model = LinearRegression(fit_intercept = False) #we do not need an intercept
model.fit(Xmat, y_train) #fit the model to the data

r_squared = model.score(Xmat, y_train) #r squared value shows how well model fits,
see how close to 1
yhat = model.predict(Xmat) #predict the model


#Now Plot this
fig, ax = plt.subplots()
ax.plot(x_train, y_train, '.')
ax.plot(x_train, yhat, 'o', markerfacecolor='none')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
ax.set_ylabel('absorbance')
ax.set_title('IR spectra data')
ax.legend(['Original Data', 'Linear Regression'])
print('r^2 = {}'.format(r_squared))

#Now I use the testing data, showing that I can do
#a piecewise function
def piecewise_lin(x_train, x_test=None):
    if x_test is None:
        x_test = x_train
    N = len(x_test) #<- number of data points
    M = len(x_train) #<- number of features
    Xmat = np.zeros((N,M))
    for i in range(N):
        for j in range(M):
            Xmat[i,j] = max(0, x_test[i] - x_train[j])
    return Xmat


x_predict = np.linspace(2650,3150,100) #range
X_predict = piecewise_lin(x_train,x_predict)
yhat_predict = model.predict(X_predict)
r_squared2 = model.score(Xmat,y_train)


# model = LinearRegression(fit_intercept = False) #we do not need an intercept
# model.fit(Xmat_third_test, y_train)
```

```
#Plot
fig, ax = plt.subplots()
ax.plot(x_train, y_train, '.')
ax.plot(x_predict, yhat_predict, '-', markerfacecolor ='none')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
ax.set_ylabel('absorbance')
ax.set_title('IR spectra data')
ax.legend(['Original Data', 'Linear Regression'])
print('r^2 = {}'.format(r_squared))
```

```
r^2 = 1.0
r^2 = 1.0
```





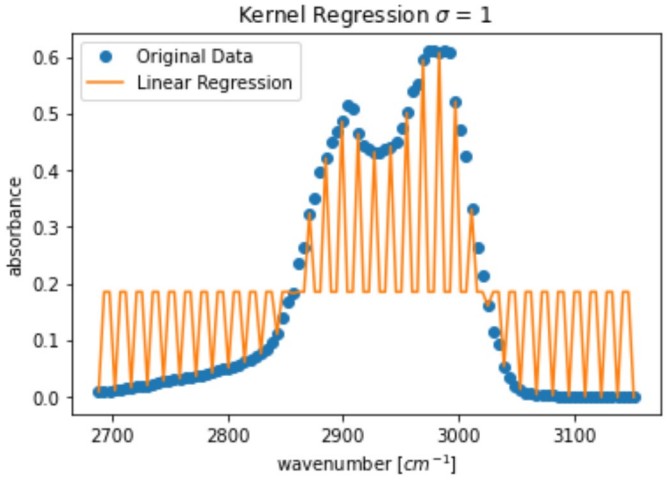
Evaluate the performance of `rbf` kernel as a function of kernel width.

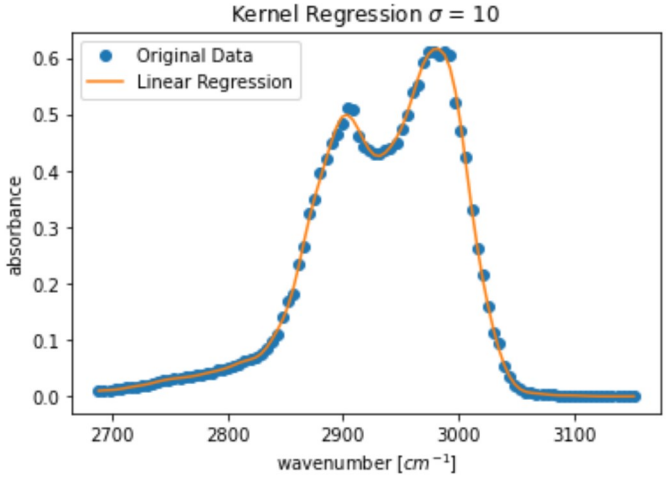
Use the same strategy as the previous exercise. Vary the width of the radial basis function with σ = [1, 10, 50, 100, 150].

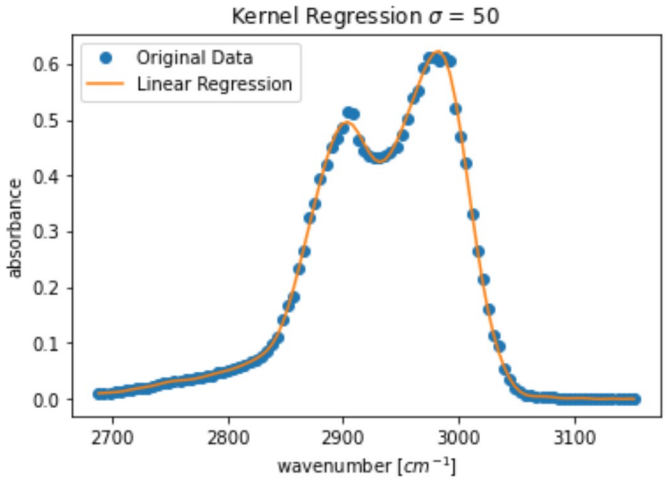
Compute the r^2 score for each using the entire dataset.

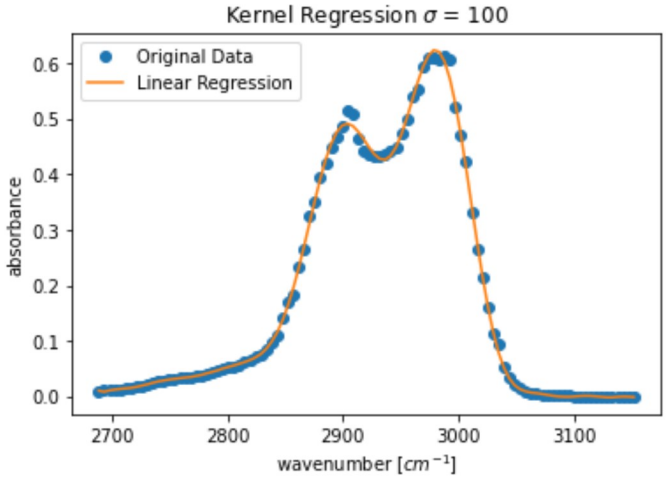
In [81]:
```
sigmas = [1, 10, 50, 100, 150]

def rbf(x_train, x_test=None, gamma=1):
    if x_test is None:
        x_test = x_train
    N = len(x_test) #<- number of data points
    M = len(x_train) #<- number of features
    X = np.zeros((N,M))
    for i in range(N):
        for j in range(M):
            X[i,j] = np.exp(-gamma*(x_test[i] - x_train[j])**2)
    return X

#1. Need to find the gamma values from each sigma
#2. Fit the model
#3. Plot

# sigma = 1
# gamma = 1./(2*sigma**2)
# X_train = rbf(x_train, x_test = x_peak, gamma = gamma)
# model_rbf = LinearRegression()
# model_rbf.fit(X_train, y_peak)
# r2 = model_rbf.score(X_train, y_peak)
# print(r2)

for sigma in sigmas:
    gamma_now = 1./(2*sigma**2)
    X_train_now = rbf(x_train, x_test = x_peak, gamma = gamma_now)
    model_rbf = LinearRegression()
    model_rbf.fit(X_train_now,y_peak)
    r_squared_now = model_rbf.score(X_train_now, y_peak)
    print('r^2 = {}'.format(r_squared_now))
    X_test = rbf(x_train,x_test = x_peak, gamma = gamma_now)
    yhat_rbf = model_rbf.predict(X_test)
    fig, ax = plt.subplots()
    ax.plot(x_peak, y_peak, 'o')
    ax.plot(x_peak, yhat_rbf, '-', markerfacecolor = 'none')
    ax.set_xlabel('wavenumber [$cm^{-1}$]')
    ax.set_ylabel('absorbance')
    ax.set_title('Kernel Regression $\sigma$ = {}'.format(str(sigma)));
    ax.legend(['Original Data', 'Linear Regression']);

#The sigma values of 10, 50, and 100 from this view seem to fit the model quite wel
l.
```

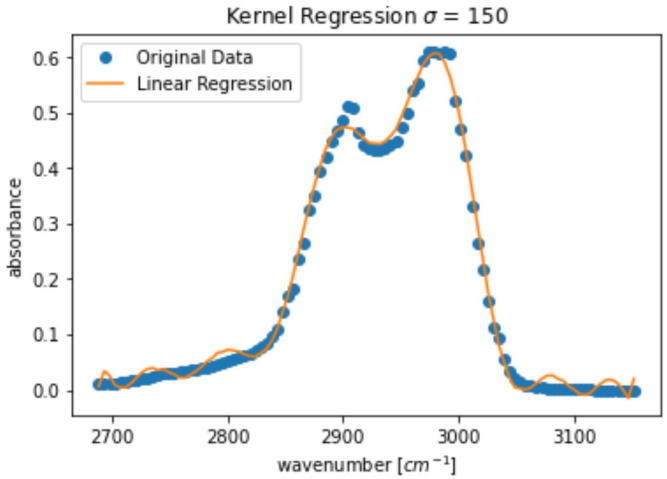
```
r^2 = 0.33185456828843174
r^2 = 0.9992031161973826
r^2 = 0.999093971097795
r^2 = 0.9988046777436903
r^2 = 0.9948178322311125
```











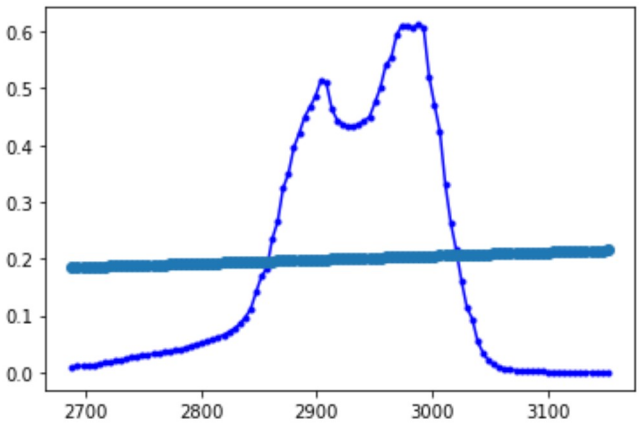
Create a model where r^2 < 0.

You can use any model from the lectures, or make one up.

The model you use does not have to optimized using the data.

In [82]:
```
#An arbitrary bad model that does not fit the data well, but
#that I will be using to show how you can get a negative r2 value is m(the slope)*x
*0.5 + 0.01
fig,ax = plt.subplots()
ax.plot(x_peak,y_peak, '-b', marker ='.')
ybar = np.mean(y_peak)
m,b = np.polyfit(x_peak,y_peak, deg = 1) #calculates the slope and any intercepts
yhat_bad = m*x_peak*0.5 +0.01#using my bad model that will have a negative r2 value
SST = sum((y_peak - ybar)**2)
SSE = sum((y_peak - yhat_bad)**2)
ax.plot(x_peak, yhat_bad, 'o')

r2_bad = (SST - SSE)/SST
print('r^2 = {}'.format(r2_bad))
```

```
r^2 = -0.0012907553202386088
```



What does negative r^2 mean?

The model is so bad, that the original data is better than the model. The error seen from the model fit is worse than an arbitrary guess for a fit for the original data.

2. Cauchy Kernel Matrix

Write a function that computes the Cauchy kernel between any two vectors x_i and x_j.

Consider the Cauchy distribution defined by:

$$C(x, x_0, \gamma) = \frac{1}{\pi\gamma}\left(\frac{\gamma^2}{(x-x_0)^2+\gamma^2}\right)$$

- x_0 is the center of the distribution. Comparable to the mean (μ) of a Gaussian distribution.
- γ is a scale factor. Comparable to the standard deviation (σ) of a Gaussian distribution.

In [83]:
```
def cauchy_kernel(x, x0, gamma):
    N = len(x)
    Cmat = np.zeros((N,N)) #initialize a Cauchy square mat
    for i in range(N):
        for j in range(N):
            Cmat[i,j] = 1/(np.pi*gamma)*(gamma**2/((x0[i]-x[j])**2 + gamma**2))
    return Cmat #cauchy_matrix
```

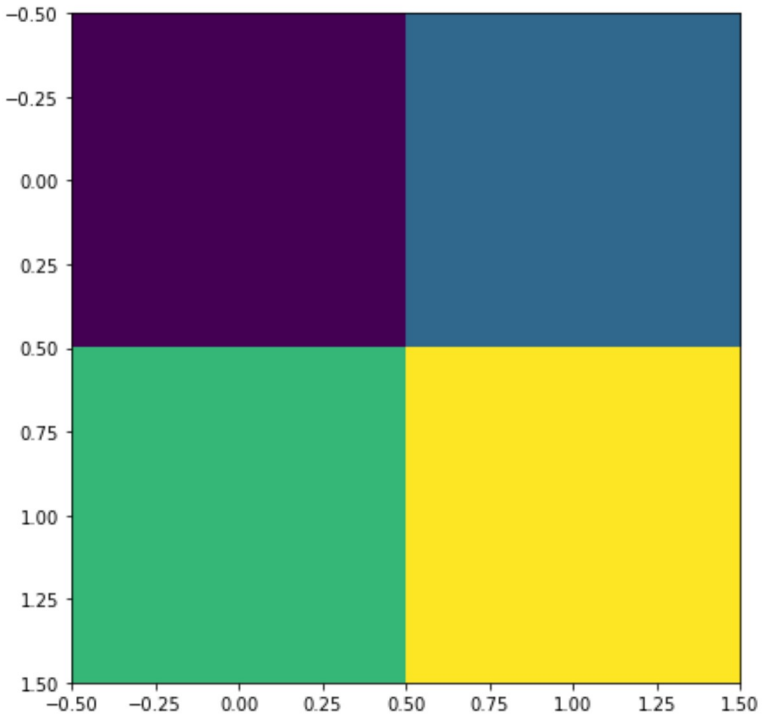
Visualize kernel matrices for the ethanol spectra dataset.

Vary the γ with [1, 10, 100].

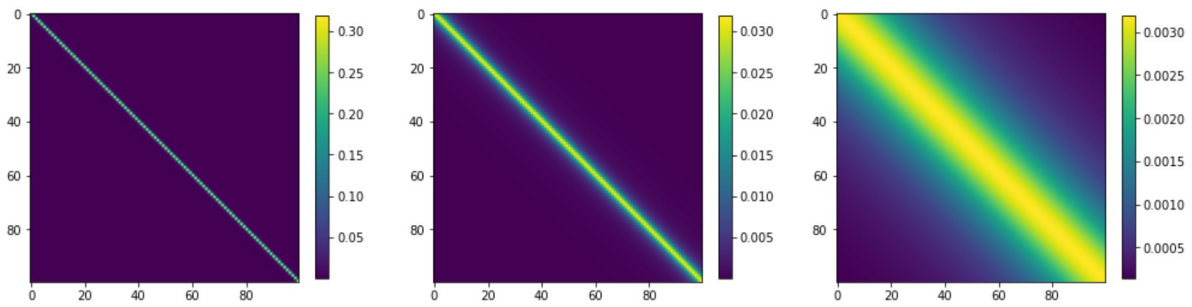
You may want to use the `plt.imshow` function to visualize the matrices. Here is an example of using `plt.imshow` .

For more details, see the documentation: https://matplotlib.org/3.2.2/api/_as_gen/matplotlib.pyplot.imshow.html (https://matplotlib.org/3.2.2/api/_as_gen/matplotlib.pyplot.imshow.html).

In [84]:
```
fig, ax = plt.subplots(figsize = (7, 7))

array = [[0, 1], [2, 3]]
ax.imshow(array, cmap = 'viridis');
```



In [65]:
```
# My group and I discussed this problem, and one of them went to office hours where
the TA let
# him know that these images are look like what I got is what is expected!
from matplotlib.pyplot import colorbar
from mpl_toolkits.axes_grid1 import make_axes_locatable

fix, ax = plt.subplots(1,3,figsize = (17,10))
x_train = x_peak

gammavals = [1, 10, 100]

for i in range(len(gammavals)):
    X_third_test = piecewise_lin(x_train, x_peak)
    X_third_test = cauchy_kernel(x_train, x_peak, gammavals[i])
    array = X_third_test

    ax_subplot = ax[i]
    im = ax[i].imshow(array, cmap = 'viridis')
    fig.colorbar(im, ax = ax_subplot, shrink = 0.4)
plt.show()


X_all = cauchy_kernel(x_peak, x_peak, gammavals[i])

fig.colorbar(im, ax = ax, shrink = 1)
plt.show()
```

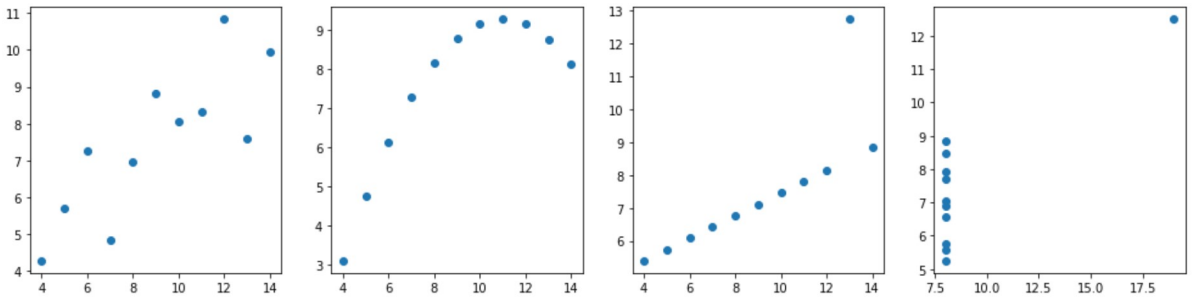


Briefly discuss the structure of these matrices.

In [85]:
```
#The width of the line shown in green gets bigger when the gamma value gets bigger.
```

3. Anscomb's Quartet

In [86]:
```
x_aq = np.array([10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5])
y1_aq = np.array([8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.6
8])
y2_aq = np.array([9.14, 8.14, 8.74, 8.77, 9.26, 8.10, 6.13, 3.10, 9.13, 7.26, 4.7
4])
y3_aq = np.array([7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42, 5.7
3])
x4_aq = np.array([8, 8, 8, 8, 8, 8, 8, 19, 8, 8, 8])
y4_aq = np.array([6.58, 5.76, 7.71, 8.84, 8.47, 7.04, 5.25, 12.50, 5.56, 7.91, 6.8
9])

fig, axes = plt.subplots(1, 4, figsize = (17, 4))

axes[0].scatter(x_aq, y1_aq)
axes[1].scatter(x_aq, y2_aq)
axes[2].scatter(x_aq, y3_aq)
axes[3].scatter(x4_aq, y4_aq);
```



Compute the means and standard deviations of each dataset.

In [87]:
```
#the values of y for x_aq for each y set should have the same mean (mu)
mu1 = np.mean(y1_aq)
mu2 = np.mean(y2_aq)
mu3 = np.mean(y3_aq)
#the values of y for x_aq for each y set should have the same mean (mu)
mu4 = np.mean(y4_aq)
#the standard deviations for the x value sets should be the same as each other
sigma1 = np.std(x_aq)
sigma2 = np.std(x4_aq)

#using a calc_stats function from the notes is a faster way to ensure
#the statistics are the same for the data sets
def calc_stats(x,y):
    y_bar = np.mean(y)
    y_std = np.std(x)
    m, b = np.polyfit(x,y,deg=1)
    SST = sum((y - y_bar)**2)
    SSE = sum((y - (m*x+b))**2)
    R2 = (SST - SSE)/SST
    return y_bar, y_std, m, b, R2

stats1 = calc_stats(x_aq,y1_aq)
print("Dataset 1: mean={:.2f}, stdev={:.2f}, m={:.2f}, b={:.2f}, R2={:.2f}".format
(*stats1))
stats2 = calc_stats(x_aq,y2_aq)
print("Dataset 2: mean={:.2f}, stdev={:.2f}, m={:.2f}, b={:.2f}, R2={:.2f}".format
(*stats2))
stats3 = calc_stats(x_aq,y3_aq)
print("Dataset 3: mean={:.2f}, stdev={:.2f}, m={:.2f}, b={:.2f}, R2={:.2f}".format
(*stats3))
stats4 = calc_stats(x4_aq,y4_aq)
print("Dataset 4: mean={:.2f}, stdev={:.2f}, m={:.2f}, b={:.2f}, R2={:.2f}".format
(*stats4))
avg, std, m, b, r2 = stats1
```

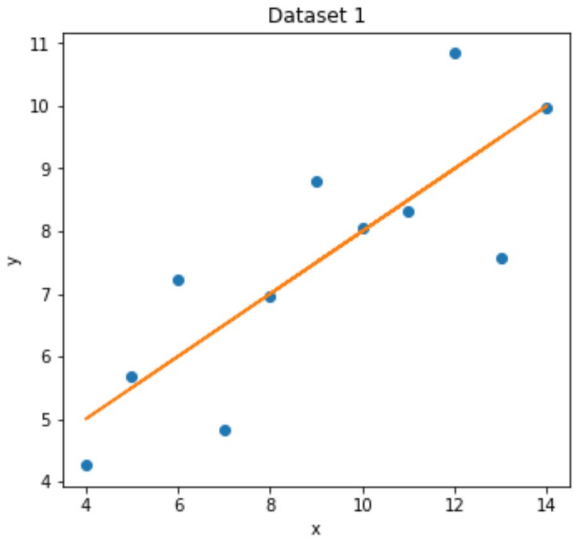
```
Dataset 1: mean=7.50, stdev=3.16, m=0.50, b=3.00, R2=0.67
Dataset 2: mean=7.50, stdev=3.16, m=0.50, b=3.00, R2=0.67
Dataset 3: mean=7.50, stdev=3.16, m=0.50, b=3.00, R2=0.67
Dataset 4: mean=7.50, stdev=3.16, m=0.50, b=3.00, R2=0.67
```

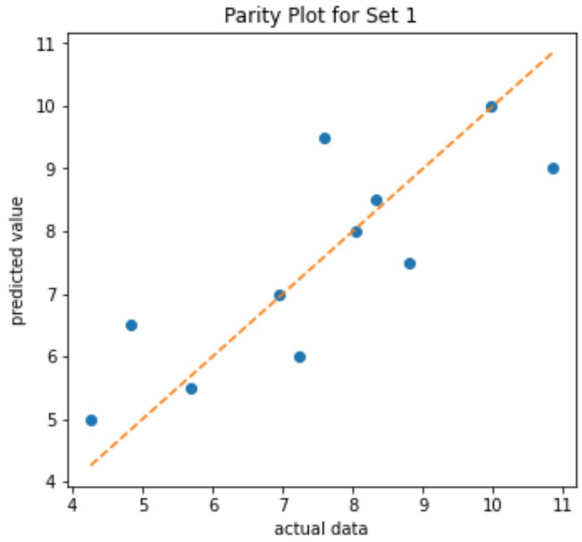
Use a linear regression to find a model $\hat{y} = mx + b$ for each dataset.

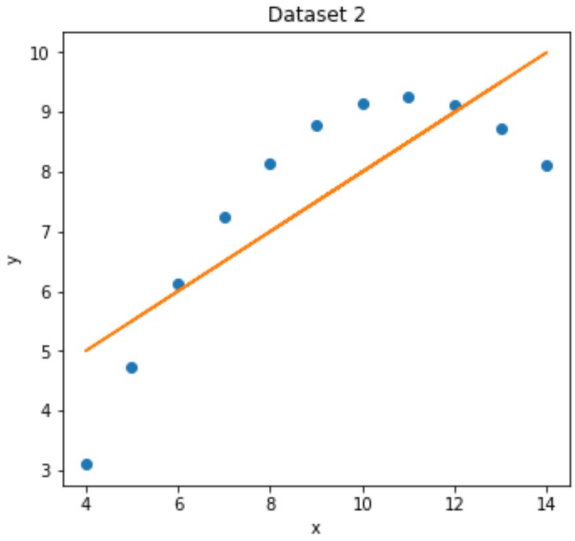
Create a parity plot between the model and the actual y values.

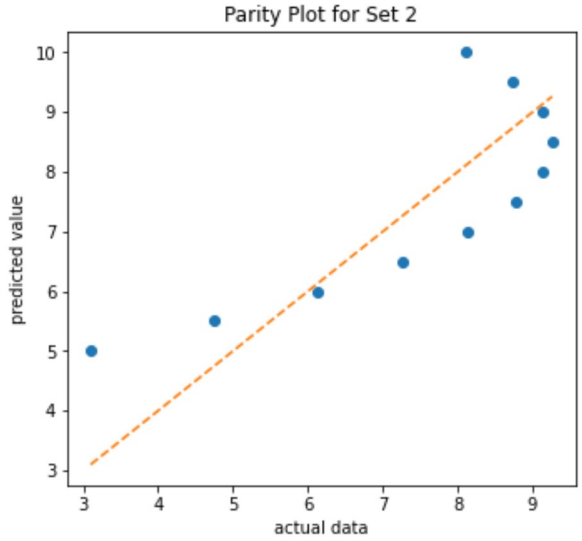
In [88]:
```
#Model 1 of 3 using x_aq
fig, axes = plt.subplots(1,2, figsize = (12, 5))
yhat = m*x_aq + b
axes[0].plot(x_aq, y1_aq, 'o')
axes[0].plot(x_aq, yhat, ls='-')
axes[0].set_xlabel('x')
axes[0].set_ylabel('y')
axes[0].set_title('Dataset 1')

axes[1].plot(y1_aq, yhat, 'o')
axes[1].plot([min(y1_aq), max(y1_aq)], [min(y1_aq), max(y1_aq)], ls='--')
axes[1].set_xlabel('actual data')
axes[1].set_ylabel('predicted value')
axes[1].set_title('Parity Plot for Set 1')
plt.show()

#Model 2 of 3
fig, axes = plt.subplots(1,2, figsize = (12, 5))
axes[0].plot(x_aq, y2_aq, 'o')
axes[0].plot(x_aq, yhat, ls='-')
axes[0].set_xlabel('x')
axes[0].set_ylabel('y')
axes[0].set_title('Dataset 2')

axes[1].plot(y2_aq, yhat, 'o')
axes[1].plot([min(y2_aq), max(y2_aq)], [min(y2_aq), max(y2_aq)], ls='--')
axes[1].set_xlabel('actual data')
axes[1].set_ylabel('predicted value')
axes[1].set_title('Parity Plot for Set 2')
plt.show()

#Model 3 of 3
fig, axes = plt.subplots(1,2, figsize = (12, 5))
axes[0].plot(x_aq, y3_aq, 'o')
axes[0].plot(x_aq, yhat, ls='-')
axes[0].set_xlabel('x')
axes[0].set_ylabel('y')
axes[0].set_title('Dataset 3')

axes[1].plot(y3_aq, yhat, 'o')
axes[1].plot([min(y3_aq), max(y3_aq)], [min(y3_aq), max(y3_aq)], ls='--')
axes[1].set_xlabel('actual data')
axes[1].set_ylabel('predicted value')
axes[1].set_title('Parity Plot for Set 3')
plt.show()

# ax.plot(y3_aq, yhat, 'o')
# ax.plot([min(y3_aq), max(y3_aq)], [min(y3_aq), max(y3_aq)], ls='--')
# ax.set_xlabel('actual data')
# ax.set_ylabel('predicted value')
# ax.set_title('Parity Plot')
# plt.show()

#Model 4 using x1_aq
yhat1 = m*x4_aq + b
fig, axes = plt.subplots(1,2, figsize = (12, 5))
axes[0].plot(x4_aq, y4_aq, 'o')
axes[0].plot(x4_aq, yhat1, ls='-')
axes[0].set_xlabel('x')
axes[0].set_ylabel('y')
axes[0].set_title('Dataset 4')

axes[1].plot(y4_aq, yhat1, 'o')
```

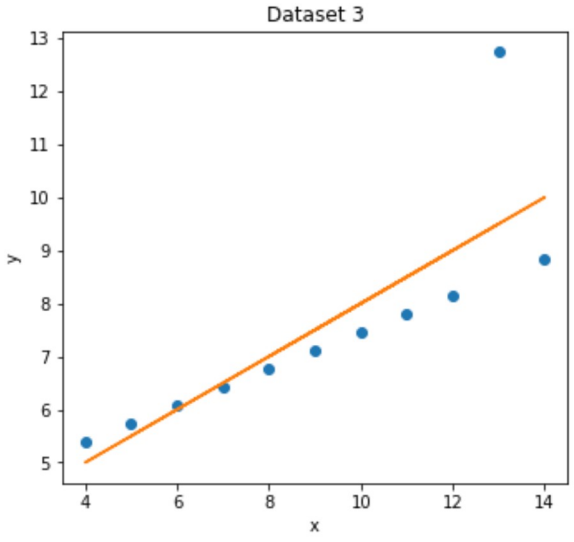
```
axes[1].plot([min(y4_aq), max(y4_aq)], [min(y4_aq), max(y4_aq)], ls='--')
axes[1].set_xlabel('actual data')
axes[1].set_ylabel('predicted value')
axes[1].set_title('Parity Plot for Second Set of X and its Y Vals')
plt.show()
```

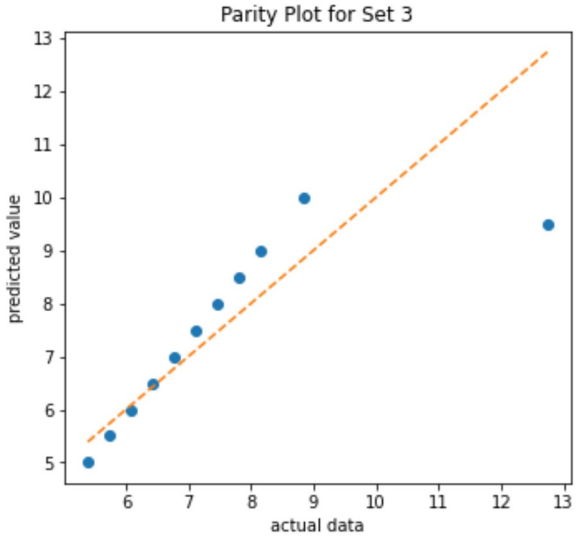

Dataset 1
y
x

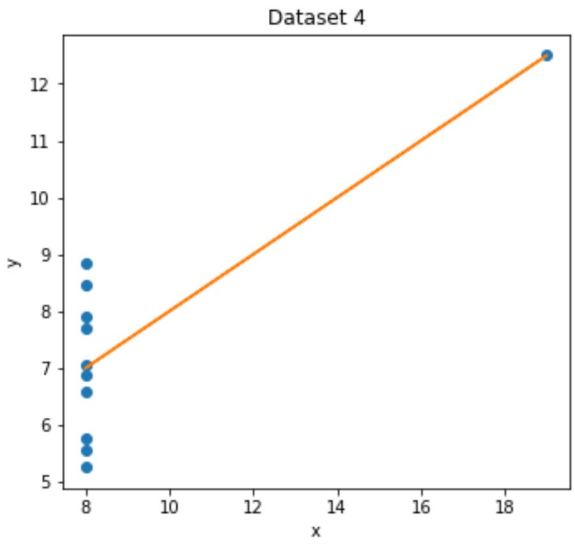

Parity Plot for Set 1
predicted value
actual data

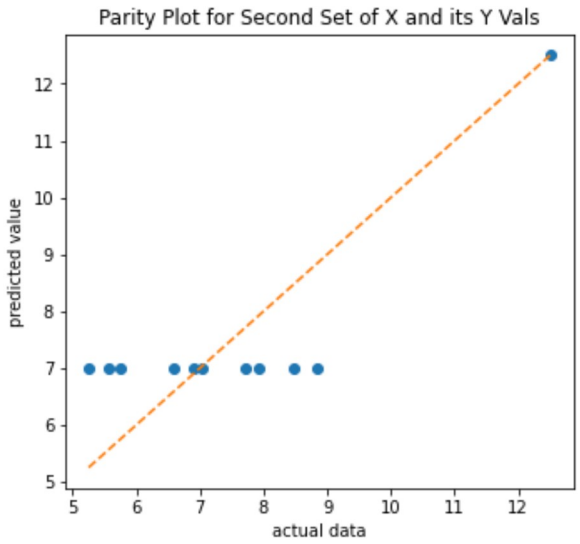

Dataset 2
y
x


Parity Plot for Set 2
predicted value
actual data


Dataset 3
y
x


Parity Plot for Set 3
predicted value
actual data





4. Assumptions for Linear Regression

List the assumptions of linear regression and the corresponding error estimation based on the standard deviation of the error.

What are the assumptions of Linear Regression using the standard deviation of the error? It's only valid if:

The error is normally distributed, meaning it follows a Gaussian distribution with a mean of 0

The error is homoscedastic, meaning the standard deviation of the Gaussian distribution doesn't depend on the independent variable

The relationship between the variables is linear, this can be checked with a scatter plot of your original data.

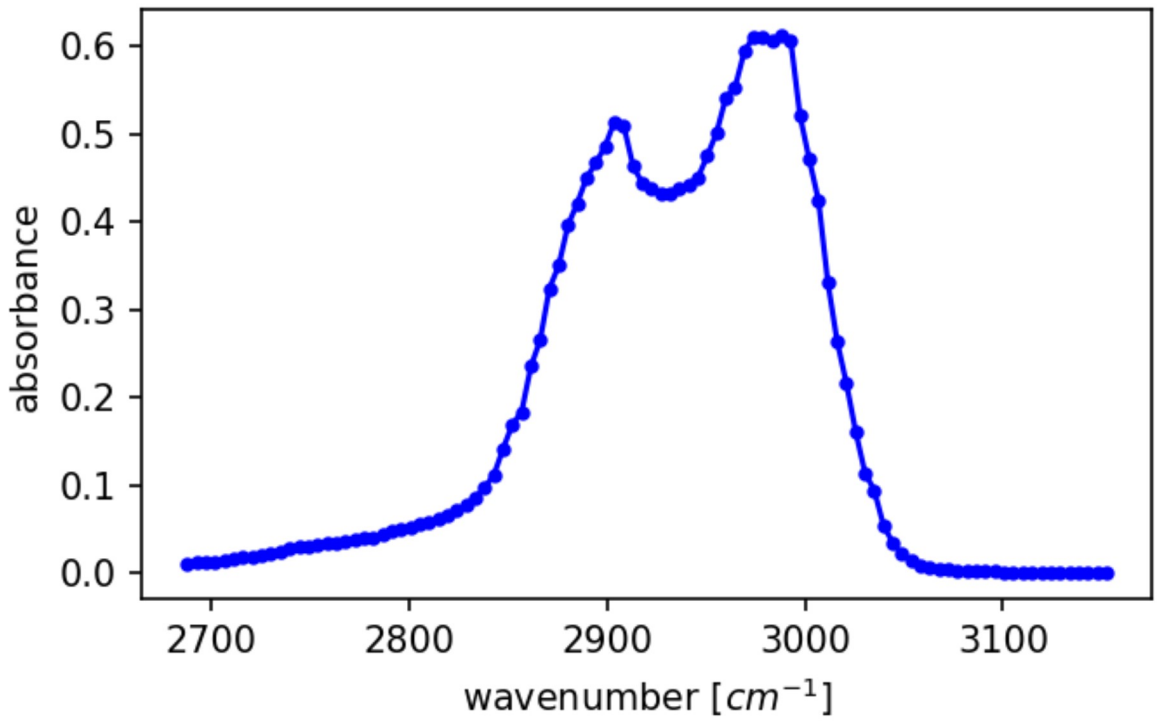
Regression - Assignment 2

Data and Package Import

In [27]:

```
%matplotlib inline
import numpy as np
import pandas as pd
import pylab as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
#I am also importing the rbf function and stats made in previous hw
#by pasting it here

def rbf(x_train, x_test=None, gamma=1):
    if x_test is None:
        x_test = x_train
    N = len(x_test) #<- number of data points
    M = len(x_train) #<- number of features
    X = np.zeros((N,M))
    for i in range(N):
        for j in range(M):
            X[i,j] = np.exp(-gamma*(x_test[i] - x_train[j])**2)
    return X
```

In [23]:
```
df = pd.read_csv('data/ethanol_IR.csv')
x_all = df['wavenumber [cm^-1]'].values
y_all = df['absorbance'].values

x_peak = x_all[475:575]
y_peak = y_all[475:575]

fig, ax = plt.subplots(figsize = (5, 3), dpi = 150)
ax.plot(x_peak, y_peak, '-b', marker = '.')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
```



1. Mean Absolute Errors

In [50]:
```
def MAE(actual, prediction):
    N = len(actual)
    sumoferrors = 0
    for i in list(range(N)):
        sumoferrors = sumoferrors + abs(actual[i] - prediction[i])

    mae = sumoferrors/N
    #print(mae)
    return mae
```

Use 8-fold cross-validation to compute the average and standard deviation of the MAE on the spectra dataset.

Use a `LinearRegression` model and an `rbf` kernel with σ=100.

Make sure to pass `shuffle = True` argument when you make a `KFold` object.

In [84]:
```
# Writing some notes on strategy:
# kfold split will split it differently on each iteration
# I will define 8 iterations
# I will do a linear regression model with rbf kernel with sig = 100 on each split
```

```
# Then we will find the MAE for each iterated split
# then will find the stats of each split using a defined calc_stats func

kf = KFold(n_splits = 8, shuffle = True)
sigma = 100
gamma = 1./2/sigma**2
r2_test = []
MAEs = []
#start doing ksplit iterations

for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=gamma)

    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t


    X_test = rbf(x_train, x_test=x_test, gamma=gamma)

    yhat_rbf = model_rbf.predict(X_test)
    currentMAE = MAE(X_test[:,0], y_test)


    MAEs.append(currentMAE)


MAEs_mean = np.mean(MAEs)
MAEs_std = np.std(MAEs)
print(MAEs_mean,MAEs_std)
```

```
0.3729140772325844 0.09647579601606301
```

Determine the optimum σ that results in the lowest mean of MAE.

Vary the width of an `rbf` kernel with σ = [1, 10, 50, 100, 150].

In [132]:

```
sigmas = [1, 10, 50, 100, 150]

kf = KFold(n_splits = 8, shuffle = True)

#gamma = 1./2/sigmas**2
r2_test = []
#I did not know how to make a for loop for both the sigmas and the k iterations SO
MAEs1 = []

total_mae_mean = []

MAEs1 = []
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=1./2/1**2)

    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t
    X_test = rbf(x_train, x_test=x_test, gamma=gamma)
    yhat_rbf = model_rbf.predict(X_test)
```

```
    currentMAE = MAE(X_test[:,0], y_test)
    MAEs1.append(currentMAE)
MAEs1_mean = np.mean(MAEs1)
total_mae_mean.append(MAEs1_mean)

MAEs10 = []
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=1./2/10**2)

    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t
    X_test = rbf(x_train, x_test=x_test, gamma=gamma)
    yhat_rbf = model_rbf.predict(X_test)
    currentMAE = MAE(X_test[:,0], y_test)
    MAEs10.append(currentMAE)
MAEs10_mean = np.mean(MAEs10)
total_mae_mean.append(MAEs10_mean)

MAEs50 = []
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=1./2/50**2)

    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t
    X_test = rbf(x_train, x_test=x_test, gamma=gamma)
    yhat_rbf = model_rbf.predict(X_test)
    currentMAE = MAE(X_test[:,0], y_test)
    MAEs50.append(currentMAE)
MAEs50_mean = np.mean(MAEs50)
total_mae_mean.append(MAEs50_mean)

MAEs100 = []
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=1./2/100**2)

    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t
    X_test = rbf(x_train, x_test=x_test, gamma=gamma)
    yhat_rbf = model_rbf.predict(X_test)
    currentMAE = MAE(X_test[:,0], y_test)
    MAEs100.append(currentMAE)
MAEs100_mean = np.mean(MAEs100)
total_mae_mean.append(MAEs100_mean)

MAEs150 = []
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=1./2/150**2)

    model_rbf = LinearRegression() #create a linear regression model instance
```

```
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t
    X_test = rbf(x_train, x_test=x_test, gamma=gamma)
    yhat_rbf = model_rbf.predict(X_test)
    currentMAE = MAE(X_test[:,0], y_test)
    MAEs150.append(currentMAE)
MAEs150_mean = np.mean(MAEs150)
total_mae_mean.append(MAEs150_mean)
low_MAE = total_mae_mean.index(min(total_mae_mean))
print(sigmas[low_MAE])
print("is the optimal sigma value")
print(total_mae_mean)

#     for sigma in sigmas:
#         X_train = rbf(x_train, gamma=1./2/sigma**2)
#         model_rbf = LinearRegression() #create a linear regression model instanc
#         model_rbf.fit(X_train, y_train) #fit the model
#         r2 = model_rbf.score(X_train, y_trin) #get the "score", which is equival


#         X_test = rbf(x_train, x_test=x_test, gamma=gamma)

#         yhat_rbf = model_rbf.predict(X_test)
#         currentMAE = MAE(X_test[:,0], y_test)
```

```
10
is the optimal sigma value
[0.2018660417114409, 0.19752068112562177, 0.19852264831007693, 0.198981975422566
1, 0.20181583263875585]
```

2. Hyperparameter Tuning

Reshape `x_peak` and `y_peak` into 2D array.

In [133]:
```
x_peakT = x_peak.reshape(-1,2)
y_peakT = y_peak.reshape(-1,2)
```

In [134]:
```
x_train, x_test, y_train, y_test = train_test_split(x_peakT, y_peakT, test_size=0.
np.random.seed(0)
xtraino = np.reshape(x_train,(-1,1))
```

Use `for` loop to determine the optimum regularization strength α of a KRR model.

Use an `rbf` kernel with σ=20.

Determine the optimum value of α out of [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1].

In [135]:
```
from sklearn.kernel_ridge import KernelRidge
alphas = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1]
sigma = 20
gamma = 1/(2*sigma**2)
r2s = []


for i in range(len(alphas)):
    KRR = KernelRidge(alpha = alphas[i], kernel = 'rbf', gamma = gamma)
```

```
    KRR.fit(xtraino, ytraino)

    x_predict = np.linspace(min(xtraino), max(xtraino), 300)
    yhat_KRR = KRR.predict(x_predict)

    r2_test = KRR.score(xtraino, ytraino)
    r2s.append(r2_test)


bestalpha = r2s.index(max(r2s))
print(alphas[bestalpha])
print("is the best alpha value")
#print(r2s)
#print(alphas[bestalpha])
#KRR = KernelRidge(kernel='rbf')
# parameter_ranges = {'alpha':alphas}
# KRR = KernelRidge(kernel='rbf')
```

```
1e-05
is the best alpha value
```

3. GridSearchCV

Import a LASSO model.

In [136]:

Shuffle the `x_peak` and `y_peak`.

You can get a shuffled array when you run `x_shuffle, y_shuffle = shuffle(x, y)`.

The reason why we shuffle the data is that `GridSearchCV` does not have an option to shuffle the input data. Note that we automatically shuffled the data in the problem 1.

In [137]:
```
from sklearn.utils import shuffle
```

Build a `GridSearchCV` model that optimizes the hyperparameters of a LASSO model for the spectra data.

Search over $\alpha \in$ [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1] and $\sigma \in$ [5, 10, 15, 20, 25, 30, 35, 40].

Use 3-fold cross-validation.

Hint: You will need to use a `for` loop over σ values. Unlike KRR, LASSO models do not take neither `gamma` nor `sigma` as a parameter. You have to make an `rbf` kernel manually and input it to a LASSO model.

Obtain the optimum α and the best score for each σ value. Use `GridSearchCV.best_score_` as accuracy metric.

In [167]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics.pairwise import rbf_kernel

# Sorry that this part isn't done, I was very confused by the PIAZZA post
# I had a lot of exams this week

alphas = np.array([1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1])
sigmas = np.array([5, 10, 15, 20, 25, 30, 35, 40])
gammas = 1./(2*sigmas**2)
```

```
vals = []
for i in range(len(sigmas)):
    gamma = 1/(2*sigmas[i]**2)
    X_train = rbf(x_train, gamma=gamma)
    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train)
    model_rbf_search = GridSearchCV(model_rbf, alphas, cv=3)
    KRR_search.fit(X_train,y_train)
    KRR_search.best_estimator_, KRR_search.best_score_
    vals.append( KRR_search.best_estimator_, KRR_search.best_score_)
```

What is the optimum σ and α?

In [ ]:
```
kf = KFold(n_splits = 8, shuffle = True)
sigma = 100
gamma = 1./2/sigma**2
r2_test = []
MAEs = []
#start doing ksplit iterations

for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=gamma)

    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t


    X_test = rbf(x_train, x_test=x_test, gamma=gamma)

    yhat_rbf = model_rbf.predict(X_test)
    currentMAE = MAE(X_test[:,0], y_test)


    MAEs.append(currentMAE)


MAEs_mean = np.mean(MAEs)

MAEs_std = np.std(MAEs)
print(MAEs_mean,MAEs_std)
```

Optional Task

Check what happens if the input data is not shuffled before the `GridSearchCV` .

In [ ]:
```

```

4. Ensemble Kernel Ridge Regression

In this problem you will combine ideas from k-fold cross-validation and bootstrapping with KRR to create an **ensemble** of KRR models.

Reshape `x_peak` and `y_peak` into 2D array.

In [140]:
```
# x_peakT = x_peak.reshape(-1,2)
# y_peakT = y_peak.reshape(-1,2)
x_peak = x_peak.reshape(-1,1) #we need to convert these to columns
```

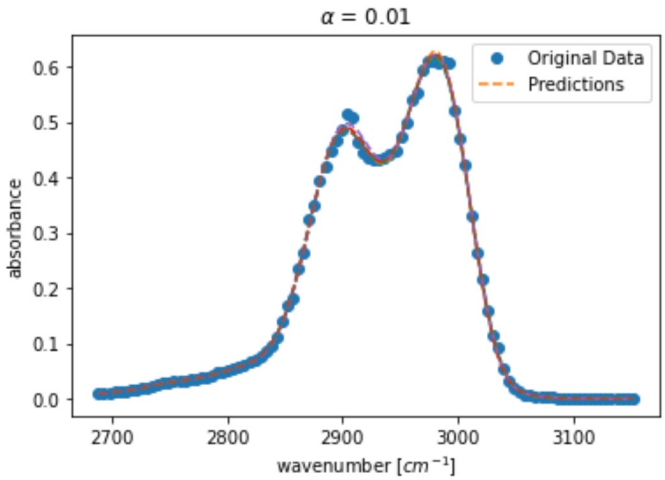
Use 5-fold cross-validation with the spectra data to construct a series of 5 KRR models with a `rbf` kernel with γ=0.0005 and α=0.01.

Each model will be trained with 80% of the data, but the exact training points will vary each time so the models will also vary.

Get the predictions from the whole `x_peak`.

In [159]:
```
gamma = 0.0005
alpha = 0.01
fig, ax = plt.subplots()
ax.plot(x_peak,y_peak,'o')
kf = KFold(n_splits = 5,shuffle = True)
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]
    #x_train, x_test, y_train, y_test = train_test_split(x_peak, y_peak, test_size
# np.random.seed(0)
# I think the instructions say to get predictions from the entire x_peak
    KRR = KernelRidge(alpha=alpha, kernel='rbf', gamma=gamma)
    KRR.fit(x_train, y_train)
#x_predict = np.linspace(min(x_peak), max(x_peak), 300) #do not use this to predic
    yhat_KRR = KRR.predict(x_peak) #what he means by predict from whole x peak
    ax.plot(x_peak,yhat_KRR, '--', markerfacecolor = 'none')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
ax.set_ylabel('absorbance')
ax.legend(['Original Data', 'Predictions'])
ax.set_title(r'$\alpha$ = {}'.format(alpha))
print('As you can see all the models are quite close to each other')
```

```
As you can see all the models are quite close to each other
so it is hard to see all of the different lines for each k fold
```



Plot the resulting ensemble of models along with the original data.

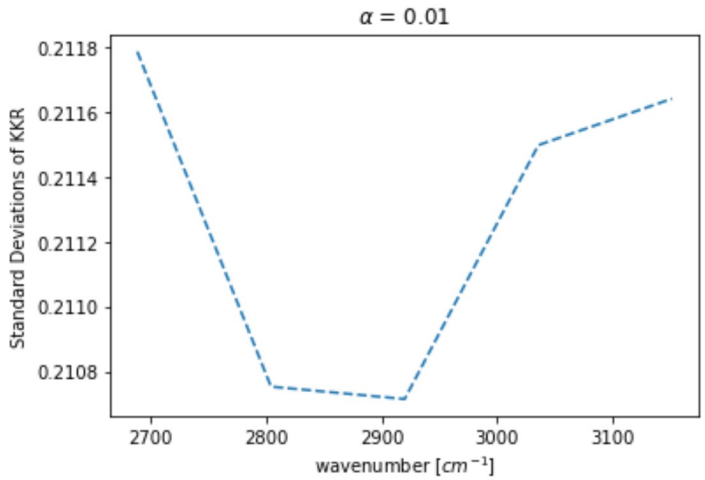
The plot should consists of 6 different lines (1 from the original data and 5 from KRR models).

In [152]:
```
#I did not know how to put these codes in separate blocks since
#I plotted the original data in the for loop for the block above!
#If you look closely you can see that there are different colors for the predictio
```

Plot the standard deviation of the 5 KRR models as a function of wavelength.

In [165]:
```
fig, ax = plt.subplots()
#fig, axes = plt.subplots(1,5, figsize = (15, 6))
dev = []
kf = KFold(n_splits = 5, shuffle = True)
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]
    KRR = KernelRidge(alpha=alpha, kernel='rbf', gamma=gamma)
    KRR.fit(x_train, y_train)
    yhat_KRR = KRR.predict(x_peak) #what he means by predict from whole x peak
    std = np.std(yhat_KRR)
    dev.append(std)
    #ax.plot(x_peak,std, '--', markerfacecolor = 'none')
wavenum = np.linspace(x_peak[0],x_peak[-1],5)
wavenum = wavenum.reshape(-1,1)

ax.plot(wavenum,dev, '--', markerfacecolor = 'none')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
ax.set_ylabel('Standard Deviations of KKR')
#ax.legend(['Original Data', 'Predictions'])
```

Out[165]: Text(0.5, 1.0, '$\\alpha$ = 0.01')



I do not think so, since the standard deviations of the predictions from the error ar
not all the same, and that it varies with the wavenumber. My graph my be wrong, but I

Table of Contents

Regression - Assignment 3

Data and Package Import

In [2]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import pylab as plt
```

In [3]:
```
df = pd.read_excel('data/impurity_dataset-training.xlsx')

def is_real_and_finite(x):
    if not np.isreal(x):
        return False
    elif not np.isfinite(x):
        return False
    else:
        return True

all_data = df[df.columns[1:]].values
numeric_map = df[df.columns[1:]].applymap(is_real_and_finite)
real_rows = numeric_map.all(axis = 1).copy().values
X = np.array(all_data[real_rows, :-5], dtype = 'float')
y = np.array(all_data[real_rows, -3], dtype = 'float')
y = y.reshape(-1, 1)

print('X matrix dimensions: {}'.format(X.shape))
print('y matrix dimensions: {}'.format(y.shape))
```

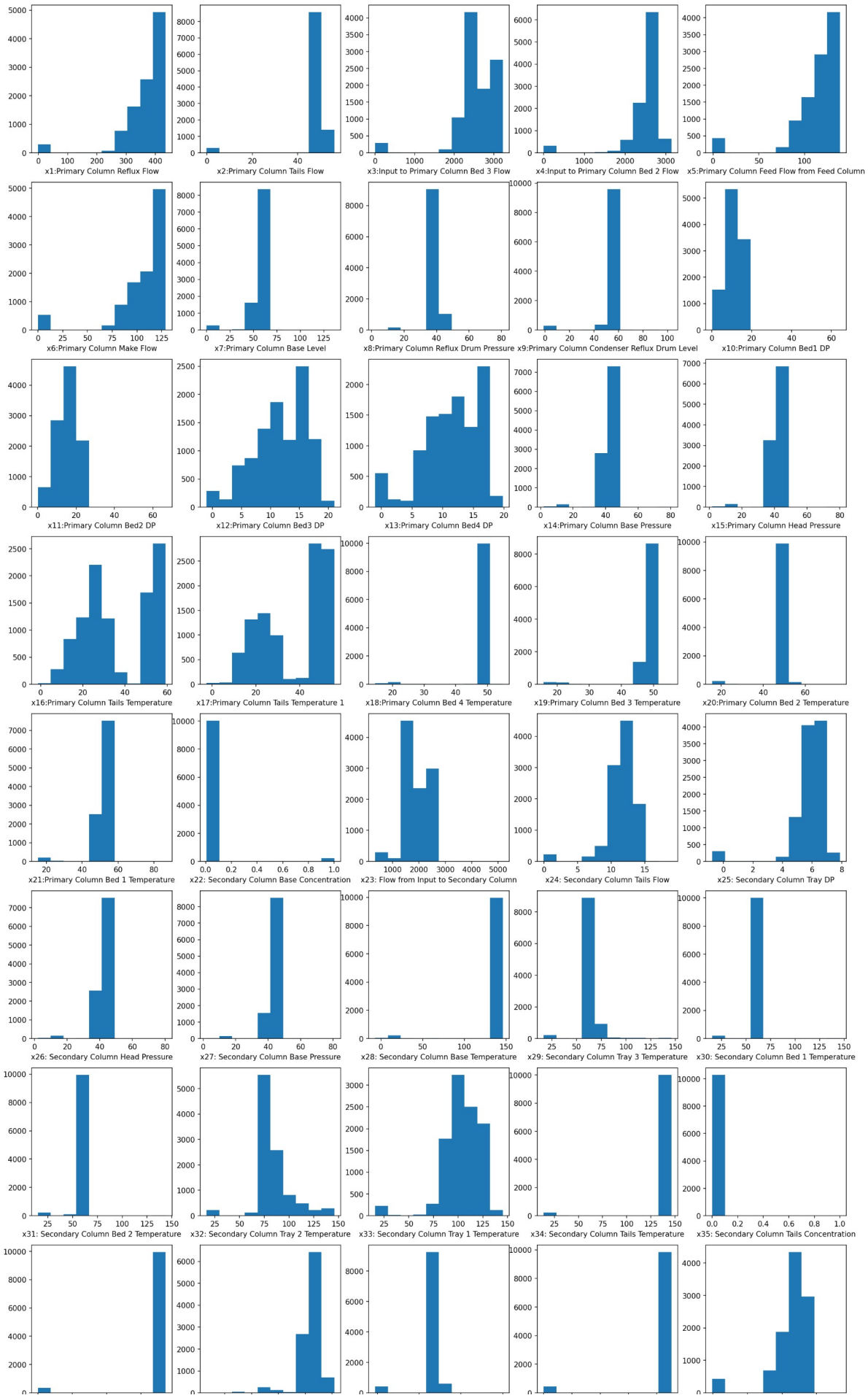
```
X matrix dimensions: (10297, 40)
y matrix dimensions: (10297, 1)
```

Distribution of Features

Plot histograms of all 40 features.

In [9]:
```
fig, axes = plt.subplots(8, 5, figsize = (20, 35), dpi = 150)
x_names = [str(x) for x in df.columns[1:41]];
y_name = str(df.columns[-3])

print('X dimensions: {}'.format(X.shape))
print('Feature names: {}'.format(x_names))
N = X.shape[-1]
n = int(np.sqrt(N))
#fig, axes = plt.subplots(n, n+1, figsize = (5*n, 5*n))
ax_list = axes.ravel()
for i in range(N):
    ax_list[i].hist(X[:,i])
    ax_list[i].set_xlabel(x_names[i])
```

```
X dimensions: (10297, 40)
Feature names: ['x1:Primary Column Reflux Flow', 'x2:Primary Column Tails Flow',
'x3:Input to Primary Column Bed 3 Flow', 'x4:Input to Primary Column Bed 2 Flow
', 'x5:Primary Column Feed Flow from Feed Column', 'x6:Primary Column Make Flow
', 'x7:Primary Column Base Level', 'x8:Primary Column Reflux Drum Pressure', 'x
9:Primary Column Condenser Reflux Drum Level', 'x10:Primary Column Bed1 DP', 'x1
1:Primary Column Bed2 DP', 'x12:Primary Column Bed3 DP', 'x13:Primary Column Bed
4 DP', 'x14:Primary Column Base Pressure', 'x15:Primary Column Head Pressure', '
x16:Primary Column Tails Temperature', 'x17:Primary Column Tails Temperature 1',
'x18:Primary Column Bed 4 Temperature', 'x19:Primary Column Bed 3 Temperature',
'x20:Primary Column Bed 2 Temperature', 'x21:Primary Column Bed 1 Temperature',
'x22: Secondary Column Base Concentration', 'x23: Flow from Input to Secondary C
olumn', 'x24: Secondary Column Tails Flow', 'x25: Secondary Column Tray DP', 'x2
6: Secondary Column Head Pressure', 'x27: Secondary Column Base Pressure', 'x28:
Secondary Column Base Temperature', 'x29: Secondary Column Tray 3 Temperature',
'x30: Secondary Column Bed 1 Temperature', 'x31: Secondary Column Bed 2 Temperat
ure', 'x32: Secondary Column Tray 2 Temperature', 'x33: Secondary Column Tray 1
Temperature', 'x34: Secondary Column Tails Temperature', 'x35: Secondary Column
Tails Concentration', 'x36: Feed Column Recycle Flow', 'x37: Feed Column Tails F
low to Primary Column', 'x38: Feed Column Calculated DP', 'x39: Feed Column Stea
m Flow', 'x40: Feed Column Tails Flow']
```


x1:Primary Column Reflux Flow
x2:Primary Column Tails Flow
x3:Input to Primary Column Bed 3 Flow
x4:Input to Primary Column Bed 2 Flow
x5:Primary Column Feed Flow from Feed Column
x6:Primary Column Make Flow
x7:Primary Column Base Level
x8:Primary Column Reflux Drum Pressure
x9:Primary Column Condenser Reflux Drum Level
x10:Primary Column Bed1 DP
x11:Primary Column Bed2 DP
x12:Primary Column Bed3 DP
x13:Primary Column Bed4 DP
x14:Primary Column Base Pressure
x15:Primary Column Head Pressure
x16:Primary Column Tails Temperature
x17:Primary Column Tails Temperature 1
x18:Primary Column Bed 4 Temperature
x19:Primary Column Bed 3 Temperature
x20:Primary Column Bed 2 Temperature
x21:Primary Column Bed 1 Temperature
x22: Secondary Column Base Concentration
x23: Flow from Input to Secondary Column
x24: Secondary Column Tails Flow
x25: Secondary Column Tray DP
x26: Secondary Column Head Pressure
x27: Secondary Column Base Pressure
x28: Secondary Column Base Temperature
x29: Secondary Column Tray 3 Temperature
x30: Secondary Column Bed 1 Temperature
x31: Secondary Column Bed 2 Temperature
x32: Secondary Column Tray 2 Temperature
x33: Secondary Column Tray 1 Temperature
x34: Secondary Column Tails Temperature
x35: Secondary Column Tails Concentration

Name a feature that is approximately normally distributed.

You may use visual inspection to answer the following questions.

x12 and x33 have a distribution that is similar to a normal distribution, resembling the bell curve that they have.

Name a feature that is approximately bimodally distributed.

In [ ]:
```
x3: Input to Primary column bed flow is bimodally distributed, as there are two hig
h points in the histogram.
```

Name a feature that has significant outliers.

x36: Feed Column Recycle flow has a large outlier since all of the data is seen in one bar on the histogram except for the very small bar at the 45 tick mark.

Feature Scaling

Down-sample the dataset by selecting every 10th data point.

In [40]:
```
X_ten = X[0::10]
y_ten = y[0::10]
```

Do a train/test split with `test_size=0.3`.

In [41]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_ten, y_ten, test_size = 0.3)
```

Use the standard scaler and make the standardized dataset.

In [42]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)

X_scaled_test = scaler.transform(X_test)
```

Build a KRR model on the Dow dataset with and without scaling.

Set γ=0.01 and α=0.01.

In [43]:
```
from sklearn.kernel_ridge import KernelRidge
alpha = 0.01
gamma = 0.01

#With scaling

KRR = KernelRidge(alpha = alpha, kernel = 'rbf', gamma = gamma)
KRR.fit(X_scaled, y_train)
yhat = KRR.predict(X_test)
r2_scaled = KRR.score(X_scaled_test,y_test)


#Without scaling
KRR = KernelRidge(alpha = alpha, kernel = 'rbf', gamma = gamma)
KRR.fit(X_train, y_train)
yhat = KRR.predict(X_test)
r2_unsc = KRR.score(X_test,y_test)
```

```
0.7966119235982759
-5.376112560367068
```

Compare the r^2 score on the test set of the two approaches.

In [45]:
```
print('r2 of unscaled training set: {}'.format(r2_unsc))
print('r2 of scaled training set: {}'.format(r2_scaled))
print('The r2 of the scaled set is much closer to 1 than the unscaled set.')
```

```
r2 of unscaled training set: -5.376112560367068
r2 of scaled training set: 0.7966119235982759
The r2 of the scaled set is much closer to 1 than the unscaled set.
```

LASSO Regression

Scale the feature matrix using the standard scaler.

In [58]:
```
scaler = StandardScaler()
X_scaled_tot = scaler.fit_transform(X)

#X_scaled_test_tot = scaler.transform(X_test)
```

Shuffle the data.

In [59]:
```
from sklearn.utils import shuffle
X_shuffle, y_shuffle = shuffle(X_scaled_tot, y)
```

Build a `GridSearchCV` model that optimizes the hyperparameters of a LASSO model.

Search over $\alpha \in$ [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1].

Use 3-fold cross-validation.

In [71]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso
import warnings
warnings.simplefilter('ignore')

r2 = []
alphas = np.array([1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1])
lasso = Lasso(max_iter = 1000000, tol = 0.005)
param_grid = {'alpha':alphas}

lasso_search = GridSearchCV(lasso,param_grid, cv = 3)
lasso_search.fit(X_shuffle,y_shuffle)
print(lasso_search.best_estimator_, lasso_search.best_score_)
r2.append(lasso_search.best_score_)
```

```
Lasso(alpha=0.0001, max_iter=1000000, tol=0.002) 0.6729721137283695
```

Evaluate the performance of the best model.

Print the optimized α as well as the r^2 score.

In [66]:
```
print(lasso_search.best_estimator_, lasso_search.best_score_)
```

```
Lasso(alpha=0.0001, max_iter=1000000, tol=0.005) 0.6727408027494165
```

Describe which features (if any) were dropped.

Dropped features have coefficients equal to zero.

In [72]:
```
coeff = lasso_search.best_estimator_.coef_
coeff.shape
for x in range(len(coeff)):
    if np.isclose(coeff[x],0):
        print(x_names[x])
```

```
x27: Secondary Column Base Pressure
```

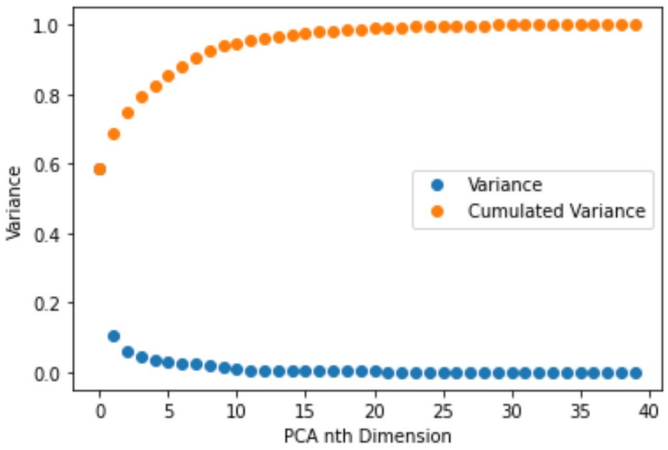
Principal Component and Forward Selection

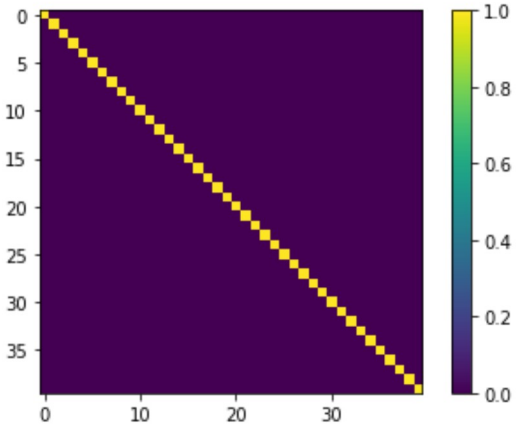
Use the eigenvalues of the covariance matrix to perform PCA on the scaled feature matrix.

Hint: You can check your answers using PCA from `scikit-learn` or other packages if you want

In [107]:
```
from scipy.linalg import eigvals, eig
corr = np.corrcoef(X.T)
covar = np.cov(X_shuffle.T)
np.isclose(corr, covar, 1e-4).all()
eigvals, eigvecs = eig(cor)

PCvals, PCvecs = eigvals, eigvecs
tot_variance = np.sum(np.real(PCvals))
variance_exp = np.real(PCvals)/tot_variance

fig, ax = plt.subplots()
ax.plot(variance_exp,'o')
ax.plot(np.cumsum(variance_exp),'o')
ax.set_xlabel('PCA nth Dimension')
ax.set_ylabel('Variance')
ax.legend(['Variance', 'Cumulated Variance'])

PC_projection = np.dot(X_scaled_tot, PCvecs)
print(PC_projection.shape)

corr_PCs = np.corrcoef(PC_projection.T)
fig,ax = plt.subplots()
c = ax.imshow(corr_PCs)
fig.colorbar(c);
```

```
(10297, 40)
```





Determine which principal component of the dataset is most linearly correlated with the impurity concentration.

Print the order of the principal component (e.g. 5th PC) and its r^2 score.

In [92]:
```
from sklearn.linear_model import LinearRegression


PC_projection = np.dot(X_scaled_tot, PCvecs)

N = 5

model_PC = LinearRegression() #create a linear regression model instance
model_PC.fit(PC_projection[:, :N], y) #fit the model
r2 = model_PC.score(PC_projection[:, :N], y) #get the "score", which is equivalent
to r^2
print("5th order r^2 PCA = {}".format(r2))

model = LinearRegression() #create a linear regression model instance
model.fit(X_scaled_tot[:, :N], y) #fit the model
r2 = model.score(X_scaled_tot[:, :N], y) #get the "score", which is equivalent to r
^2
print("r^2 regular = {}".format(r2))
```

```
r^2 PCA = 0.4454460131377648
r^2 regular = 0.4644174145725659
```

Determine which original feature of the dataset is most linearly correlated to the impurity concentration.

Print the name of the feature and its r^2 score.

In [106]:
```
score_list = []
max_score = 0
max_r2 = 0
for j in range(PC_projection.shape[1]):
    model = LinearRegression() #create a linear regression model instance
    xj = PC_projection[:,j].reshape(-1,1)
    model.fit(xj, y) #fit the model
    r2 = model.score(xj, y) #get the "score", which is equivalent to r^2
    score_list.append([r2, j])
    if r2 < 1:
        if r2 > max_r2:
            max_r2 = r2
            max_score = j
score_list.sort()
score_list.reverse()

print('The feature that is most linearaly correlated is ',x_names[max_score])
for r, j in score_list:
    print("{} : r^2 = {}".format(j, r))
print((score_list))
```

```
The feature that is most linearaly correlated is  x2:Primary Column Tails Flow
1 : r^2 = 0.20685135722275394
0 : r^2 = 0.1740523250716769
6 : r^2 = 0.061224828716807345
7 : r^2 = 0.06048989471356592
4 : r^2 = 0.044172097738576
25 : r^2 = 0.017497338519020134
8 : r^2 = 0.016205721751365476
5 : r^2 = 0.01395158068641944
2 : r^2 = 0.013223153135118126
16 : r^2 = 0.013047707758553129
33 : r^2 = 0.011755340770008949
18 : r^2 = 0.009381481309652218
9 : r^2 = 0.00914449012666807
15 : r^2 = 0.008497745937736445
3 : r^2 = 0.007147079969638592
21 : r^2 = 0.006899441884438695
31 : r^2 = 0.006459664342175042
22 : r^2 = 0.0051135909850487105
14 : r^2 = 0.003515332215413447
11 : r^2 = 0.0033082402426463098
39 : r^2 = 0.003210578115308449
38 : r^2 = 0.002510692115424873
10 : r^2 = 0.0023541786992695712
27 : r^2 = 0.0022663723214011444
32 : r^2 = 0.002216144465406522
13 : r^2 = 0.0019406439983039592
37 : r^2 = 0.0019400593063473304
20 : r^2 = 0.0018165356203506677
28 : r^2 = 0.0014754678401431853
12 : r^2 = 0.0009878389646386099
36 : r^2 = 0.0007256293006049352
34 : r^2 = 0.0006972577893956666
24 : r^2 = 0.0006704453274829492
26 : r^2 = 0.0005656925838339877
17 : r^2 = 0.00047279622293650014
35 : r^2 = 0.00044635061456155256
30 : r^2 = 0.00035149123509436997
19 : r^2 = 0.00020390636051270672
29 : r^2 = 3.351129185191759e-05
23 : r^2 = 1.63738252623169e-07
[[0.20685135722275394, 1], [0.1740523250716769, 0], [0.061224828716807345, 6],
[0.06048989471356592, 7], [0.044172097738576, 4], [0.017497338519020134, 25],
[0.016205721751365476, 8], [0.01395158068641944, 5], [0.013223153135118126, 2],
[0.013047707758553129, 16], [0.011755340770008949, 33], [0.009381481309652218, 1
8], [0.00914449012666807, 9], [0.008497745937736445, 15], [0.007147079969638592,
3], [0.006899441884438695, 21], [0.006459664342175042, 31], [0.0051135909850487
105, 22], [0.003515332215413447, 14], [0.0033082402426463098, 11], [0.00321057811
5308449, 39], [0.002510692115424873, 38], [0.0023541786992695712, 10], [0.002266
3723214011444, 27], [0.002216144465406522, 32], [0.0019406439983039592, 13], [0.
0019400593063473304, 37], [0.0018165356203506677, 20], [0.0014754678401431853, 2
8], [0.0009878389646386099, 12], [0.0007256293006049352, 36], [0.000697257789395
6666, 34], [0.0006704453274829492, 24], [0.0005656925838339877, 26], [0.00047279
622293650014, 17], [0.00044635061456155256, 35], [0.00035149123509436997, 30],
[0.00020390636051270672, 19], [3.351129185191759e-05, 29], [1.63738252623169e-0
7, 23]]
```

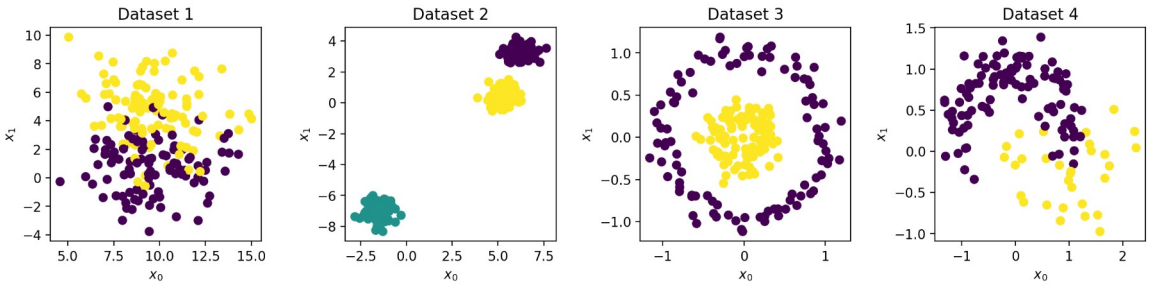
Classification - Assignment 1

Data and Package Import

In [17]:
```
%matplotlib inline
import numpy as np
import pandas as pd
```

In [18]:

```
from sklearn.datasets import make_blobs, make_moons, make_circles
np.random.seed(4)

noisiness = 1

X_blob, y_blob = make_blobs(n_samples = 200, centers = 2, cluster_std

X_mc, y_mc = make_blobs(n_samples = 200, centers = 3, cluster_std = 0.

X_circles, y_circles = make_circles(n_samples = 200, factor = 0.3, noi

X_moons, y_moons = make_moons(n_samples = 200, noise = 0.25 * noisines

N_include = 30
idxs = []
Ni = 0
for i, yi in enumerate(y_moons):
    if yi == 1 and Ni < N_include:
        idxs.append(i)
        Ni += 1
    elif yi == 0:
        idxs.append(i)

y_moons = y_moons[idxs]
X_moons = X_moons[idxs]

fig, axes = plt.subplots(1, 4, figsize = (15, 3), dpi = 200)

all_datasets = [[X_blob, y_blob], [X_mc, y_mc], [X_circles, y_circles]

labels = ['Dataset 1', 'Dataset 2', 'Dataset 3', 'Dataset 4']
for i, Xy_i in enumerate(all_datasets):
    Xi, yi = Xy_i
    axes[i].scatter(Xi[:, 0], Xi[:, 1], c = yi)
    axes[i].set_title(labels[i])
    axes[i].set_xlabel('$x_0$')
    axes[i].set_ylabel('$x_1$')

fig.subplots_adjust(wspace = 0.4);
import numpy as np
clrs = np.array(['#003057', '#EAAA00', '#4B8B9B', '#B3A369', '#377117'
```



1. Discrimination Lines

Derive the equation for the line that discriminates between the two classes.

Consider a model of the form:

$\bar{\bar{X}}\vec{w} > 0$ if $y_i = 1$ (class 1)

$\bar{\bar{X}}\vec{w} < 0$ if $y_i = -1$ (class 2)

where $\bar{\bar{X}} = [\vec{x_0}, \vec{x_1}, \vec{1}]$ and $\vec{w} = [w_0, w_1, w_2]$.

The equation should be in the form of $x_1 = f(x_0)$. Show your work, and/or explain the process

x0w0 + x1w1 + w2 = 0 x1 = (-w0/w1)*x0 - (w2/w1)

#Therefore, x1 deprends of x0, as seen above.

Derive the discrimination line for a related non-linear model

In this case, consider a model defined by:

$y_i = w_0x_0 + w_1x_1 + w_2(x_0^2 + x_1^2)$

where the model predicts class 1 if $y_i > 0$ and predicts class 2 if $y_i \leq 0$.

The equation should be in the form of $x_1 = f(x_0)$. Show your work, and/or explain the process you used to arrive at the answer.

w1x1 = yi - w0x0 - w2x0^2 + w2x1^2 w1x1 - w2x1^2 = yi - w0x0 - w2x0^2 w2x1^2-w1x1 + yi - w0x0 - w2x0^2 = 0

#quadratic formula shows that it is a function of x0 x1 = (-w1 +/- sqrt((w1^2 - 4(-w2)(yi - w0x0 - w2x0^2)))/(2w2)

Briefly describe the nature of this boundary.

What is the shape of the boundary? Is it linear or non-linear?

The decision boundary depends on how the classes may be separated. If it is able to be linearly separated with a straight line, then the boundary is linear, if the classes can still be separated with a continuous line that is not straight, then the boundary is non-linear but the classes would still be separable.

2. Assessing Loss Functions

In [19]:
```
def add_intercept(X):
    intercept = np.ones((X.shape[0], 1))
    X_intercept = np.append(intercept, X, 1)
    return X_intercept
```

In [20]:
```
def linear_classifier(X, w):
    X_intercept = add_intercept(X)
    p = np.dot(X_intercept, w)
```

Write a function that computes the loss function for the perceptron model.

The function should take the followings as arguments:

- weight vector w
- the feature matrix $\bar{\bar{X}}$
- the output vector $\vec{y}$

You may want to use functions above.

In [21]:
```
def perceptron(w, X, y): #"max cost function"
    X_intercept = add_intercept(X)
    Xb = np.dot(X_intercept,w)
    loss = sum(np.maximum(0, -y*Xb))
    return loss
```

Write a function that computes the loss function for the logistic regression model.

The function should take the followings as arguments:

- weight vector w
- the feature matrix $\bar{\bar{X}}$
- the output vector $\vec{y}$

You may want to use functions above.

In [22]:
```
#need to fix problems within max cost function
#Problems: trivial soln at w = 0, max func not differentiable
def log_reg(w, X, y): #this is the softmax function from the notes
    X_intercept = add_intercept(X)
    Xb = np.dot(X_intercept, w)
    exp_yXb = np.exp(-y * Xb)
    loss = sum(np.log(1 + exp_yXb))
    return loss
```

Minimize the both loss functions using the Dataset 3 above.

In [23]:
```
from scipy.optimize import minimize
noisiness = 1
X,y = make_circles(n_samples = 200, factor = 0.3, noise = 0.1*noisines
w = np.array([1,2,3])

percep_min = minimize(perceptron, w, args = (X,y))
w_perceptron = percep_min.x

loss_reg_min = minimize(log_reg, w, args = (X,y))
```

What is the value of the loss function for the perceptron model after optimization?

In [24]:
```
print(w_perceptron)
print('The optimized perceptron loss func is',perceptron(w_perceptron,
```

```
[1.95101589 1.79683778 2.72724758]
The optimized perceptron loss func is 0.0
```

What is the value of the loss function for the logistic regression model after optimization?

In [25]:
```
print(w_log_reg)
print('The optimized log reg is', log_reg(w_log_reg,X,y))
```

```
[16.24231948 -1.11773931 -1.6600907 ]
The optimized log reg is 69.31472797610287
```

What are the two main challenges of the perceptron loss function?

The perceptron loss function has two issues, one being that it has a trivial solution at w(vec) = 0 becomes a zero vector, and the other being that the function is not differentiable.

3. Support Vector Machine

Write a function that computes the loss function of the support vector machine model.

This functions should take the followings as arguments:

- weight vector w
- the feature matrix $\bar{\bar{X}}$
- the output vector $\vec{y}$
- regularization strength α

You may want to use `add_intercept` and `linear_classifier` functions from the Problem 2.

In [26]:
```
def svm(w, X, y, alpha):
    X_intercept = add_intercept(X)
    Xb = np.dot(X_intercept,w)
    cost = sum(np.maximum(0,1-y*Xb))
```

```
    cost += alpha*np.linalg.norm(w[1:],2)
    loss = cost
```

Evaluate the effect of regularization strength.

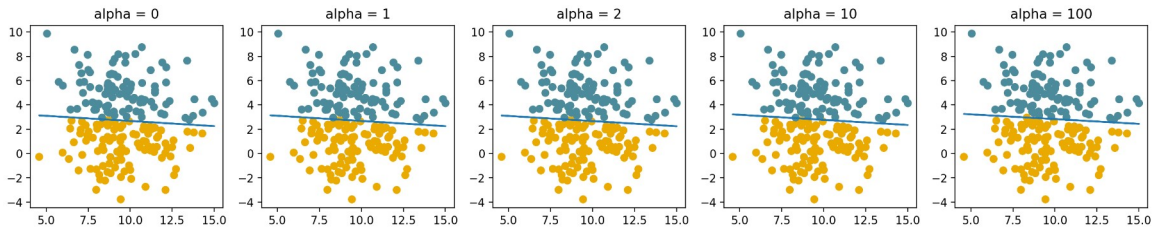
Optimize the SVM model for **Dataset 1**.

Search over α = [0, 1, 2, 10, 100] and assess the loss function of the SVM model.

In [27]:
```
alphas = np.array([0,1,2,10,100])
w_guess = np.array([1,2,3])

for i in np.array([0,1,2,3,4]):
    cur_result = minimize(svm, w_guess, args = (X, y, alphas[i]))
    w_svm = cur_result.x
    m = -w_svm[1] / w_svm[2]
    b = -w_svm[0] / w_svm[2]
```

Plot the discrimination lines for α = [0, 1, 2, 10, 100].

In [28]:
```
alphas = np.array([0,1,2,10,100])
w_guess = np.array([1,2,3])
X = X_blob
y = y_blob * 2 - 1
fig, axes = plt.subplots(1, 5, figsize = (18, 3), dpi = 200)

for i in np.array([0,1,2,3,4]):
    cur_result = minimize(svm, w_guess, args = (X, y, alphas[i]))
    w_svm = cur_result.x
    prediction = linear_classifier(X, w_svm)
    m = -w_svm[1] / w_svm[2]
    b = -w_svm[0] / w_svm[2]
    axes[i].plot(X[:, 0], m*X[:, 0] + b, ls = '-')
    axes[i].scatter(X[:, 0], X[:, 1], c = clrs[y_blob + 1])
    axes[i].scatter(X[:, 0], X[:, 1], c = clrs[prediction + 1])
    axes[i].set_title("alpha = {}".format(alphas[i]))
```



Find the optimal set of hyperparameters for an SVM model with Dataset 1.

Use `GridSearchCV` and find the optimal value of α and γ.

In [38]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics.pairwise import rbf_kernel
```

```
np.random.seed(0)

alphas = np.array([1,2,10,100])
Cs = 1./alphas
sigmas = np.array([5, 10, 15, 20])
gammas = 1./(2*sigmas**2)
params = {'C': Cs}
r2s = []
for i in np.array([0,1,2,3]):
    svc_model = SVC(kernel = 'rbf', gamma = gammas[i],C=Cs[i])
    x_train = rbf_kernel(X, X,gammas[i])
    svc_search = GridSearchCV(svc_model,params,cv=3)
    svc_search.fit(x_train,y)
    print('For {}, r2 = : {}, alpha = {}'.format(svc_search.best_estim
    r2s.append(svc_search.best_score_)

#I could not use alpha is 0 because when passing in the C vector I got
print(' ')
```

```
For SVC(gamma=0.02), r2 = : 0.854741444293683, alpha = 1
For SVC(gamma=0.005), r2 = : 0.8649178350670889, alpha = 2
For SVC(gamma=0.0022222222222222222), r2 = : 0.5859339665309814, alph
a = 10
For SVC(gamma=0.00125), r2 = : 0.5808834614804764, alpha = 100

The optimal alpha is 1, and in my case with a gamma of 0.02
```

Calculate the accruacy, precision, and recall for the best model.

You can write your own function that calculates the metrics or you may use built-in functions.

In [39]:

```
def acc_prec_recall(y_model, y_actual):
    TP = np.sum(np.logical_and(y_model == y_actual, y_model == 1))
    TN = np.sum(np.logical_and(y_model == y_actual, y_model == 0))
    FP = np.sum(np.logical_and(y_model != y_actual, y_model == 1))
    FN = np.sum(np.logical_and(y_model != y_actual, y_model == 0))
    acc = (TP + TN) / (TP + TN + FP + FN)
    if TP == 0:
        prec = 0
        recall = 0
    else:
        prec = TP / (TP + FP)
        recall = TP / (TP + FN)
    return acc, prec, recall
# I know from earlier that the optimal alpha is 1 and gamma is 0.02

svc = SVC(kernel = 'rbf',gamma = 0.02, C=1)
svc.fit(X,y)
y_model = svc.predict(X)
ans = acc_prec_recall(y_model,y)
```

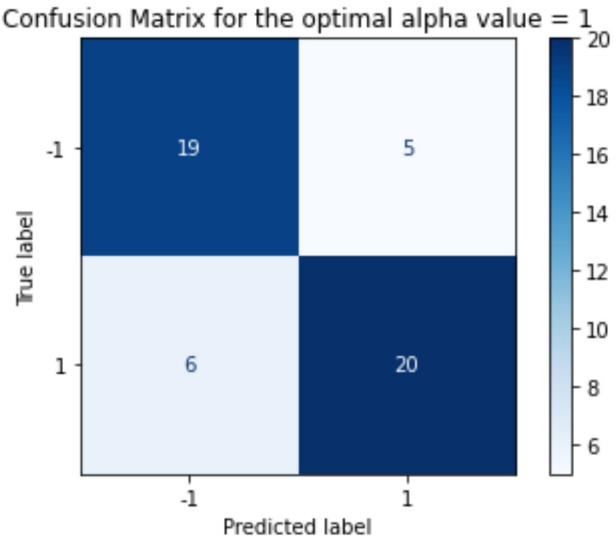
```
(0.8924731182795699, 0.8924731182795699, 1.0)
```

Plot the confusion matrix.

In [40]:

```
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix

# I looked up the documentation from sklearn

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state
svc_model = SVC(kernel='rbf', C=1)
svc_model.fit(X_train, y_train)
disp = plot_confusion_matrix(svc_model, X_test, y_test,cmap=plt.cm.Blu
disp.ax_.set_title('Confusion Matrix for the optimal alpha value = 1')
print(disp.confusion_matrix)
```

```
[[19  5]
 [ 6 20]]
```



What happens to the decision boundary as α goes to ∞?

As alpha goes to infinity C goes to 0, and as C decreases we have more "support vectors," meaning the decision boundary between the different classes is not as precise around the classes.

What happens to the decision boundary as γ goes to 0?

As gamma goes to 0 the boundary is "less complex" which I take to mean the same thing as if the alpha goes to infinity, the decision boundary will be less precise when separating the classes.

4. 6745 Only: Analytical Derivation - I am taking 4745, not 6745! I am not a grad student

Derive an analytical expression for the gradient of the softmax function with respect to $\vec{w}$.

The **softmax** loss function is defined as:

$$g(\vec{w}) = \sum_i log(1 + \exp(-y_i \vec{x}_i^T \vec{w}))$$

where $\vec{x}_i$ is the i-th row of the input matrix $\bar{\bar{X}}$.

Hint 1: The function $g(\vec{w})$ can be expressed as $f(r(s(\vec{w})))$ where r and s are arbitrary functions and the chain rule can be applied.

Type *Markdown* and LaTeX: α^2

Optional: Logistic regression from the regression perspective

An alternate interpretation of classification is that we are performing non-linear regression to fit a **step function** to our data (because the output is whether 0 or 1). Since step functions are not differentiable at the step, a smooth approximation with non-zero derivatives must be used. One such approximation is the *tanh* function:

$$\tanh(x) = \frac{2}{1+\exp(-x)} - 1$$

This leads to a reformulation of the classification problem as:

$$\vec{y} = \tanh(\bar{\bar{X}}\vec{w})$$

Show that this is mathematically equivalent to **logistic regression**, or minimization of the **softmax** cost function.

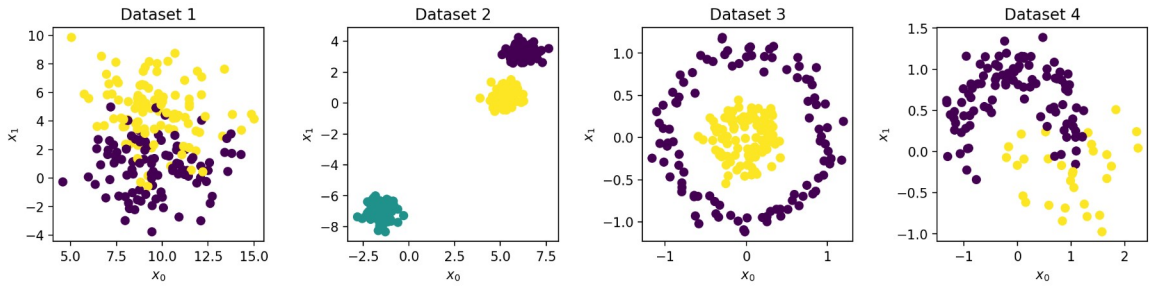
Type *Markdown* and LaTeX: α^2

Classification - imahmood7_HW07

Data and Package Import

In [3]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import pylab as plt
```

In [4]:
```
from sklearn.datasets import make_blobs, make_moons, make_circles
np.random.seed(4)

noisiness = 1

X_blob, y_blob = make_blobs(n_samples = 200, centers = 2, cluster_std =
2 * noisiness, n_features = 2)

X_mc, y_mc = make_blobs(n_samples = 200, centers = 3, cluster_std = 0.5
* noisiness, n_features = 2)

X_circles, y_circles = make_circles(n_samples = 200, factor = 0.3, nois
e = 0.1 * noisiness)

X_moons, y_moons = make_moons(n_samples = 200, noise = 0.25 * noisines
s)

N_include = 30
idxs = []
Ni = 0
for i, yi in enumerate(y_moons):
    if yi == 1 and Ni < N_include:
        idxs.append(i)
        Ni += 1
    elif yi == 0:
        idxs.append(i)

y_moons = y_moons[idxs]
X_moons = X_moons[idxs]

fig, axes = plt.subplots(1, 4, figsize = (15, 3), dpi = 200)

all_datasets = [[X_blob, y_blob], [X_mc, y_mc], [X_circles, y_circles],
[X_moons, y_moons]]

labels = ['Dataset 1', 'Dataset 2', 'Dataset 3', 'Dataset 4']
for i, Xy_i in enumerate(all_datasets):
    Xi, yi = Xy_i
    axes[i].scatter(Xi[:, 0], Xi[:, 1], c = yi)
    axes[i].set_title(labels[i])
    axes[i].set_xlabel('$x_0$')
    axes[i].set_ylabel('$x_1$')

fig.subplots_adjust(wspace = 0.4);
```



In [5]:
```
df = pd.read_csv('data/perovskite_data.csv')
X_perov = df[['nA', 'nB', 'nX', 'rA (Ang)', 'rB (Ang)', 'rX (Ang)', 't
', 'tau']].values
y_perov = df['exp_label'].values
```

1. k-nearest Neighbors Model

1-NN

Calculate the accuracy of a 1-nearest Neighbors model for the training data.

A 1-nearest Neighbors model considers a point as its own nearest neighbors.

Hint: the block below is not a code block.

The accuracy is 1, since for the training data if we say the point itself is the nearest neighbor, then it will just make the model based on exactly the training data, making the accuracy 1, which is up for suspicion.

Will this be a reliable indicator of its accuracy for testing data?

Briefly explain your answer.

No, anytime we have an accuracy of 1, we should be skeptical when applying it to the testing data, and in this case it is not which we will find out if we cross-validate it. It has overfitted for the training data.

Weighted Neighbors Classification

Instead of selecting the k-nearest neighbors to vote, we could design an algorithm where all neigbors get to vote, but their vote is weighted inversely to their distance from the point of interest:

$y_i = \sum_j y_j/(||x_i - x_j||)$

where j is an index over all training points.

The class will be assigned as follows:

- class 1 if $y_i \geq 0$
- class -1 if $y_i < 0$

In [6]:
```
#This function finds the distance between two points using the L2 norm
def distance(x1, x2):
    return np.linalg.norm(x1 - x2, 2)
```

In [7]:
```
#This function finds the neighbors for point x
def get_neighbor(x, x_list): #point x, and x_list is a list of training
points
    dist_pairs = []
    for i, xi in enumerate(x_list):
        dist = distance(x, xi)
        dist_pairs.append([dist, i]) #every run appends the distance wi
th that index
    return dist_pairs
```

Write a function that assigns a class to a point.

The function should take the followings as arguments:

- a single point `x`
- a list of training points `x_list`
- a list of training labels `y_list`

You may want to use functions above. You will also need to add a statement to avoid dividing by zero if the point is in the training set. If the distance between 2 points is zero, then the label from the same point in the training set should be used (e.g. if $x_i = x_j$ then $y_i = y_j$).

In [8]:
```
def assign_class(x, x_list, y_list):
    yi = 0
    for i, xj in enumerate(x_list):
        x_xj_dist = distance(x,xj)
        if x_xj_dist == 0:
            yi_now = y_list[i]
        yi_now = y_list[i]/x_xj_dist
        yi += yi_now

    if yi < 0:
        assignment = -1
    if yi >= 0:
        assignment = 1

    return assignment
```

Write a function that returns the prediction for a given list of testing points.

The function should take the followings as arguments:

- a list of testing points `X`
- a list of training points `X_train`
- a list of training labels `y_train`

In [9]:
```
def weighted_neighbors(X, X_train, y_train):
    y_hat = []
    for x in X:
        y_now = assign_class(x, X_train, y_train)
        y_hat.append(y_now)
    return y_hat
```

Train the model for the perovskite dataset using a random selection of 75% of the data as training data.

In [10]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_perov, y_perov, t
est_size = 0.75)
y_hat = weighted_neighbors(X_perov, X_train, y_train) #In office hours
Sihoon mentioned
# to use the entire X_perov when doing the y_hat
```

```
<ipython-input-8-fe3ac42a026f>:7: RuntimeWarning: divide by zero enco
untered in true_divide
  yi_now = y_list[i]/x_xj_dist
```

Compute the accuracy and precision of the prediction.

In [11]:
```
# def acc_prec_recall(y_model, y_actual): I had tried defining a functi
on myself, but
# in office hours it did not work well for me so I will be using the bu
ilt in function
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score

print("The accuracy is")
print(accuracy_score(y_perov,y_hat))
print("The precision is")
print(precision_score(y_perov,y_hat))
```

```
The accuracy is
0.9149305555555556
The precision is
0.8771428571428571
```

In [ ]:
```

```

Train a 5-NN model using the same training data.

In [12]:
```
# def kNN(X, k, X_train, y_train):
#     y_out = []  #this is analogous to the y_hat in the weighted neigh
bors func
#     for xi in X:
#         y_out.append(assign_class(xi, X_train, y_train, k))
#     y_out = np.array(y_out)
#     return y_out

# y_hat_kNN = kNN(X_perov,5,X_train,y_train)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_perov,y_perov)
y_hat_knn = knn.predict(X_perov)
```

Compute the accuracy and precision.

In [13]:
```
print("The accuracy is")
print(accuracy_score(y_perov,y_hat_knn))
print("The precision is")
print(precision_score(y_perov,y_hat_knn))
```

```
The accuracy is
0.9444444444444444
The precision is
0.934984520123839
```

2. Multi-dimensional Classification

Simple logistic regression

Train a logistic regression model using all columns except the `tau` column of the perovskite dataset.

You may use some functions that have been already built in the previous assignments.

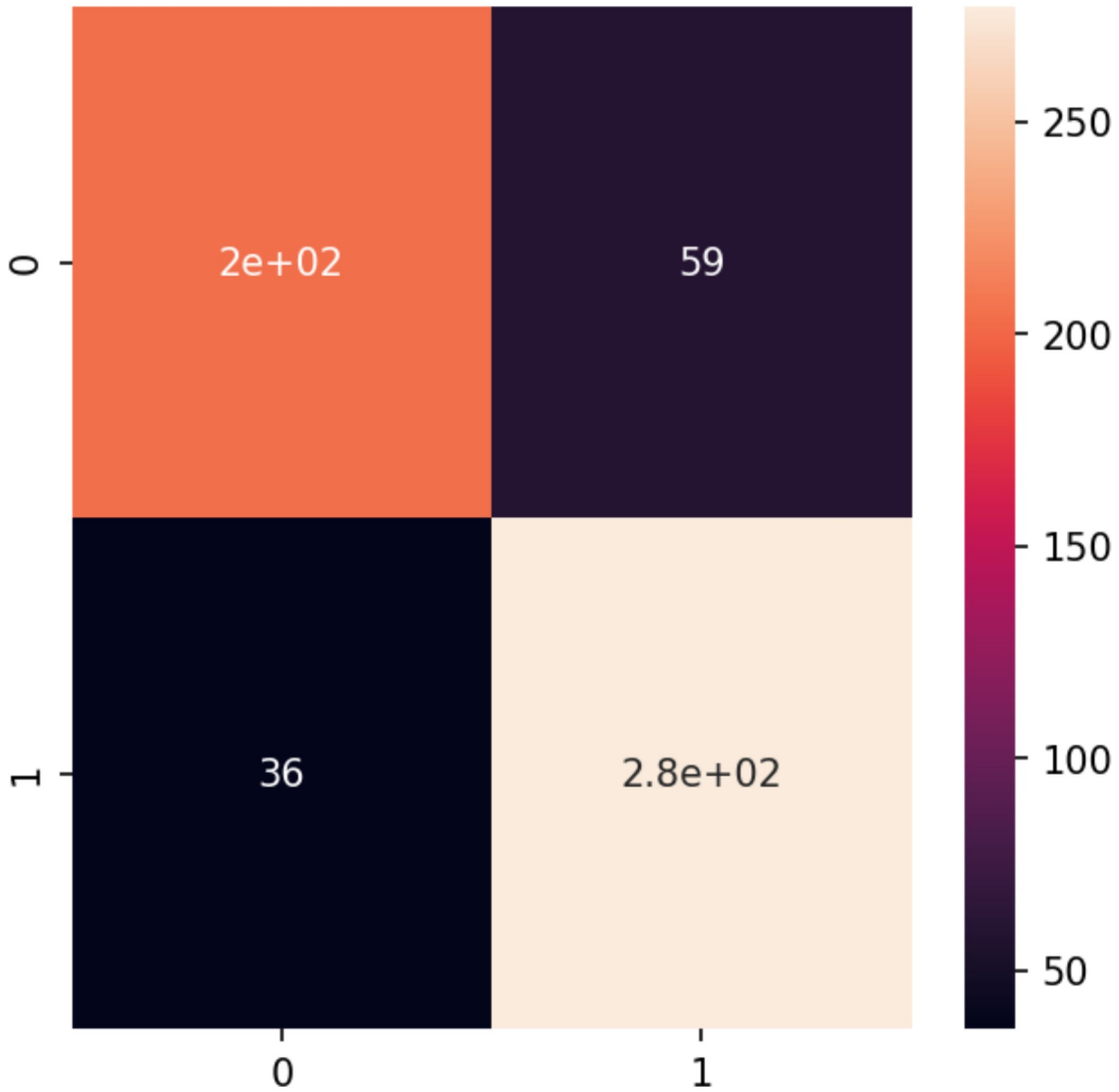
In [14]:
```
from scipy.optimize import minimize
def add_intercept(X):
    intercept = np.ones((X.shape[0],1))
    X_intercept = np.append(intercept,X,1)
    return X_intercept

def log_reg(w, X, y):
    X_intercept = add_intercept(X)
    Xb = np.dot(X_intercept, w)
    exp_yXb = np.exp(-y * Xb)
    loss = sum(np.log(1 + exp_yXb))
    return loss
w = [1, 2, 3, 4, 5, 6 ,7, 8]

X_cut = np.delete(X_perov,7,axis = 1)


result = minimize(log_reg,w, args = (X_cut,y_perov))
w_logit = result.x


def linear_classifier(X, w):
    X_intercept = add_intercept(X)
    p = np.dot(X_intercept, w)
    return p > 0

prediction = linear_classifier(X_cut,w_logit)

new_predict = np.zeros(np.size(prediction))
for i, x in enumerate(new_predict): #need to convert the prediction mat
rix from true/false to -1 and 1 for the classes
    if prediction[i]:
        new_predict[i] = 1
    else:
        new_predict[i] = -1
```

Plot the confusion matrix.

In [15]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
import seaborn as sns


cm = confusion_matrix(y_perov,new_predict)
print(cm)
fig, ax = plt.subplots(figsize = (5,5),dpi = 150)
sns.heatmap(cm,annot = True)
```

```
[[204  59]
 [ 36 277]]
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x251e19b7f70>



Compute the accuracy, precision and recall.

In [16]:
```
from sklearn.metrics import recall_score
print("The accuracy is")
print(accuracy_score(y_perov,new_predict))
print("The precision is")
print(precision_score(y_perov,new_predict))
print("The recall is")
print(recall_score(y_perov,new_predict))
```

```
The accuracy is
0.8350694444444444
The precision is
0.8244047619047619
The recall is
0.8849840255591054
```

6745 Only: Customizing non-linear boundaries

In this problem, you will create a single custom feature that improves the separation performance as much as possible.

Plot the `y_perov` as a function of `rA (Ang)` and `rB (Ang)` .

N/A. I am not taking 6745.

Build a baseline model based on logistic regression.

Report the accuracy and precision of the baseline model.

N/A

Plot the prediction of the baseline model.

N/A

Create a new feature based on a non-linear combination of `rA (Ang)` and `rB (Ang)` .

Plot the new feature as a function of `rA (Ang)` .

N/A

Build a new model that includes `rA (Ang)` , `rB (Ang)` and your new feature.

Report the accuracy and precision.

N/A

Plot the result of your new model.

N/A

Briefly explain how you decided on the feature.

N/A

3. Comparison of Classification Model

In this problem, you will compare the classification performance of three different models using the perovskite dataset.

Choose three different classification models and import them.

These could be models discussed in the lectures, or others that you have learned about elsewhere.

In [52]:
```
#choosing SVC, choose an arbitrary hyperparam for each
from sklearn.svm import SVC
svc = SVC(kernel = 'rbf',gamma = 100, C = 1000)
svc.fit(X_perov, y_perov)
y_predict_svc = svc.predict(X_perov)
print('The accuracy of SVC before optimization is:{}'.format(accuracy_s
core(y_perov,y_predict)))

#choosing Decision Tree
from sklearn.tree import DecisionTreeClassifier
tree=DecisionTreeClassifier()
tree.fit(X_perov,y_perov)
y_predict = tree.predict(X_perov)
print('The accuracy of Decision Tree before optimization is:{}'.format
(accuracy_score(y_perov,y_predict)))


#choosing kNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 20)
knn.fit(X_perov, y_perov)
y_predict = knn.predict(X_perov)
print('The accuracy of kNN before optimization is:{}'.format(accuracy_s
core(y_perov,y_predict)))
```

```
The accuracy of SVC before optimization is:0.9236111111111112
The accuracy of Decision Tree before optimization is:0.99652777777777
78
The accuracy of kNN before optimization is:0.9236111111111112
```

Make a hyperparameter grid for each model.

You should optimize at least one hyperparameter for each model.

In [32]:
```
#param for svc
gammas = 1./np.linspace(5,50,10)
Cs = 1./np.array([1,2,10,50,100])
param_grid_svc = {'gamma': gammas,'C': Cs}


#param for decision tree
depths = np.array([1,2,3,4,5,6,7,8,9,10])
param_grid_tree={'max_depth':depths}

#param for k neighbor
depths = np.array([1,2,3,4,5,6,7,8,9,10])
param_grid_knn = {'n_neighbors': depths}
```

Optimize hyperparameters.

First, you select a validation set using hold-out (`train_test_split`). Optimize hyperparameters using `GridSearchCV` on the training set.

In [49]:
```
from sklearn.model_selection import GridSearchCV
# svm hyperparameters
X_train, X_test, y_train, y_test = train_test_split(X_perov, y_perov, t
est_size = 0.75)

svcg = SVC(kernel = 'rbf')
svm_search = GridSearchCV(svcg,param_grid_svc,cv=3)
svm_search.fit(X_train,y_train)
opt_C = svm_search.best_estimator_.C
opt_gamma = svm_search.best_estimator_.gamma
print('SVC:----------')
print('Optimized C for SVC:{}'.format(opt_C))
print('Optimized Gamma for SVC:{}'.format(opt_gamma))
print('With Accuracy of:{}'.format(svm_search.best_score_))

#decision tree
dtree = DecisionTreeClassifier()
dt_search = GridSearchCV(dtree, param_grid_tree, cv=3)
dt_search.fit(X_train,y_train)
#opt_neighbor = dt_search.best_estimator_.n_neighbors
dt_search.best_estimator_, dt_search.best_score_
print("Decision Tree:------------")
print('Optimized Depth for Decistion Tree:{}'.format(dt_search.best_est
imator_.max_depth),'With Accuracy of:{}'.format(dt_search.best_score_))

#knn
knng = KNeighborsClassifier()
knn_search = GridSearchCV(knng,param_grid_knn,cv=3)
knn_search.fit(X_train,y_train)
knn_search.best_estimator_, dt_search.best_score_
print("K Nearest Neighbors:-------------")
print('Optimized Number of Neighbors for kNN:{}'.format(dt_search.best_
estimator_.max_depth),'With Accuracy of:{}'.format(dt_search.best_score
_))
```

```
SVC:----------
Optimized C for SVC:1.0
Optimized Gamma for SVC:0.2
With Accuracy of:0.8125
Decision Tree:------------
Optimized Depth for Decistion Tree:1 With Accuracy of:0.8958333333333
334
K Nearest Neighbors:-------------
Optimized Number of Neighbors for kNN:1 With Accuracy of:0.8958333333
333334
```

Compare the accuracy by predicting the results of the validation set.

This was calculated in the code block above this one, as you can see the accuracy for each cross-validated model is lower than it was for the arbitrarily made ones.

Briefly describe your conclusions based on the results.

I think the accuracy score of my models are higher before cross validating it with the train_test_split and then optimizing the hyperparameters, and with a high accuracy, like the one seen close to 99% is a cause for skepticism. Therefore this shows that it is necessary to validate a model to check if that is a real accuracy metric. This means the models were originally overfitted, and then after doing the test_train_split and GridSearchCV, they were optimized and corrected.

In [ ]: