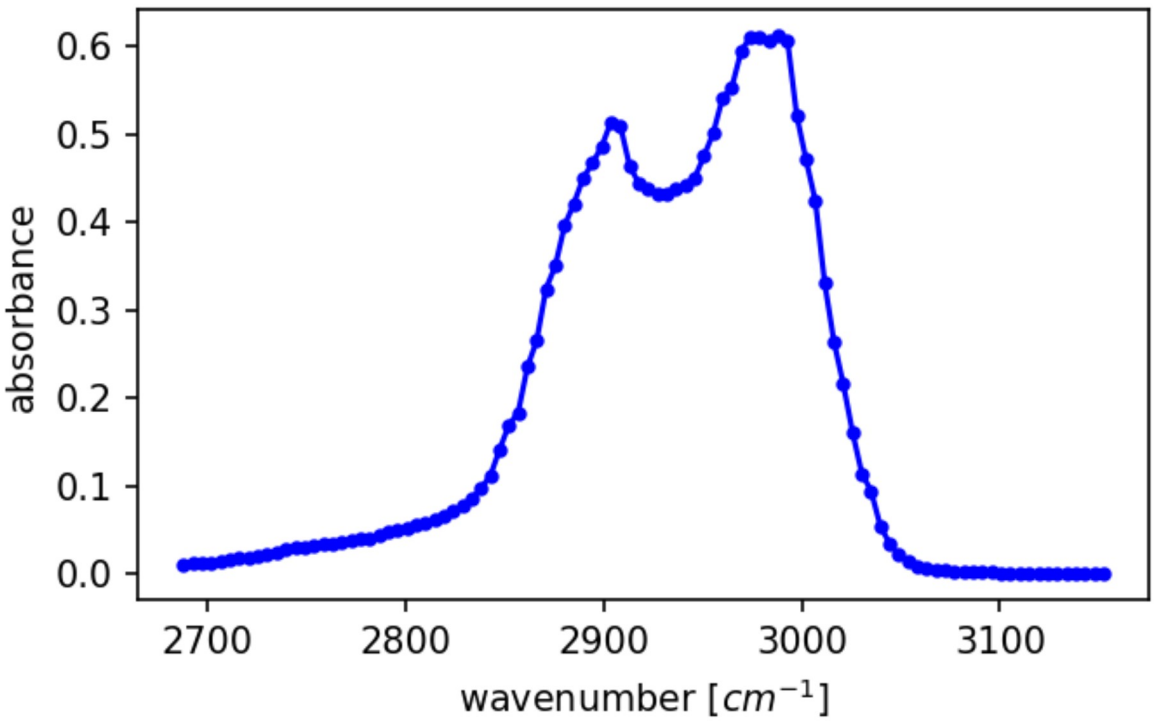
Regression - Assignment 1

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]
```

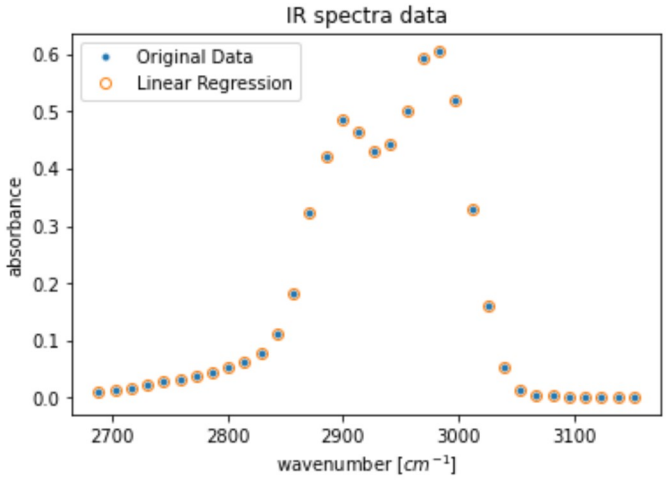
Use these datapoints to train a linear interpolation model.

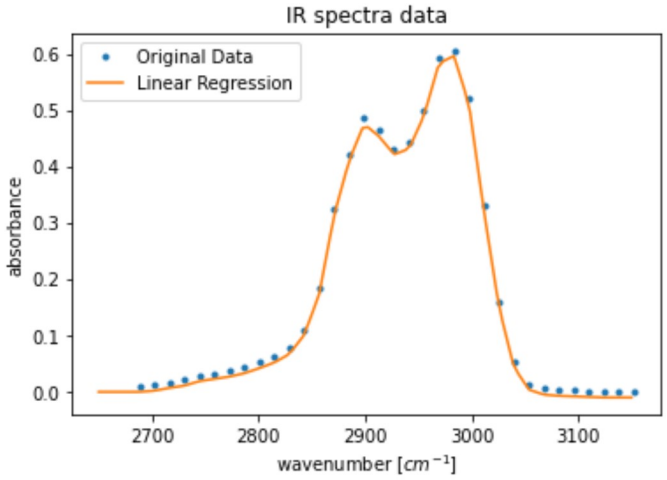
Predict the full dataset using the model and plot the result along with the original dataset.

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
```





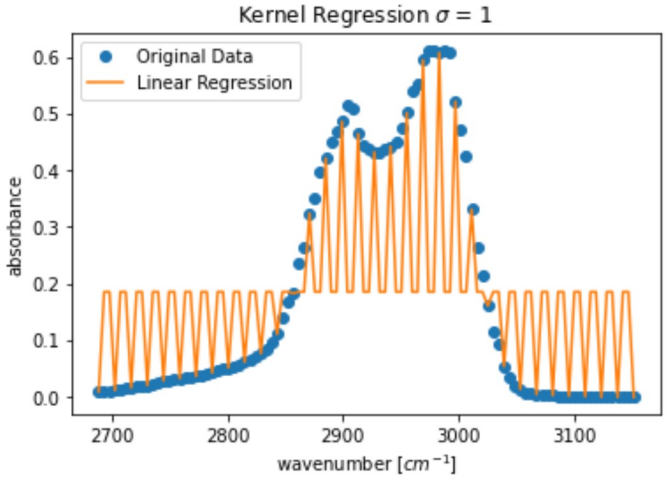
Evaluate the performance of `rbf` kernel as a function of kernel width.

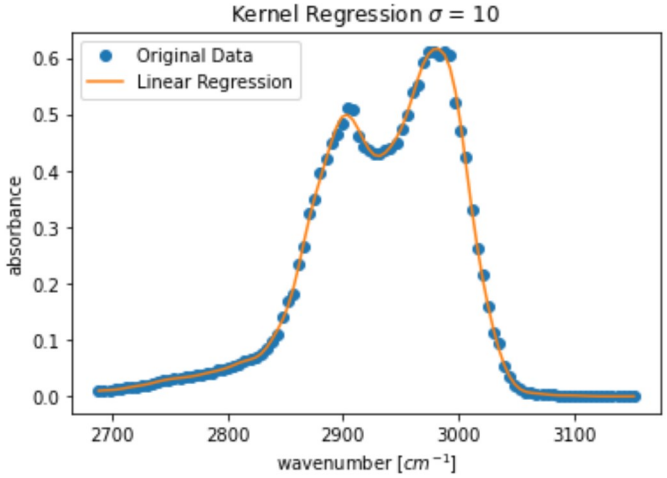
Use the same strategy as the previous exercise. Vary the width of the radial basis function with σ = [1, 10, 50, 100, 150].

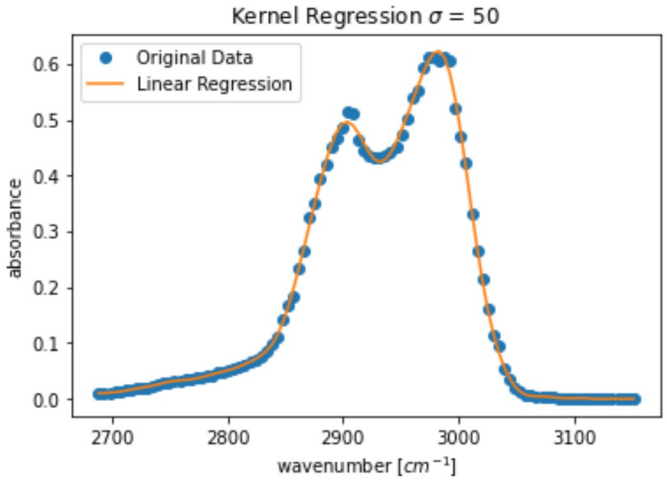
Compute the r^2 score for each using the entire dataset.

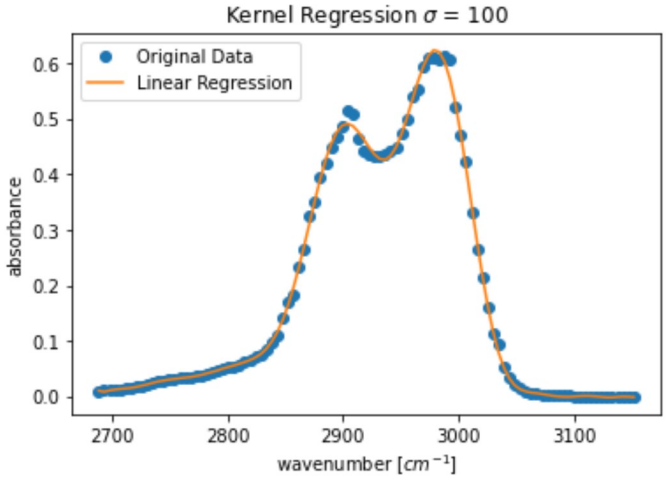
In [81]:
```
sigmas = [1, 10, 50, 100, 150]

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.
```

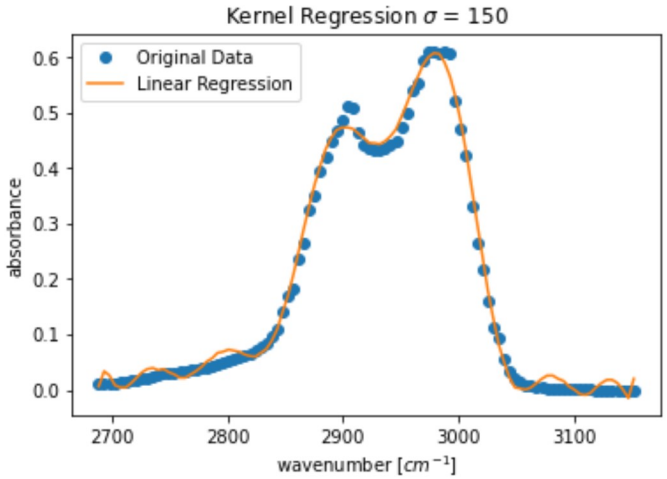
```
r^2 = 0.33185456828843174
r^2 = 0.9992031161973826
r^2 = 0.999093971097795
r^2 = 0.9988046777436903
r^2 = 0.9948178322311125
```











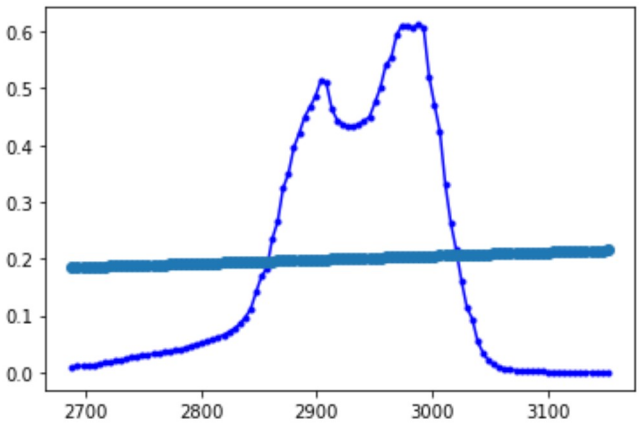
Create a model where r^2 < 0.

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
```

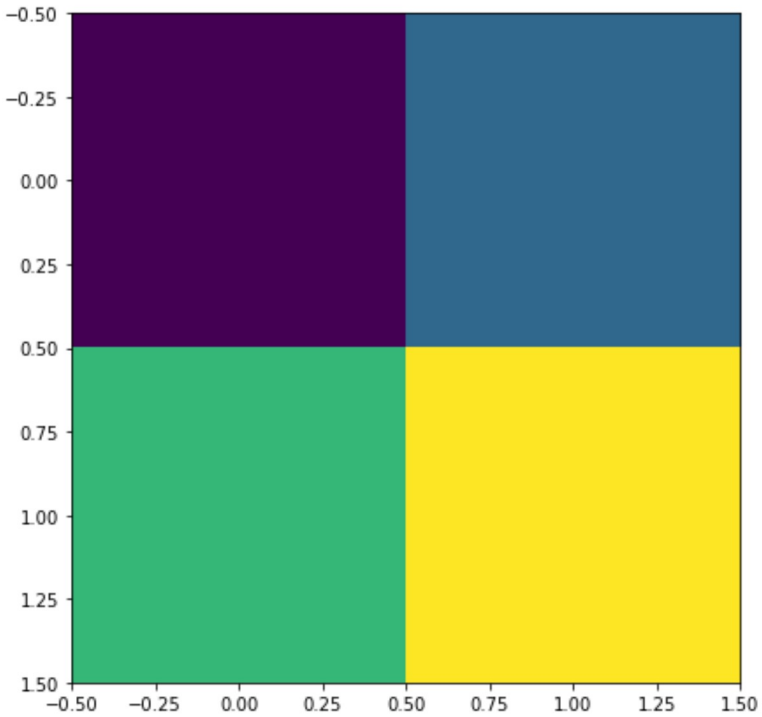
Visualize kernel matrices for the ethanol spectra dataset.

Vary the γ with [1, 10, 100].

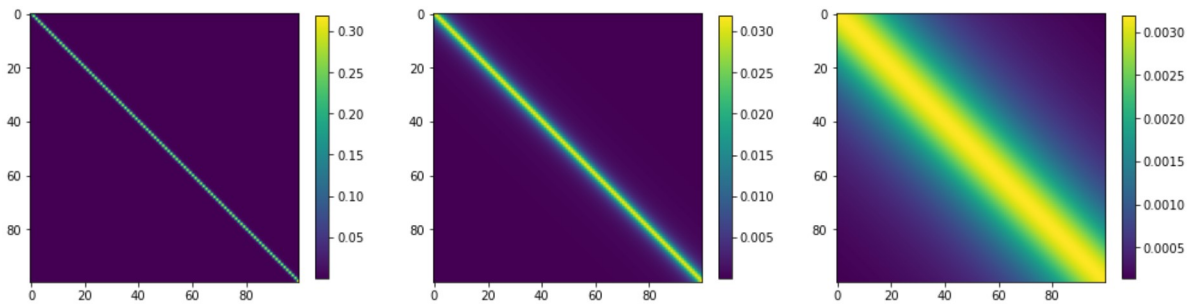
You may want to use the `plt.imshow` function to visualize the matrices. Here is an example of using `plt.imshow` .

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()
```

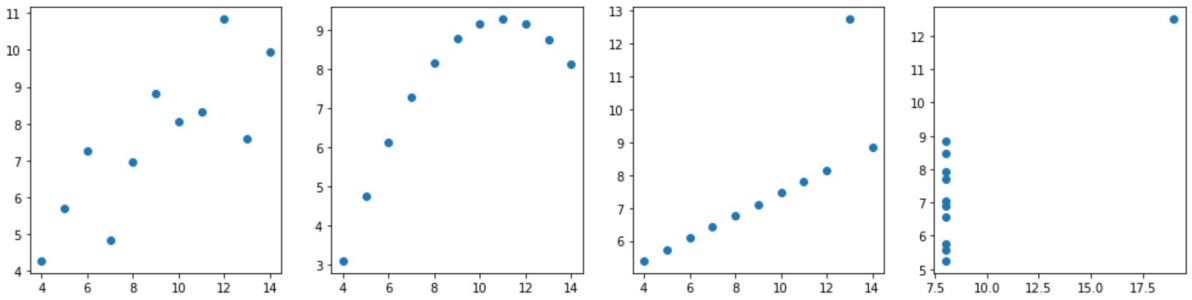


Briefly discuss the structure of these matrices.

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
```

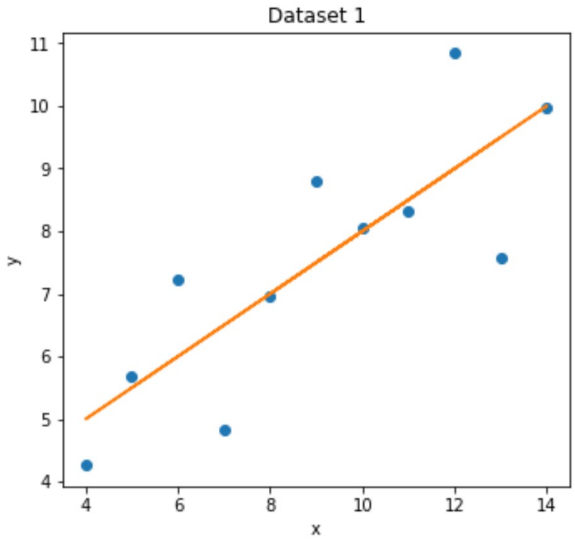
```
Dataset 1: mean=7.50, stdev=3.16, m=0.50, b=3.00, R2=0.67
Dataset 2: mean=7.50, stdev=3.16, m=0.50, b=3.00, R2=0.67
Dataset 3: mean=7.50, stdev=3.16, m=0.50, b=3.00, R2=0.67
Dataset 4: mean=7.50, stdev=3.16, m=0.50, b=3.00, R2=0.67
```

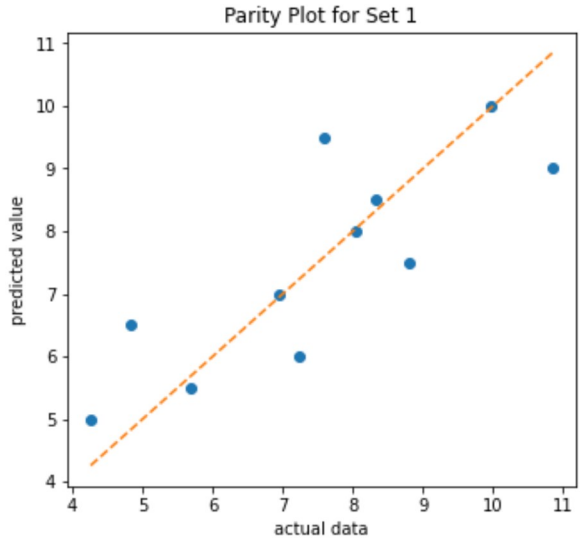
Use a linear regression to find a model $\hat{y} = mx + b$ for each dataset.

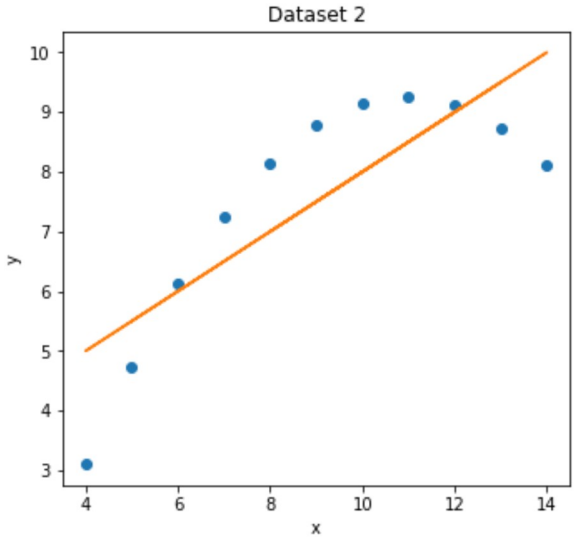
Create a parity plot between the model and the actual y values.

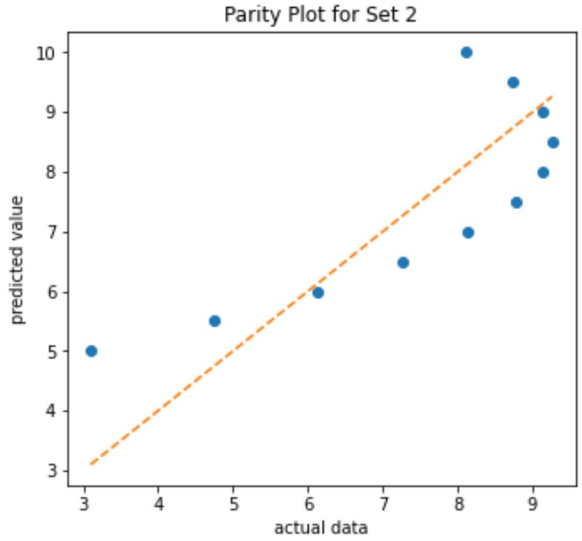
In [88]:
```
#Model 1 of 3 using x_aq
fig, axes = plt.subplots(1,2, figsize = (12, 5))
yhat = m*x_aq + b
axes[0].plot(x_aq, y1_aq, 'o')
axes[0].plot(x_aq, yhat, ls='-')
axes[0].set_xlabel('x')
axes[0].set_ylabel('y')
axes[0].set_title('Dataset 1')

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')
```

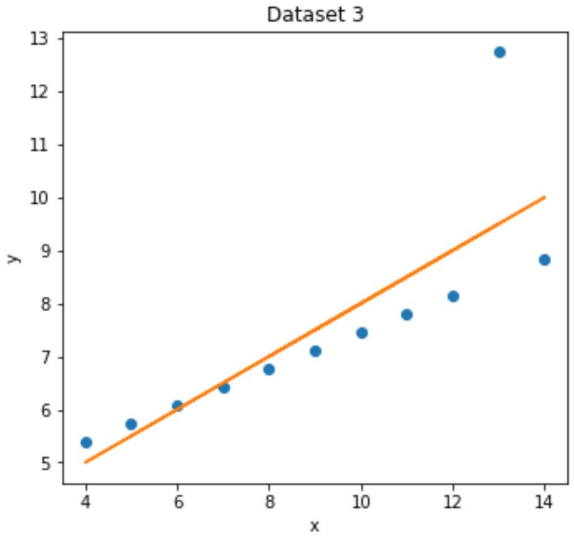
```
axes[1].plot([min(y4_aq), max(y4_aq)], [min(y4_aq), max(y4_aq)], ls='--')
axes[1].set_xlabel('actual data')
axes[1].set_ylabel('predicted value')
axes[1].set_title('Parity Plot for Second Set of X and its Y Vals')
plt.show()
```

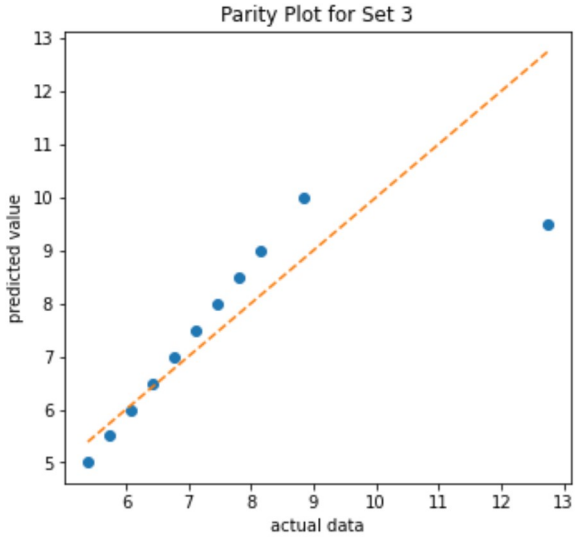

Dataset 1
y
x

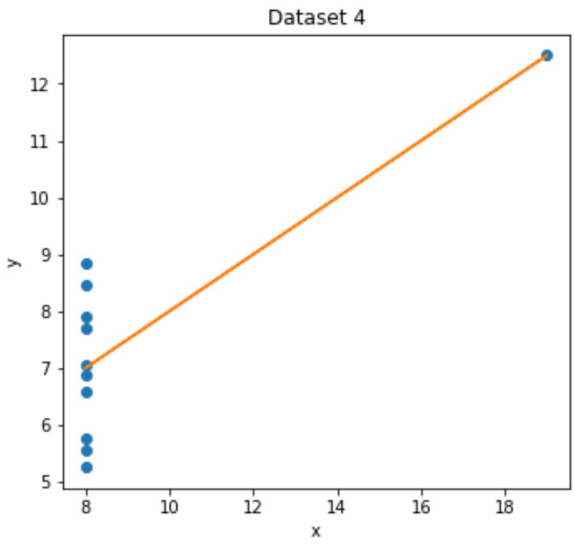

Parity Plot for Set 1
predicted value
actual data

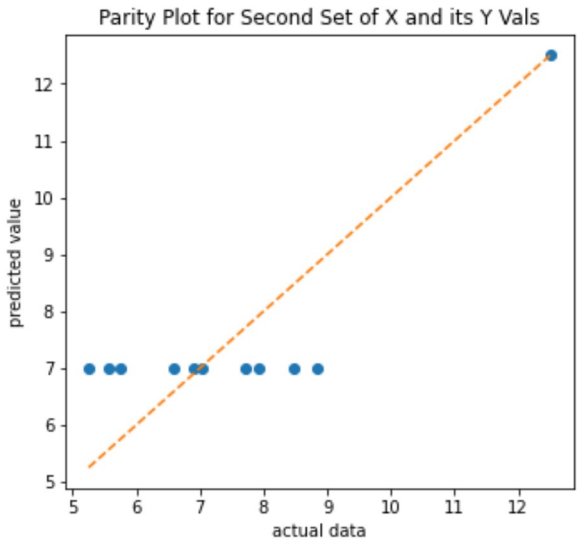

Dataset 2
y
x


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.