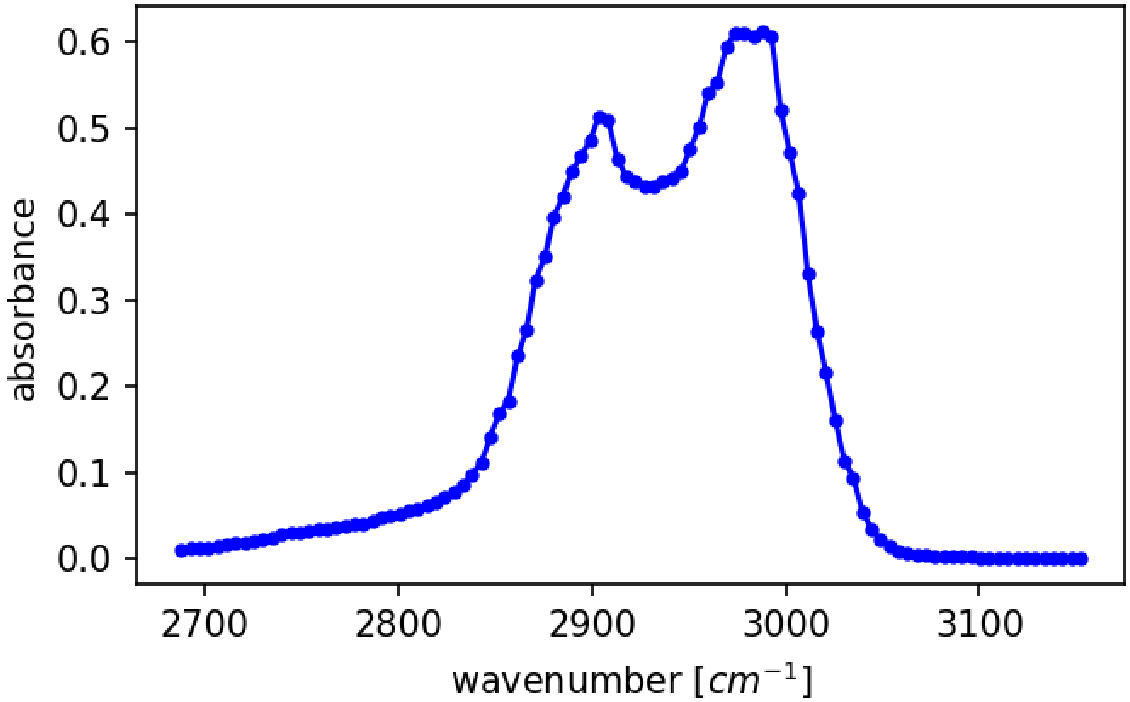
Table of Contents

Regression - Assignment 1

Data and Package Import

In [58]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import pylab as plt
```

In [59]:
```
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');
```



Linear Interpolation

Select every third datapoint from x_peak and y_peak dataset.

In [60]:
```
x_peak3rd = x_peak[::3] #used slice operator to get every third --> should ge
print(len(x_peak))
print(len(x_peak3rd))

y_peak3rd = y_peak[::3]
```

```
100
34
```

Use these datapoints to train a linear interpolation model.

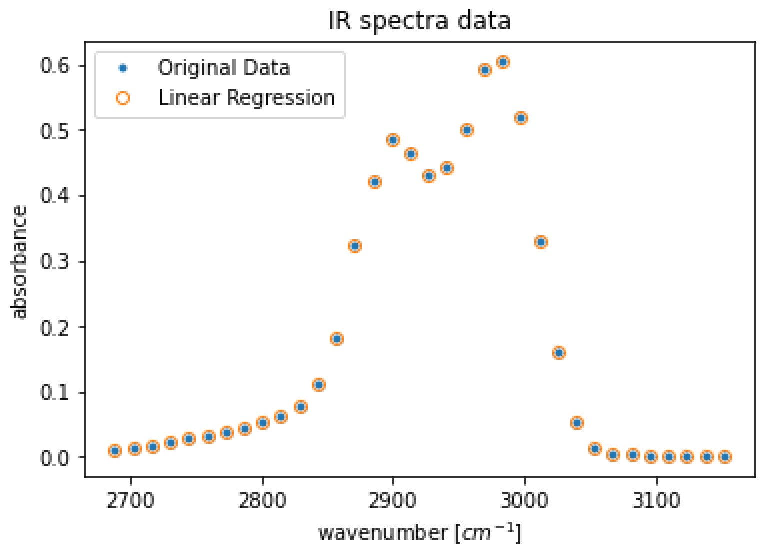
Predict the full dataset using the model and plot the result along with the original dataset.

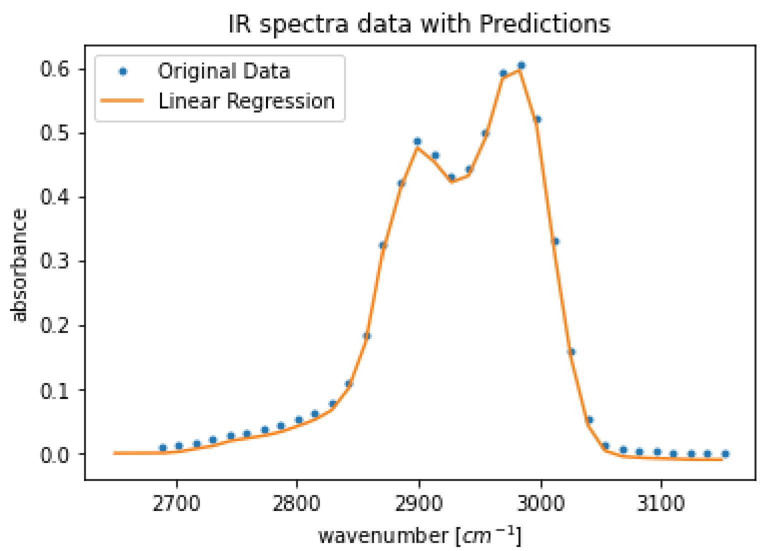
In [61]:

```
def piecewise_linear(x):
    N = len(x)
    X = np.zeros((N,N))
    for i in range(N):
        for j in range(N):
            X[i,j] = max(0, x[i] - x[j])
    return X

X = piecewise_linear(x_peak3rd)

X[:,-1] += 1 #make final column equal to 1

from sklearn.linear_model import LinearRegression

model = LinearRegression(fit_intercept = False)
model.fit(X,y_peak3rd)
r2 = model.score(X,y_peak3rd)

yhat = model.predict(X)

fig, ax = plt.subplots()
ax.plot(x_peak3rd, y_peak3rd, '.')
ax.plot(x_peak3rd, 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(r2))

def piecewise_linear(x_train, x_test = None):
    if x_test is None:
        x_test = x_train
    N = len(x_test)
    M = len(x_train)
    X = np.zeros((N,M))
    for i in range(N):
        for j in range(M):
            X[i,j] = max(0, x_test[i] - x_train[j])
    return X

x_predict = np.linspace(2650,3150,1000)

X_predict = piecewise_linear(x_peak3rd, x_predict)
yhat_predict = model.predict(X_predict)
new_r2 = model.score(X,y_peak3rd)

fig,ax = plt.subplots()
ax.plot(x_peak3rd, y_peak3rd, '.')
ax.plot(x_predict, yhat_predict, '-', markerfacecolor = 'none')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
ax.set_ylabel('absorbance')
ax.set_title('IR spectra data with Predictions')
ax.legend(['Original Data','Linear Regression'])
print('new r^2 = {}'.format(new_r2))
```

```
r^2 = 1.0
```

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





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

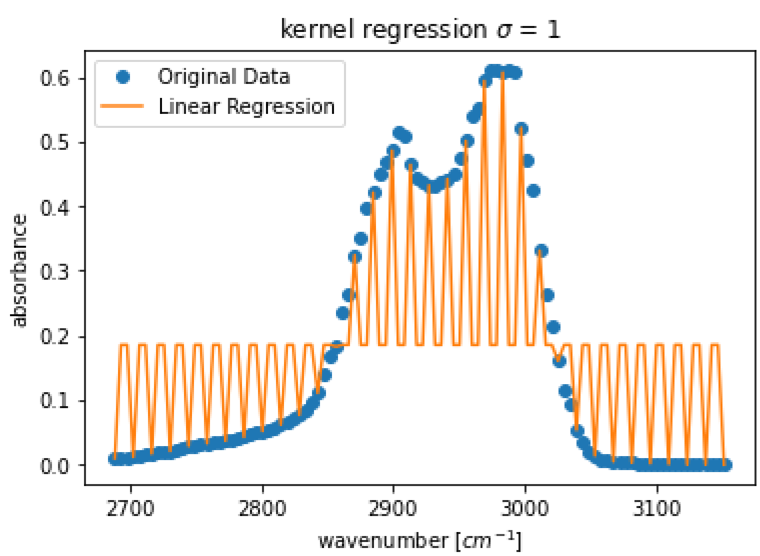
Use the same strategy as the previous exercise. Vary the width of the radial basis function with σ = [1, 10, 50, 100, 150].

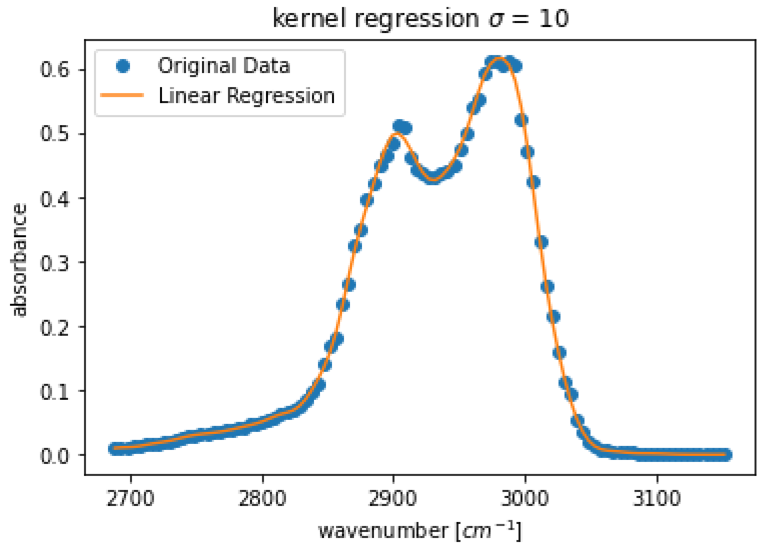
Compute the r^2 score for each using the entire dataset.

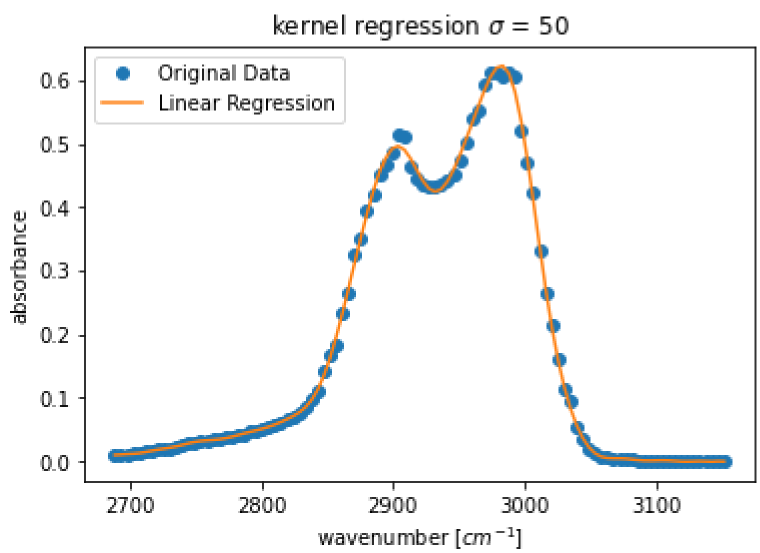
In [62]:

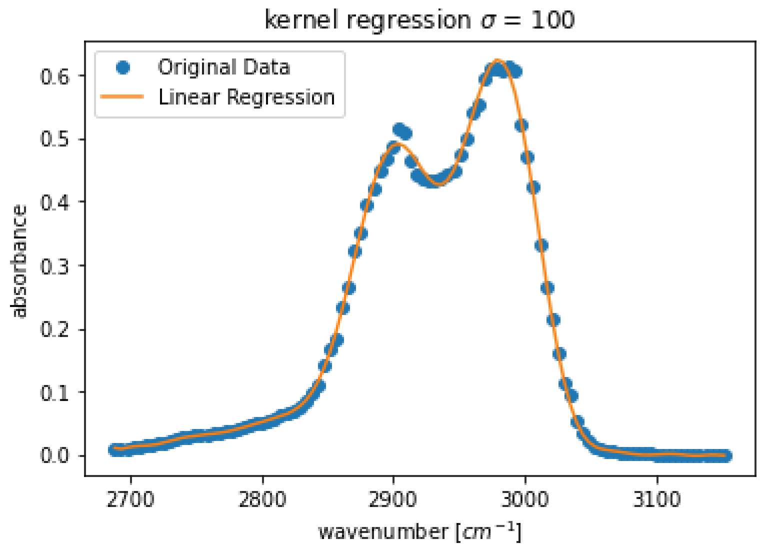
```
def rbf(x_train, x_test = None, gamma=1):
    if x_test is None:
        x_test = x_train
    N = len(x_test)
    M = len(x_train)
    X = np.zeros((N,M))
    for i in range(N):
        for j in range(M):
            X[i,j] = np.exp(-gamma*(x_test[i] - x_train[j])**2)
    return X

sigmaVec = [1, 10, 50, 100, 150]

#Iterate to find gamma for sigma, model fit, plot, print r^2 value
for i in range(len(sigmaVec)):
    gamma_i = 1./(2*sigmaVec[i]**2)
    X_train1 = rbf(x_peak3rd, x_test = x_peak, gamma = gamma_i)
    modelRBF1 = LinearRegression()
    modelRBF1.fit(X_train1, y_peak)
    r2_1 = modelRBF1.score(X_train1, y_peak)
    print('r^2 (sigma = {}) = {}'.format(str(sigmaVec[i]), r2_1))
    X_test = rbf(x_peak3rd, x_test = x_peak, gamma = gamma_i)
    yhat_rbf = modelRBF1.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(sigmaVec[i])))
    ax.legend(['Original Data','Linear Regression']);
```

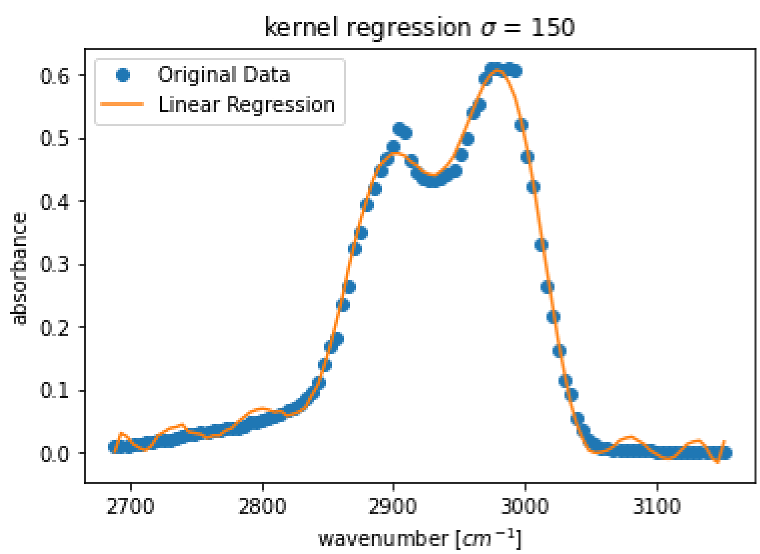
```
r^2 (sigma = 1) = 0.331854568288431 74
r^2 (sigma = 10) = 0.9992031161973826
r^2 (sigma = 50) = 0.9990939054757347
r^2 (sigma = 100) = 0.9988134360826092
r^2 (sigma = 150) = 0.9951469477765074
```




kernel regression $\sigma = 10$
Original Data
Linear Regression
absorbance
wavenumber $[cm^{-1}]$
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2700
2800
2900
3000
3100


kernel regression $\sigma = 50$
Original Data
Linear Regression
absorbance
wavenumber $[cm^{-1}]$
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2700
2800
2900
3000
3100





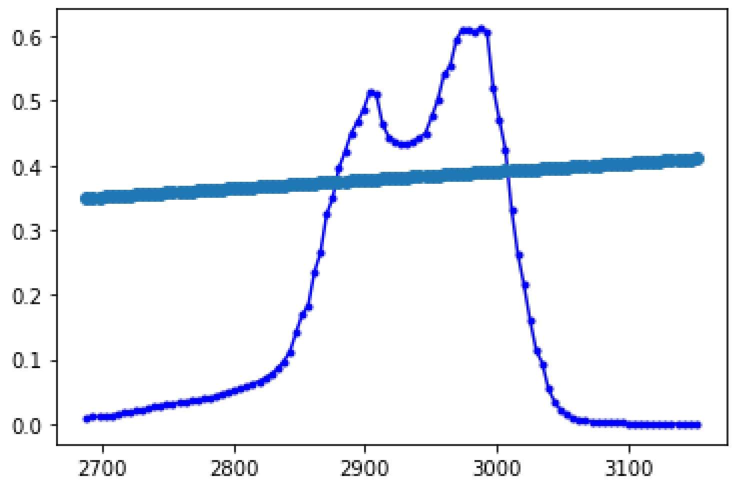
Create a model where r^2 < 0.

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

The model you use does not have to optimized using the same data that you use to compute the r^2 score.

In [63]:
```
fig,ax = plt.subplots()
ax.plot(x_peak, y_peak, '-b', marker = '.')

ybar_peak = np.mean(y_peak)
slopeM, bINT = np.polyfit(x_peak,y_peak,deg = 1)
yhat_peak = slopeM  * x_peak #I made up a model: m*x = y
SST = sum((y_peak - ybar_peak)**2)
SSE = sum((y_peak - yhat_peak)**2)

ax.plot(x_peak, yhat_peak, 'o')

R2 = (SST - SSE)/SST
print('r^2 = {}'.format(R2))
print(np.less(R2,0)) #check if value is less than zero
```

```
r^2 = -0.859848847463132
True
```



What does a negative r^2 mean?

Hint: Think about interpreting r^2 as a comparison to mean of the data.

A negative r^2 model means that the error from the model is less than just using a guess from the mean. The original data is better than what the model could output.

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 [64]:
```
def cauchy_kernel(x, x_0, gamma):
    N = len(x)
    cauchy_matrix = np.zeros((N,N))
    for i in range(N):
        for j in range(N):
            cauchy_matrix[i,j] = 1/(np.pi*gamma)*(gamma**2/((x_0[i]-x[j])**2
    return cauchy_matrix
```

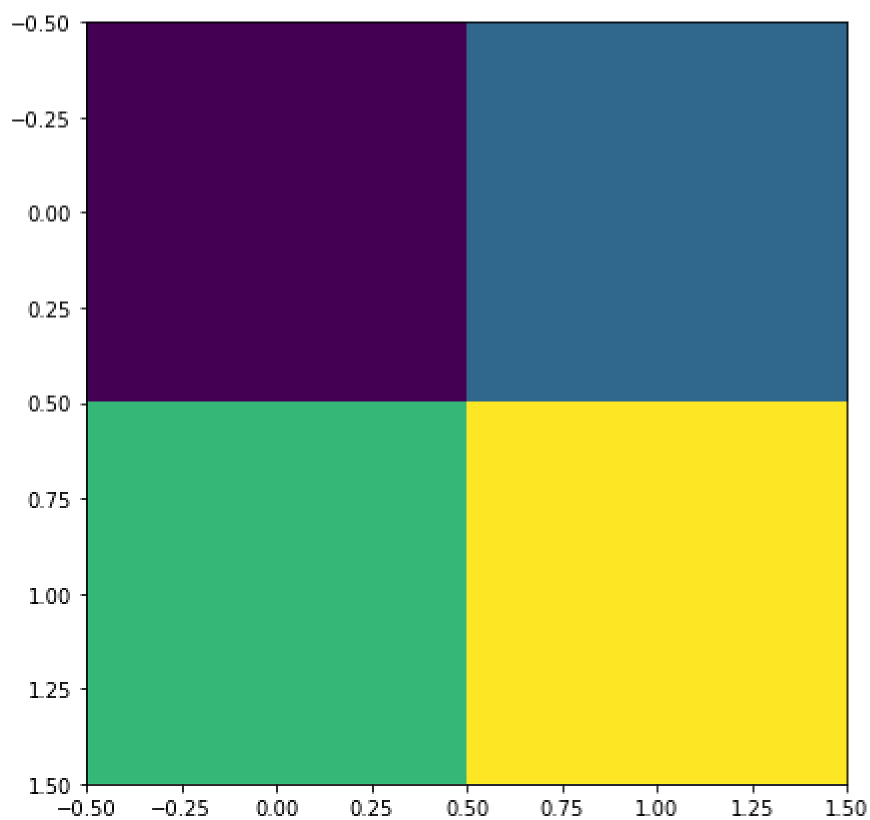
Visualize kernel matrices for the ethanol spectra dataset.

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` .

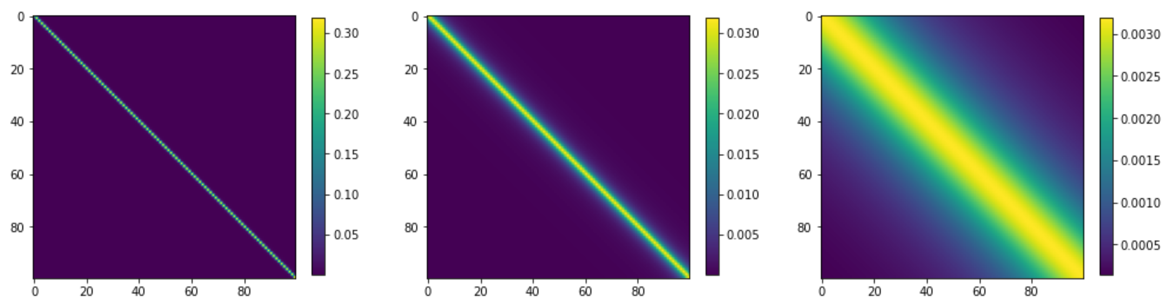
For more details, see the documentation:
[https://matplotlib.org/3.2.2/api/_as_gen/matplotlib.pyplot.imshow.html (https://matplotlib.org/3.2.2/api/_as_gen/matplotlib.pyplot.imshow.html)](https://matplotlib.org/3.2.2/api/_as_gen/matplotlib.pyplot.imshow.html).

In [65]:
```
fig, ax = plt.subplots(figsize = (7, 7))

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



In [66]:

```
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

gammaVec = [1, 10, 100]

for i in range(len(gammaVec)):
    x_test3 = piecewise_linear(x_train, x_peak)
    x_test3 = cauchy_kernel(x_train, x_peak, gammaVec[i])
    array = x_test3

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

plt.show()

x_all = cauchy_kernel(x_peak, x_peak, gammaVec[i])
fig.colorbar(image, ax = ax, shrink = 1)
plt.show()
```



Briefly discuss the structure of these matrices.

In [39]:

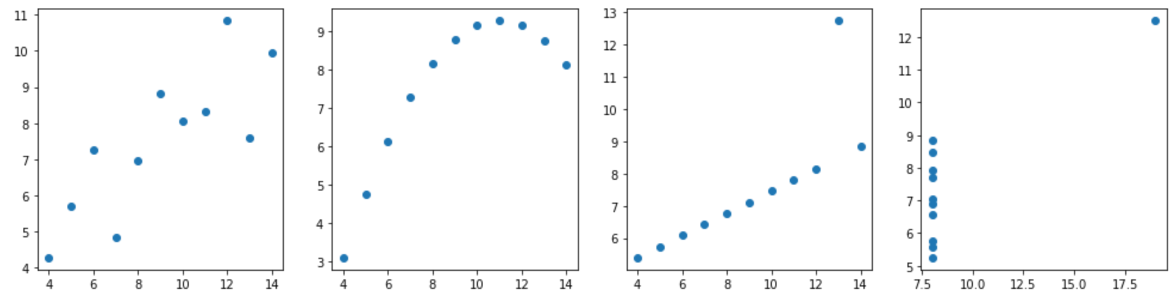
```
increases. If it were to continue to increase to infinity, I think the entire
```

```
It is evident that the width of the line grows as the gamma value increase
s. If it were to continue to increase to infinity, I think the entire pictu
re would just be the bright yellow color.
```

3. Anscomb's Quartet

In [67]:

```
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
y2_aq = np.array([9.14, 8.14, 8.74, 8.77, 9.26, 8.10, 6.13, 3.10, 9.13, 7.26,
y3_aq = np.array([7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42
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

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 mean and standard deviations of each dataset.

In [68]:
```
def calcStats(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

statsCall1 = calcStats(x_aq, y1_aq)
print("Dataset 1: mean = {:.2f}, stdev = {:.2f}, m = {:.2f}, b = {:.2f}, R2 =

statsCall2 = calcStats(x_aq, y2_aq)
print("Dataset 2: mean = {:.2f}, stdev = {:.2f}, m = {:.2f}, b = {:.2f}, R2 =

statsCall3 = calcStats(x_aq, y3_aq)
print("Dataset 3: mean = {:.2f}, stdev = {:.2f}, m = {:.2f}, b = {:.2f}, R2 =

statsCall4 = calcStats(x4_aq, y4_aq)
print("Dataset 4: mean = {:.2f}, stdev = {:.2f}, m = {:.2f}, b = {:.2f}, R2 =

avg, std, m, b, r2 = statsCall1
```

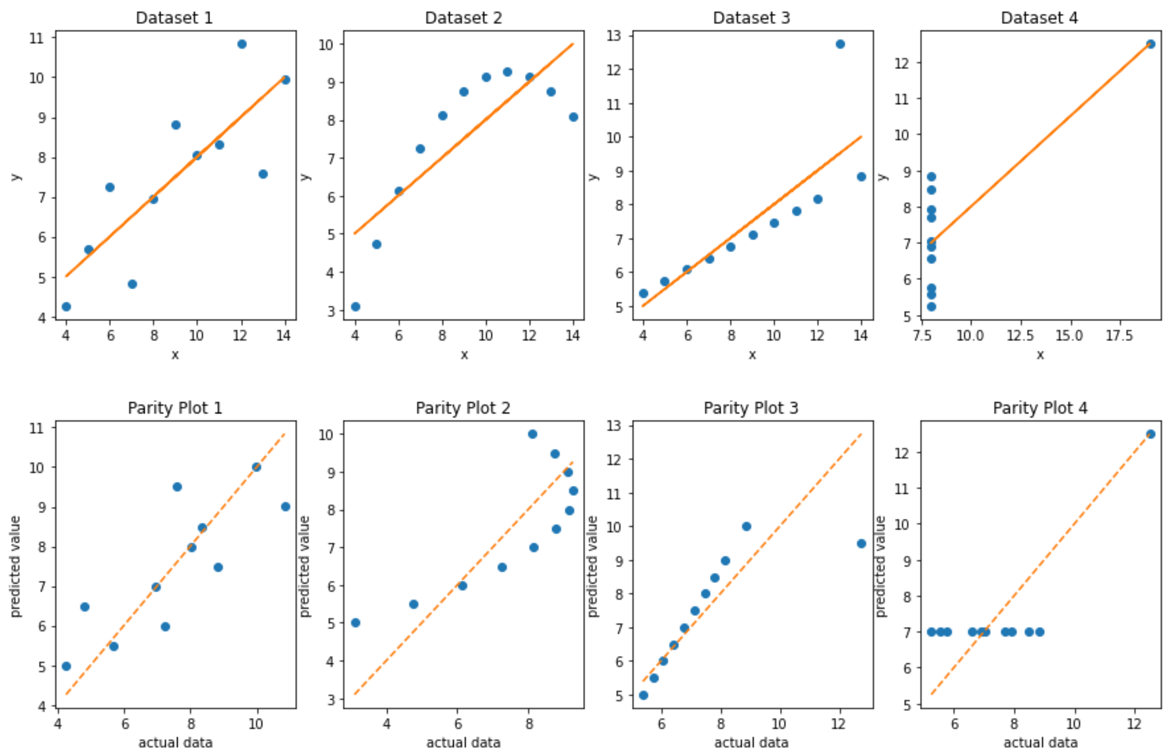
```
Dataset 1: mean = 7.50, stdev = 3.16, m = 0.50, b = 3.00, R2 =0.67
Dataset 2: mean = 7.50, stdev = 3.16, m = 0.50, b = 3.00, R2 =0.67
Dataset 3: mean = 7.50, stdev = 3.16, m = 0.50, b = 3.00, R2 =0.67
Dataset 4: mean = 7.50, stdev = 3.16, m = 0.50, b = 3.00, R2 =0.67
```

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 [69]:
```
#LINEAR REGRESSION PLOTS
fig, axes = plt.subplots(1,4,figsize=(15,4))
yhat_aq = m*x_aq + b

axes[0].plot(x_aq, y1_aq, 'o')
axes[0].plot(x_aq, yhat_aq, ls='-')
axes[0].set_xlabel('x')
axes[0].set_ylabel('y')
axes[0].set_title('Dataset 1')

axes[1].plot(x_aq, y2_aq, 'o')
axes[1].plot(x_aq, yhat_aq, ls='-')
axes[1].set_xlabel('x')
axes[1].set_ylabel('y')
axes[1].set_title('Dataset 2')

axes[2].plot(x_aq, y3_aq, 'o')
axes[2].plot(x_aq, yhat_aq, ls='-')
axes[2].set_xlabel('x')
axes[2].set_ylabel('y')
axes[2].set_title('Dataset 3')

axes[3].plot(x4_aq, y4_aq, 'o')
axes[3].plot(x4_aq, m * x4_aq + b, ls='-')
axes[3].set_xlabel('x')
axes[3].set_ylabel('y')
axes[3].set_title('Dataset 4')

plt.show()

#PARITY PLOTS
fig, axes = plt.subplots(1,4,figsize=(15,4))

axes[0].plot(y1_aq, yhat_aq, 'o')
axes[0].plot([min(y1_aq), max(y1_aq)], [min(y1_aq), max(y1_aq)], ls='--')
axes[0].set_xlabel('actual data')
axes[0].set_ylabel('predicted value')
axes[0].set_title('Parity Plot 1')

axes[1].plot(y2_aq, yhat_aq, '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 2')

axes[2].plot(y3_aq, yhat_aq, 'o')
axes[2].plot([min(y3_aq), max(y3_aq)], [min(y3_aq), max(y3_aq)], ls='--')
axes[2].set_xlabel('actual data')
axes[2].set_ylabel('predicted value')
axes[2].set_title('Parity Plot 3')

axes[3].plot(y4_aq, m * x4_aq + b, 'o')
axes[3].plot([min(y4_aq), max(y4_aq)], [min(y4_aq), max(y4_aq)], ls='--')
axes[3].set_xlabel('actual data')
axes[3].set_ylabel('predicted value')
axes[3].set_title('Parity Plot 4')
```

```
plt.show()
```



4. Assumptions for Linear Regression

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

The standard deviation of the error is a way to measure the uncertainty. Important assumptions for linear regression are normally distributed error, homoscedastic (standard deviation of Gaussian distribution independent of independent variable) error, and a linear relationship between variables.

In [ ]: