In [70]:
```
from __future__ import print_function
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import cv2
from IPython.display import HTML, YouTubeVideo
import matplotlib.patches as patches
from matplotlib.lines import Line2D
# import ganymede
# ganymede.configure('uav.beaver.works')
```

Enter your name below and run the cell:

Individual cells can be run with `Ctrl + Enter`

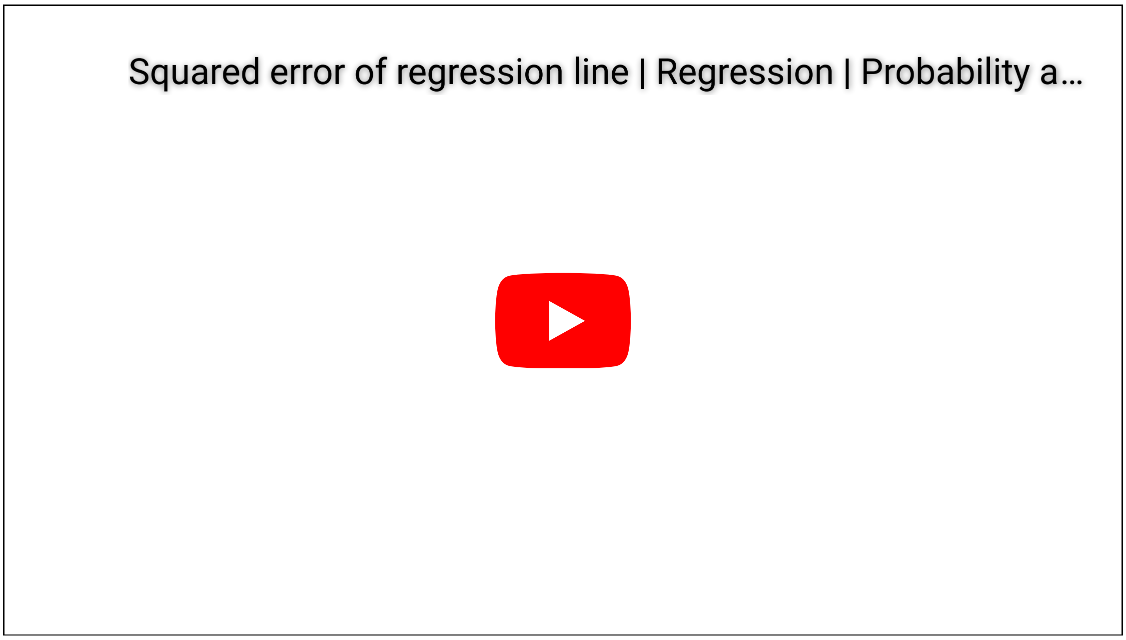
In [71]:
```
# ganymede.name('YOUR NAME HERE')
# def check(p):
    # ganymede.update(p,True)
# check(0)
```

https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/more-on-regression/v/squared-error-of-regression-line

Note: All Khan Academy content is available for free at khanacademy.org

In [72]:
```
YouTubeVideo('60vhLPS7rj4', width=560, height=315)
```

Out[72]:



In [73]:
```
YouTubeVideo('mIx20j5y9Q8', width=560, height=315)
```

Out[73]:

Proof (part 1) minimizing squared error to regression line | ...

In [74]:
```
YouTubeVideo('f6OnoxctvUk', width=560, height=315)
```

Out[74]:

Proof (part 2) minimizing squared error to regression line | ...

In [75]:
```
YouTubeVideo('u1HhUB3NP8g', width=560, height=315)
```

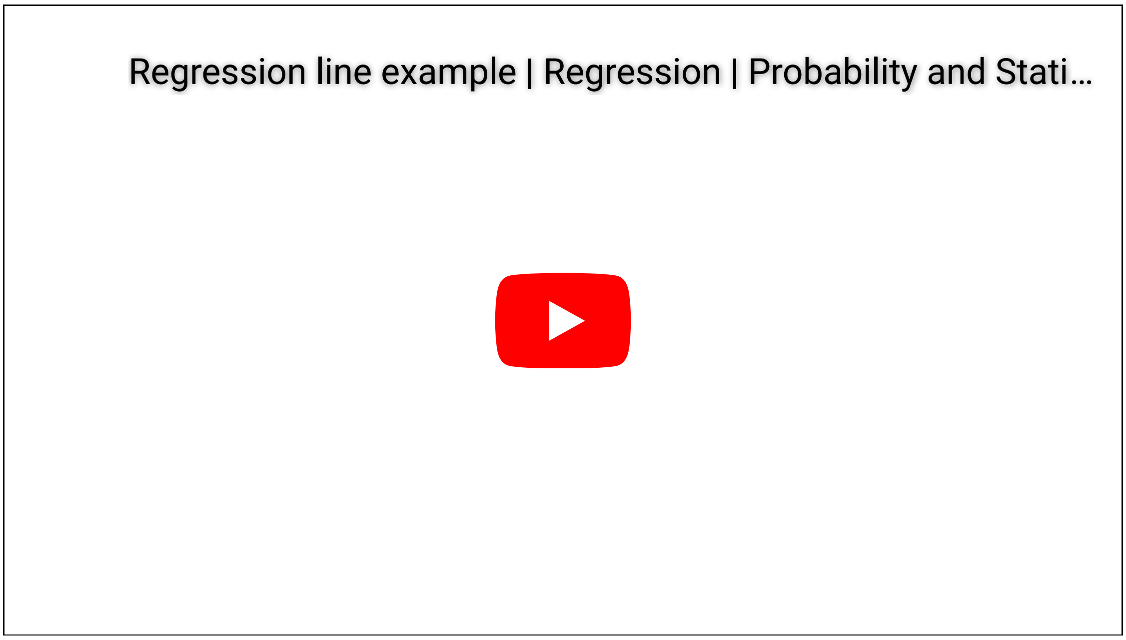
Out[75]:

In [76]:
```
YouTubeVideo('8RSTQl0bQuw', width=560, height=315)
```

Out[76]:

In [77]:
```
YouTubeVideo('GAmzwIkGFgE', width=560, height=315)
```

Out[77]:



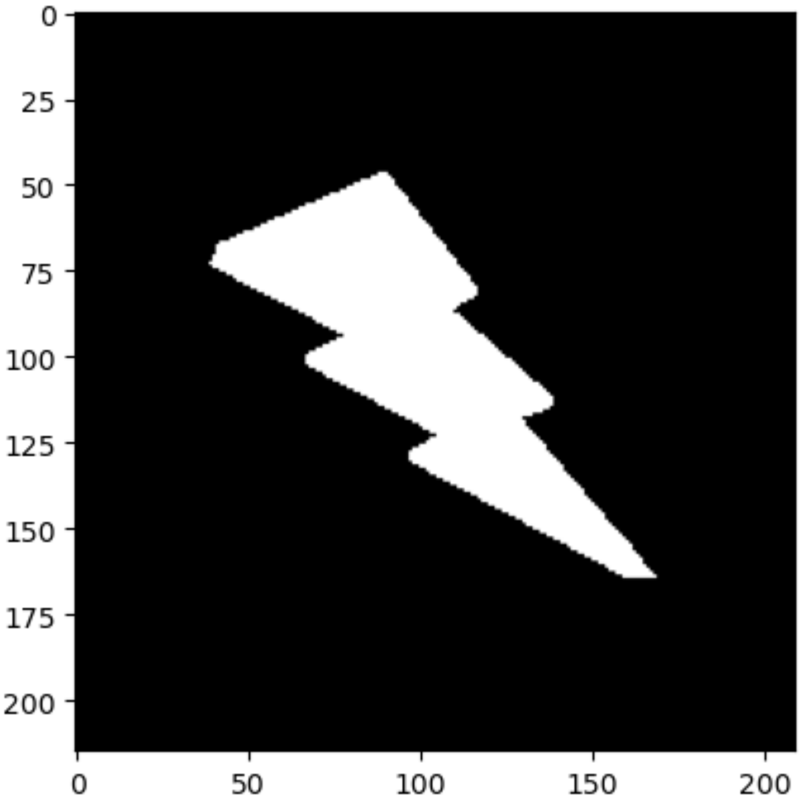
The last video is optional

In [78]:
```
YouTubeVideo('ww_yT9ckPWw', width=560, height=315)
```

Out[78]:

In [79]:
```
lightningbolt = cv2.imread('shapes/lightningbolt.png', cv2.IMREAD_GRAYSCA
_, lightningbolt = cv2.threshold(lightningbolt,150,255,cv2.THRESH_BINARY)
print(lightningbolt.shape)
fig,ax = plt.subplots()
ax.imshow(lightningbolt, cmap='gray');
# check(1)
```

```
(215, 209)
```



In [80]:
```
np.argwhere?
```

```
Signature: np.argwhere(a)
Docstring:
Find the indices of array elements that are non-zero, grouped by element.

Parameters
----------
a : array_like
    Input data.

Returns
-------
index_array : (N, a.ndim) ndarray
    Indices of elements that are non-zero. Indices are grouped by element.
    This array will have shape ``(N, a.ndim)`` where ``N`` is the number o
f
    non-zero items.

See Also
--------
where, nonzero

Notes
-----
``np.argwhere(a)`` is almost the same as ``np.transpose(np.nonzero(a))``,
but produces a result of the correct shape for a 0D array.

The output of ``argwhere`` is not suitable for indexing arrays.
For this purpose use ``nonzero(a)`` instead.

Examples
--------
>>> x = np.arange(6).reshape(2,3)
>>> x
array([[0, 1, 2],
       [3, 4, 5]])
>>> np.argwhere(x>1)
array([[0, 2],
       [1, 0],
       [1, 1],
       [1, 2]])
File:      ~/.local/lib/python3.10/site-packages/numpy/core/numeric.py
Type:      function
```

In [81]:
```
bolt = np.argwhere(lightningbolt)
bolt
```

Out[81]:
```
array([[ 47,  88],
       [ 47,  89],
       [ 47,  90],
       ...,
       [164, 166],
       [164, 167],
       [164, 168]])
```

Linear Regression

$$m = \frac{\bar{x}\bar{y} - \overline{xy}}{(\bar{x})^2 - \overline{x^2}}$$

$$b = \bar{y} - m\bar{x}$$

Question: how can we extract the xs and ys separately from the result of argwhere?

Hint: review numpy slicing by columns and rows

In [82]:
```
# TODO
# Your answer here

# Use a loop to pull out each row of lists, then indexing by using list[0
```

Question: Why would we want to convert x and y points from int values to floats?

In [83]:
```
# TODO
# Your answer here

# to make the line more accurate when graphed and to avoid any "type" err
```

In [84]:
```
def calculate_regression(points): # input is the result of np.argwhere
    # convert points to float
    points = points.astype(float) #TODO (see astype, https://docs.scipy.o

    xs = []
    ys = []
    for lists in points:
        xs.append(lists[0])
        ys.append(lists[1])

    # xs = points[:][0]
    # ys = points[:][1]
    x_mean = np.mean(xs)
    y_mean = np.mean(ys)
    xys=np.multiply(xs,ys)

    xy_mean = np.mean(xys)

    x_squared_mean = np.mean(np.multiply(xs,xs))

    m = ((x_mean*y_mean)-xy_mean)/ (((x_mean)**2)-x_squared_mean)

    b = (y_mean) - m*x_mean

    return (m,b)
```

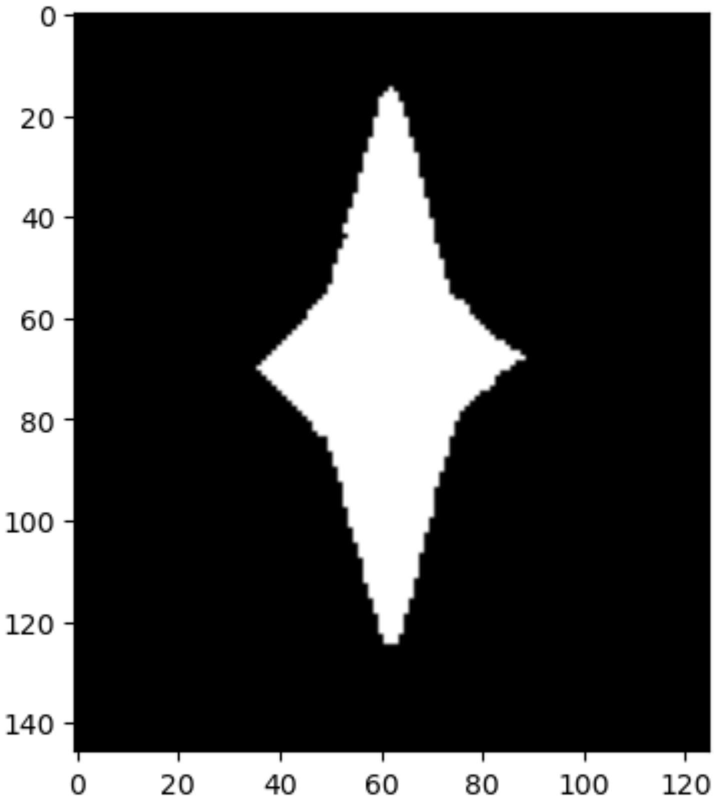
The intercept we calculated, b, may be outside of the pixel space of the image, so we must find two points inside of pixel space, (x_1, y_1) and $(x2, y2)$ which will allow us to plot our regression line on the image. It may be best to choose points on the regression line which also occur on the boundaries/extrema of the image.

In [85]:
```
def find_inliers(m, b, shape):
```

```
    xs=[0,shape[0],-b/m, (shape[1] - b) / m] #todo
    xs.sort()
    x1=xs[1]
    x2=xs[2]
    y1=m*x1+b
    y2=m*x2+b
    return x1,y1,x2,y2
```

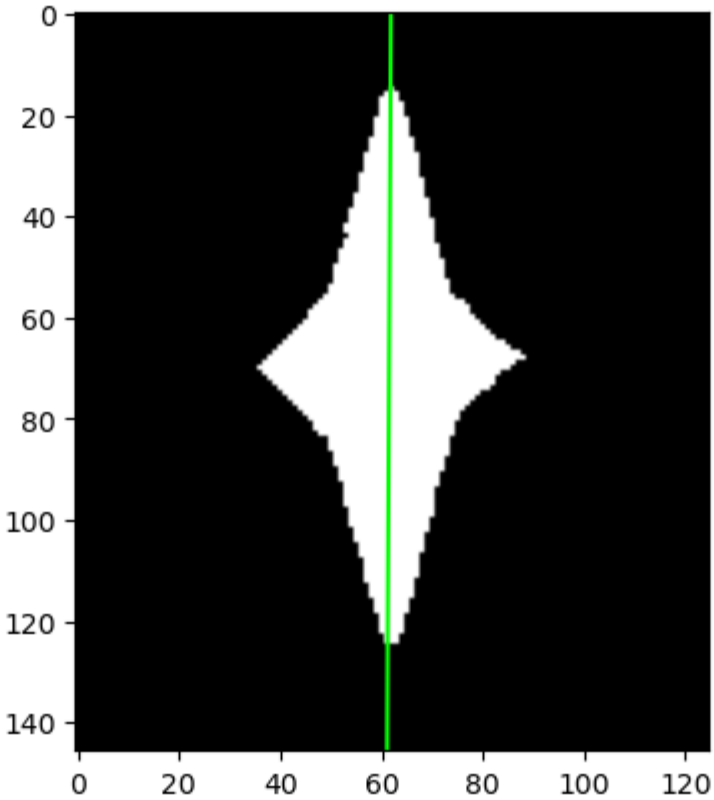
In [86]:
```
star = cv2.imread('shapes/squishedstar.png', cv2.IMREAD_GRAYSCALE)
print(star.shape)

_, star = cv2.threshold(star,125,255,cv2.THRESH_BINARY)
fig,ax = plt.subplots()
ax.imshow(star, cmap='gray');
```

```
(146, 125)
```



In [87]:
```
m,b = calculate_regression(np.argwhere(star))
_ = find_inliers(m,b, star.shape)
```

In [88]:
```
# below is an example of how to draw a random line from (10,25) to (10,55
# TODO: replace this with the result of find_inliers
# -- pay attention to the directions of the x and y axes
#    in image space, row-column space, and cartesian space
# Look at the help function for Line2D below

fig,ax = plt.subplots()
ax.imshow(star, cmap='gray');
regression = Line2D([_[1], _[3]],[_[0],_[2]], color='lime')
ax.add_line(regression);
```



In [89]:
```
Line2D?
```

```
Init signature:
Line2D(
    xdata,
    ydata,
    linewidth=None,
    linestyle=None,
    color=None,
    marker=None,
    markersize=None,
    markeredgewidth=None,
    markeredgecolor=None,
    markerfacecolor=None,
    markerfacecoloralt='none',
    fillstyle=None,
    antialiased=None,
    dash_capstyle=None,
    solid_capstyle=None,
    dash_joinstyle=None,
    solid_joinstyle=None,
    pickradius=5,
    drawstyle=None,
    markevery=None,
    **kwargs,
)
Docstring:
A line - the line can have both a solid linestyle connecting all
the vertices, and a marker at each vertex.  Additionally, the
drawing of the solid line is influenced by the drawstyle, e.g., one
can create "stepped" lines in various styles.
Init docstring:
Create a `.Line2D` instance with *x* and *y* data in sequences of
*xdata*, *ydata*.

Additional keyword arguments are `.Line2D` properties:

Properties:
    agg_filter: a filter function, which takes a (m, n, 3) float array and
a dpi value, and returns a (m, n, 3) array
    alpha: scalar or None
    animated: bool
    antialiased or aa: bool
    clip_box: `.Bbox`
    clip_on: bool
    clip_path: Patch or (Path, Transform) or None
    color or c: color
    dash_capstyle: `.CapStyle` or {'butt', 'projecting', 'round'}
    dash_joinstyle: `.JoinStyle` or {'miter', 'round', 'bevel'}
    dashes: sequence of floats (on/off ink in points) or (None, None)
    data: (2, N) array or two 1D arrays
    drawstyle or ds: {'default', 'steps', 'steps-pre', 'steps-mid', 'step
s-post'}, default: 'default'
    figure: `.Figure`
    fillstyle: {'full', 'left', 'right', 'bottom', 'top', 'none'}
    gid: str
    in_layout: bool
    label: object
    linestyle or ls: {'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
    linewidth or lw: float
    marker: marker style string, `~.path.Path` or `~.markers.MarkerStyle`
    markeredgecolor or mec: color
```

```
    markeredgewidth or mew: float
    markerfacecolor or mfc: color
    markerfacecoloralt or mfcalt: color
    markersize or ms: float
    markevery: None or int or (int, int) or slice or list[int] or float or
(float, float) or list[bool]
    path_effects: `.AbstractPathEffect`
    picker: float or callable[[Artist, Event], tuple[bool, dict]]
    pickradius: float
    rasterized: bool
    sketch_params: (scale: float, length: float, randomness: float)
    snap: bool or None
    solid_capstyle: `.CapStyle` or {'butt', 'projecting', 'round'}
    solid_joinstyle: `.JoinStyle` or {'miter', 'round', 'bevel'}
    transform: unknown
    url: str
    visible: bool
    xdata: 1D array
    ydata: 1D array
    zorder: float

See :meth:`set_linestyle` for a description of the line styles,
:meth:`set_marker` for a description of the markers, and
:meth:`set_drawstyle` for a description of the draw styles.
File:           /usr/lib/python3/dist-packages/matplotlib/lines.py
Type:           type
Subclasses:     _AxLine, Line3D
```

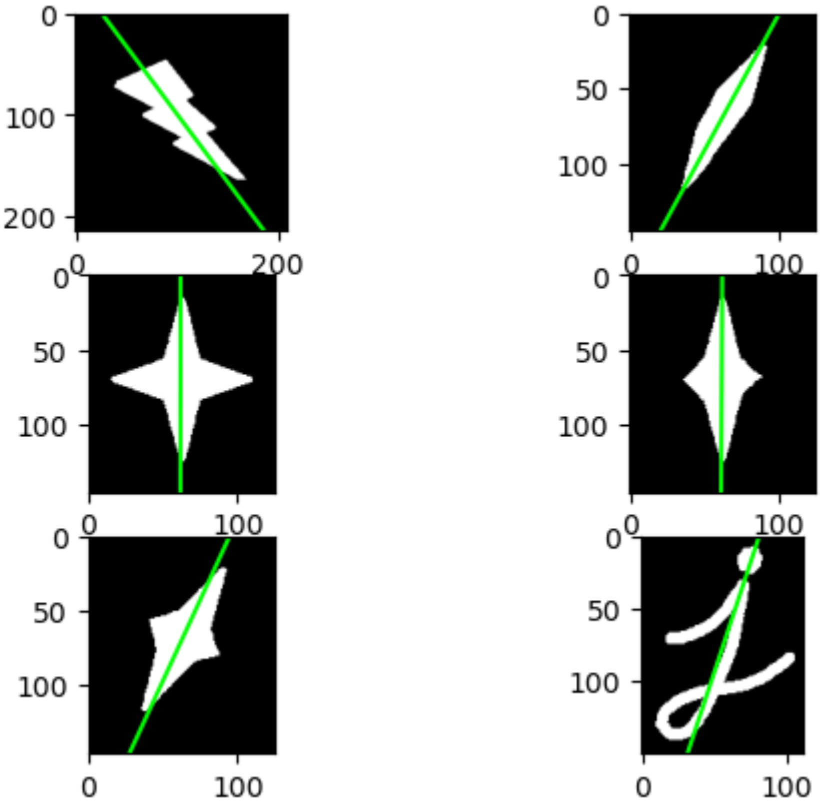
TODO

1. Run your linear regression algorithm on the following images.
2. Plot each of the results.
3. Include each result in your submitted PDF.

In [90]:

```
lightningbolt      = cv2.imread('shapes/lightningbolt.png', cv2.IMREAD_GR
blob               = cv2.imread('shapes/blob.png', cv2.IMREAD_GRAYSCALE)
star               = cv2.imread('shapes/star.png', cv2.IMREAD_GRAYSCALE)
squishedstar       = cv2.imread('shapes/squishedstar.png', cv2.IMREAD_GRA
squishedturnedstar = cv2.imread('shapes/squishedturnedstar.png', cv2.IMRE
letterj = cv2.imread('shapes/letterj.png', cv2.IMREAD_GRAYSCALE)

# _11, lightningbolt = cv2.threshold(lightningbolt,125,255,cv2.THRESH_BIN
# ax.imshow(lightningbolt, cmap='gray');

# m1,b1 = calculate_regression(np.argwhere(lightningbolt))
# _1 = find_inliers(m1,b1, lightningbolt.shape)

# fig1,ax1 = plt.subplots()
# ax.imshow(lightningbolt, cmap='gray');
# regression1 = Line2D([_1[1], _1[3]],[_1[0],_1[2]], color='lime')
# ax.add_line(regression1);

# ########################

# m2,b2 = calculate_regression(np.argwhere(blob))
# _2 = find_inliers(m2,b2, blob.shape)

# fig2,ax2 = plt.subplots()
```

```
# ax.imshow(blob, cmap='gray');
# regression2 = Line2D([_2[1], _2[3]],[_2[0],_2[2]], color='lime')
# ax.add_line(regression2);

# ########################

# m3,b3 = calculate_regression(np.argwhere(star))
# _3 = find_inliers(m3,b3, star.shape)

# fig3,ax3 = plt.subplots()
# ax.imshow(star, cmap='gray');
# regression3 = Line2D([_3[1], _3[3]],[_3[0],_3[2]], color='lime')
# ax.add_line(regression3);

# ########################

# m4,b4 = calculate_regression(np.argwhere(squishedstar))
# _4 = find_inliers(m4,b4, squishedstar.shape)

# fig4,ax4 = plt.subplots()
# ax.imshow(squishedstar, cmap='gray');
# regression4 = Line2D([_4[1], _4[3]],[_4[0],_4[2]], color='lime')
# ax.add_line(regression4);

# #######################

# m5,b5 = calculate_regression(np.argwhere(squishedturnedstar))
# _5 = find_inliers(m5,b5, squishedturnedstar.shape)

# fig5,ax5 = plt.subplots()
# ax.imshow(squishedturnedstar, cmap='gray');
# regression5 = Line2D([_5[1], _5[3]],[_5[0],_5[2]], color='lime')
# ax.add_line(regression5);

# #######################

# m6,b6 = calculate_regression(np.argwhere(letterj))
# _6 = find_inliers(m6,b6, letterj.shape)

# fig6,ax6 = plt.subplots()
# ax.imshow(letterj, cmap='gray');
# regression6 = Line2D([_6[1], _6[3]],[_6[0],_6[2]], color='lime')
# ax.add_line(regression6);

fig,ax = plt.subplots(nrows=3, ncols=2)
images = [lightningbolt, blob, star, squishedstar, squishedturnedstar, le
for a,i in zip(ax.flatten(), images):
    _, i = cv2.threshold(i,125,255,cv2.THRESH_BINARY)
    m,b = calculate_regression(np.argwhere(i))
    _ = find_inliers(m,b, i.shape)
    a.imshow(i, cmap='gray');
    regression = Line2D([_[1], _[3]],[_[0],_[2]], color='lime')
    a.add_line(regression);
```



When you are done:

You should have six images with regression lines plotted on top of them.

1. Double-check that you filled in your name at the top of the notebook!
2. Click `File -> Export Notebook As -> PDF`
3. Email the PDF to `YOURTEAMNAME@beaver.works`

Stretch goal

Implement a machine learning algorithm!

Random **Sa**mple **C**onsensus, commonly referred to as *RANSAC*, is one of the most widely used machine learning algorithms. In essence, it is a 'guess and check' algorithm. Take a small random sample of your data - two points in this case. Next, define a line through those two points. After doing so, count the number of *inliers*, or points closest to that line (euclidean distance is one way to do this).

https://en.wikipedia.org/wiki/Random_sample_consensus

Implement RANSAC for linear regression, and run it on all of your images.