Assignment 1

Please fill out the relevant cells below according to the instructions. When done, save the notebook and export it to PDF, upload both the `ipynb` and the PDF file to Canvas.

Group Members

Group submission is highly encouraged. If you submit as part of group, list all group members here. Groups can comprise up to 4 students.

- Vanessa Roser
- Declan Campbell

Problem 1: Central Limit Theorem (2pts)

Use `scipy.stats` to draw N samples from the uniform and the Cauchy distribution. Confirm whether the mean μ of these samples (which is itself a RV) has a distribution $p(\mu)$ that converges to a normal distribution when $N \rightarrow \infty$.

A simple way of testing for normality of the distribution of means is the 68–95–99.7 rule, i.e. you expect that there are only about 5% of the means (of a draw of N samples) that deviate from $\mathrm{mean}(\mu)$ by more than $2\sqrt{\mathrm{var}(\mu)}$.

Visualization can be helpful but is itself not a sufficient confirmation of normality!

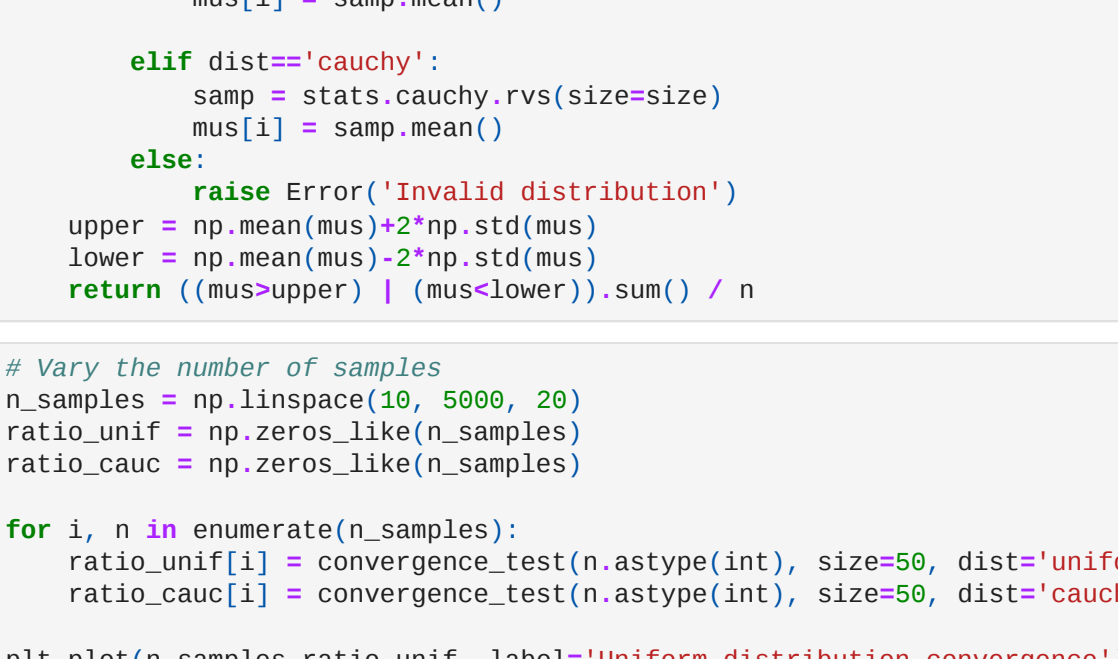
In [1]:
```
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt
```

```
/home/declan/miniconda3/envs/analysis/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (d
etected version 1.23.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

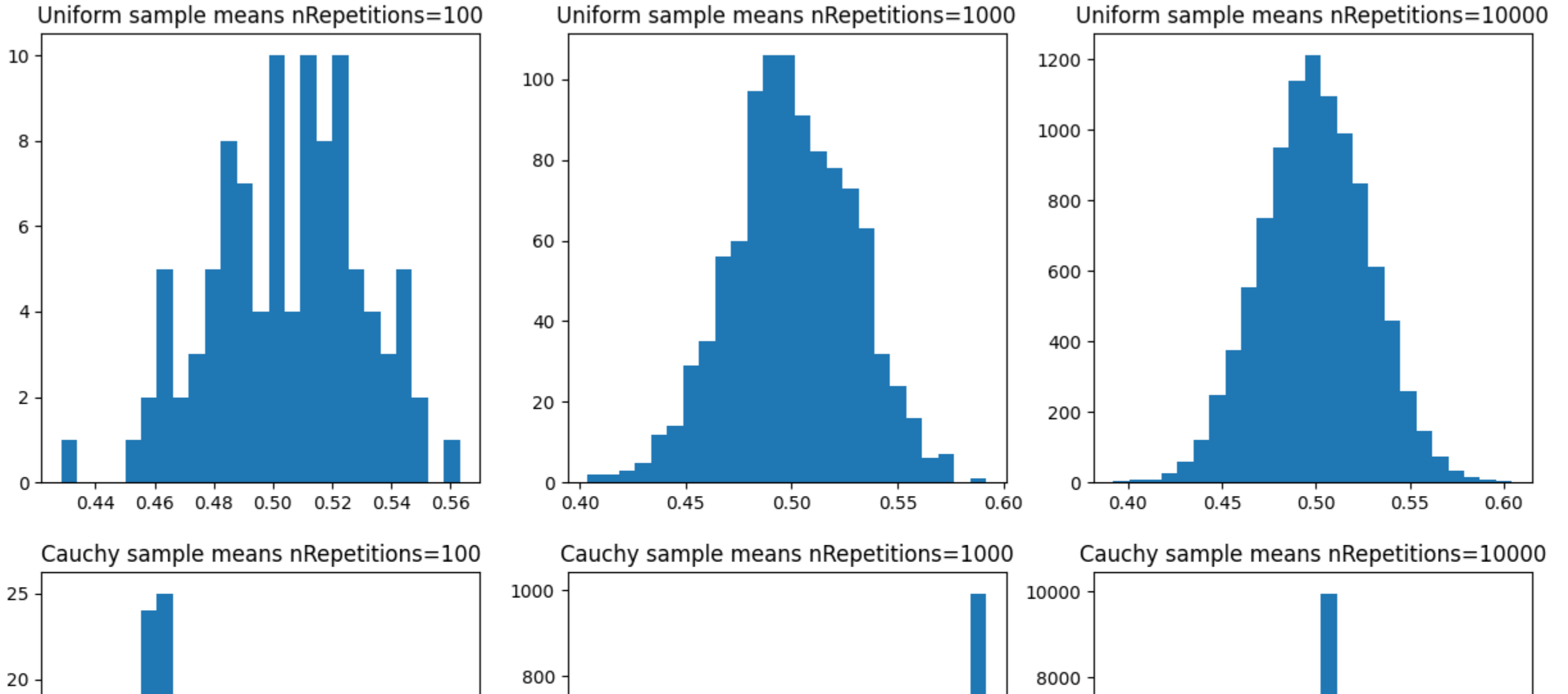
In [2]:
```
def convergence_test(n, size=100, dist='uniform'):
    '''Sample from the specified distribution n times.
    '''
    mus = np.zeros([n])
    for i in range(n):
        if dist=='uniform':
            samp = stats.uniform.rvs(size=size)
            mus[i] = samp.mean()

        elif dist=='cauchy':
            samp = stats.cauchy.rvs(size=size)
            mus[i] = samp.mean()
        else:
            raise Error('Invalid distribution')
    upper = np.mean(mus)+2*np.std(mus)
    lower = np.mean(mus)-2*np.std(mus)
    return ((mus>upper) | (mus<lower)).sum() / n
```

In [20]:
```
# Vary the number of samples
n_samples = np.linspace(10, 5000, 20)
ratio_unif = np.zeros_like(n_samples)
ratio_cauc = np.zeros_like(n_samples)

for i, n in enumerate(n_samples):
    ratio_unif[i] = convergence_test(n.astype(int), size=50, dist='uniform')
    ratio_cauc[i] = convergence_test(n.astype(int), size=50, dist='cauchy')

plt.plot(n_samples,ratio_unif, label='Uniform distribution convergence')
plt.plot(n_samples,ratio_cauc, label='Cauchy distribution convergence')
plt.axhline(0.05,color='r', ls=':', label='0.05')
plt.legend()
plt.title('% samples >2SD from mean')
print("NOTE: Since the Cauchy distribution has neither a mean nor a variance, the central limit theorem does not apply.")
```

```
NOTE: Since the Cauchy distribution has neither a mean nor a variance, the central limit theorem does not apply.
```



In [4]:
```
fig, axes = plt.subplots(2, 3, figsize=(15,10))
for i, n in enumerate([100, 1000, 10000]):
    unif_means = []
    cauc_means = []
    for _ in range(n):
        unif_means.append(stats.uniform.rvs(size=100).mean())
        cauc_means.append(stats.cauchy.rvs(size=100).mean())
    axes[0, i].hist(unif_means, bins=25)
    axes[0, i].set_title(f'Uniform sample means nRepetitions={n}')
    axes[1, i].hist(cauc_means, bins=25)
    axes[1, i].set_title(f'Cauchy sample means nRepetitions={n}')
```



Problem 2: Hereditary Probabilities (3pts)

The height of children is related to their parents: tall parents tend to have tall children. The same is true for small parents and small children. Over the course of just a few generations the standard deviation in children's heights should therefore get larger and larger. But this is not the case! The distribution of heights of children at fixed age is well described by a Normal and has been remarkable stable over hundreds of years (improvements in nutrition have led to height increases overall, but the standard deviation remains stable). Something does not add up! Francis Galton thought so, too, in a study in 1885.

Step 1 (1pt):

Load the data he had assembled, from the file `Galton.txt`, into an array. Use `numpy.genfromtxt`, and make use of its arguments `names=True` and `dtype=None` to read in the column names from the header and choose the data type on its own as needed. You will get the columns

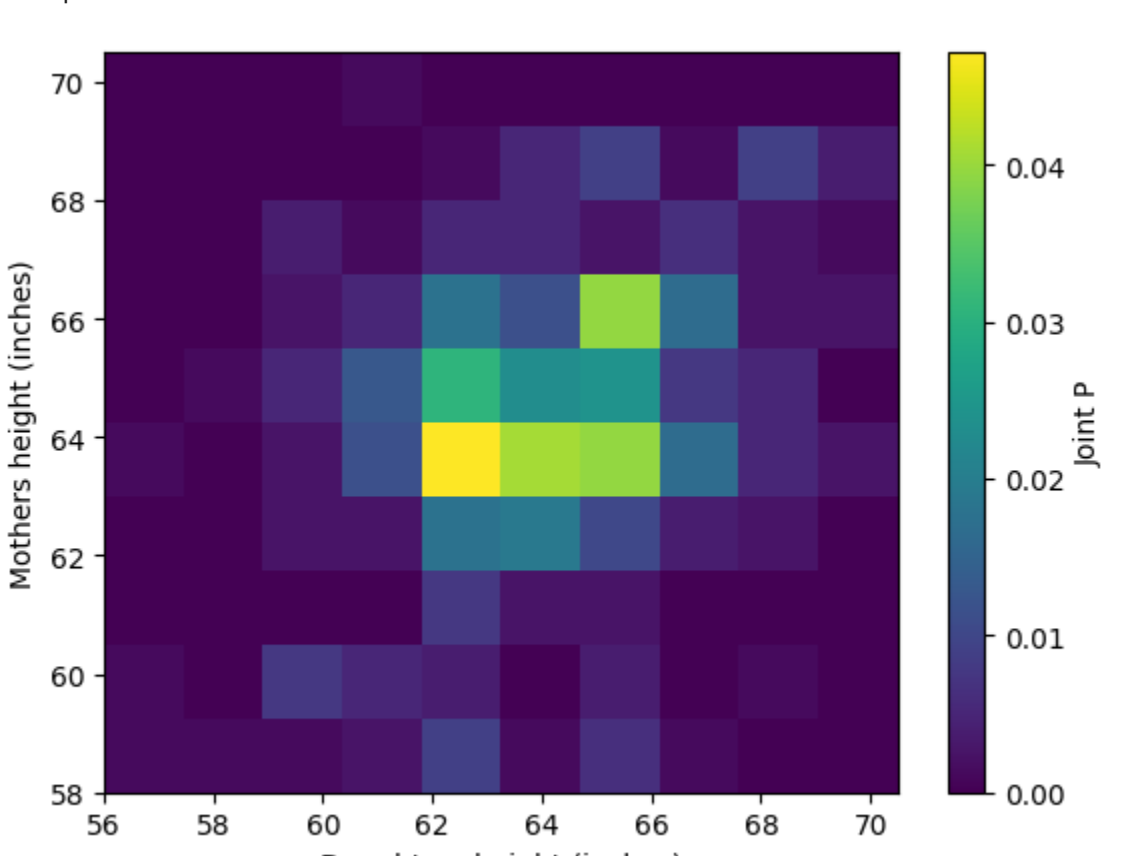
- `Family`: The family that the child belongs to, labeled from 1 to 204 and 136A
- `Father`: The father's height, in inches
- `Mother`: The mother's height, in inches
- `Gender`: The gender of the child, male (M) or female (F)
- `Height`: The height of the child, in inches (presumably fully grown)
- `Kids`: The number of kids in the family of the child

Make a visualization of the joint distribution of X, the parent's height (pick either father or mother), and Y, the children's height (pick either son or daughter).

Tip: The `matplotlib.hist2d` is useful. Don't forget labels and units.

In [16]:
```
import pandas as pd

data = pd.read_csv('Galton_1.txt', sep='\t')

#select for only mothers and daughters
fheight = data[data.Gender == 'F']['Height'].values
fmother = data[data.Gender == 'F']['Mother'].values
ax = plt.hist2d(fheight, fmother, density=True)
plt.ylabel('Mothers height (inches)')
plt.xlabel('Daughters height (inches)')
plt.colorbar(label='Joint P')
```

Out[16]:
```
<matplotlib.colorbar.Colorbar at 0x7fdb4b698d30>
```



Step 2 (2pts)

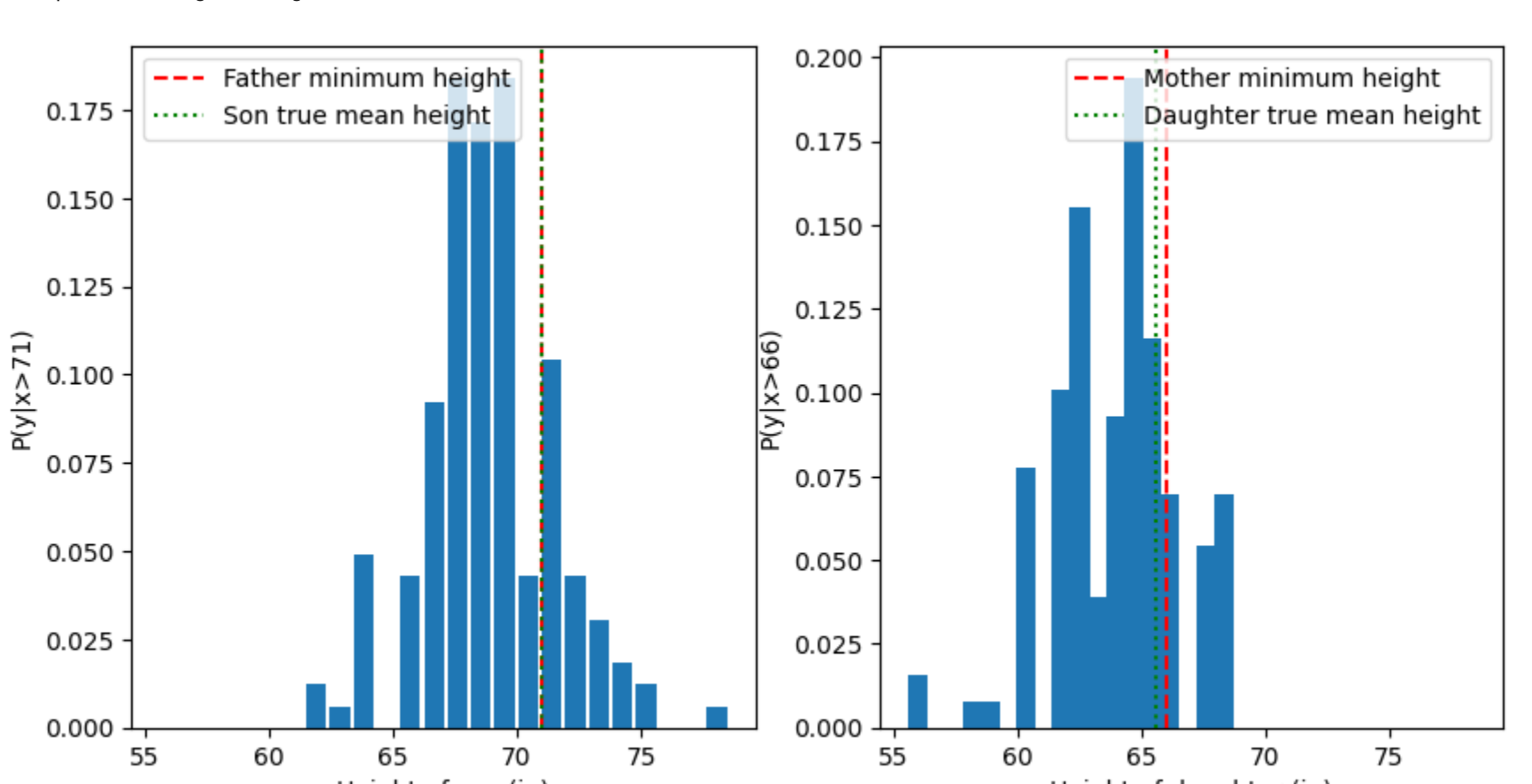
Select X and Y such that they have the same gender. Now compute the conditional distribution $p(y \mid x > 71)$ (for fathers and sons) or $p(y \mid x > 66)$ (for mothers and daughters). Plot their histograms and compute their means.

Answer these three questions:

- What do you find?
- With the same data, can you think of a way to test whether tall parents are causally responsible for their children being less tall?
- If there is no causal connection, what does that mean for conditioning on extreme events?

In [18]:
```
# SONS PLOTTING
Ysons = data[data['Gender'] == 'M']['Height'].values
Xdads = data[data['Gender'] == 'M']['Father'].values

fig, ax = plt.subplots(1,2,figsize=(10, 5),sharex=True)
joints, xedge, yedge = np.histogram2d(Xdads, Ysons, density=True, bins=20)
#get p(x>71)
px71 = len(Xdads[Xdads>71]) / len(Xdads)

#get conditional by dividing joint / p(x)
py_x71 = np.sum(joints[:,xedge[:-1]>71]/px71,1)
py_x71 /= py_x71.sum()
ax[0].bar(yedge[:-1], py_x71, align='center')
ax[0].set_xlabel('Height of son (in)')
ax[0].set_ylabel('P(y|x>71)')
ax[0].axvline(71,c='r',ls='--', label='Father minimum height')

# Plot the true mean.
true_mean = data[(data.Gender=='M') & (data.Father>71)].Height.mean()
ax[0].axvline(true_mean,c='green',ls=':', label='Son true mean height')
ax[0].legend()


# DAUGHTER PLOTTING
Ygirls = data[data['Gender'] == 'F']['Height']
Xmoms = data[data['Gender'] == 'F']['Mother']
joints, xedge, yedge= np.histogram2d(Xmoms,Ygirls,density=True,bins=20)
#get p(x>66)
px66 = len(Xmoms[Xmoms>66]) / len(Xmoms)

#get conditional by dividing joint / p(x)
py_x66 = np.sum(joints[:,xedge[:-1]>66]/px66,1)
py_x66 /= py_x66.sum()
ax[1].bar(yedge[:-1],py_x66,align='center')
ax[1].set_xlabel('Height of daughter (in)')
ax[1].set_ylabel('P(y|x>66)')
ax[1].axvline(66,c='r',ls='--', label='Mother minimum height')

# Plot the true mean.
true_mean = data[(data.Gender=='F') & (data.Mother>66)].Height.mean()
ax[1].axvline(true_mean,c='green',ls=':',label='Daughter true mean height')
ax[1].legend()
```

Out[18]:
```
<matplotlib.legend.Legend at 0x7fdb4b6c0dc0>
```



1. What do you find?

For tall parents, the center of the distribution of childrens heights seems to be the same or smaller than the minimum cutoff for the parents heights. tall parents seem to have shorter kids.

2. With the same data, can you think of a way to test whether tall parents are causally responsible for their children being less tall?

No, inference of a causal relationship from observational data is not possible. There could be confounding variables that influence parent and/or child height that are captured in the dataset and obscure any causal connection between parent and child height.

3. If there is no causal connection, what does that mean for conditioning on extreme events?

If you condition on an extreme event, the conditional distribution of the other variable would be LESS extreme. This is because extreme events do not necessarily bias the resulting distribution if there is NO causal relationship.

Problem 3: Likelihood vs Prior (1pt)

Leveraging again `scipy.stats`, reproduce the figure from this tweet. To see it here, execute the next cell.

In detail, choose Student's t distribution with 4 degrees of freedom and the standard normal distribution for either the likelihood or the prior, but separate them by $\delta x = 10$. Compute and plot the posteriors.

Which of these prior distributions is more robust to outliers in the data?

1. Which of these prior distributions is more robust to outliers in the data?

If we consider robustness to be the ability to accomodate outliers in the data, the t-distribution is more robust, since the Student-t prior has heavier tails than the normal prior and therefore assigns heavier weights to distant data. When we set the prior to be the normal distribution, and test with a non-overlapping dataset, the posterior only shifts slightly. The opposite behavior is observed with a student-T prior where outliers tend to shift the posterior more easily.

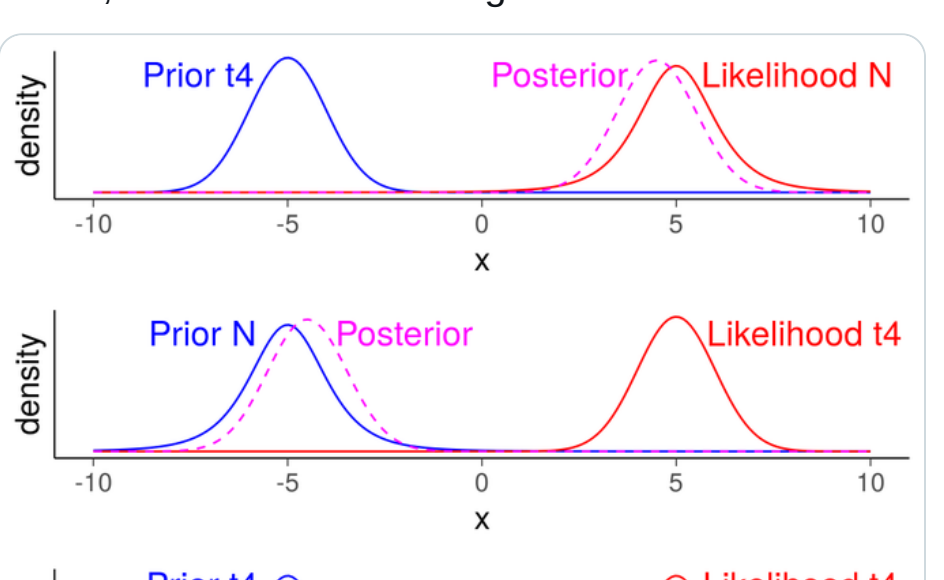
In [7]:
```
import requests

class Tweet(object):
    def __init__(self, s, embed_str=False):
        if not embed_str:
            # Use Twitter's oEmbed API
            # https://dev.twitter.com/web/embedded-tweets
            api = 'https://publish.twitter.com/oembed?url={}'.format(s)
            response = requests.get(api)
            self.text = response.json()["html"]
        else:
            self.text = s

    def _repr_html_(self):
        return self.text

Tweet("https://twitter.com/avehtari/status/1218896617346162688?s=20")
```

Out[7]:

Aki Vehtari (@avehtari@bayes.club)
@avehtari · **Follow**

Here's an illustration of this...

...wait, where did that mule go?



Data Science Fact @DataSciFact
'A Bayesian is one who, vaguely expecting a horse, and catching a glimpse of a donkey, strongly believes he has seen a mule.' ~ @stephensenn

9:02 AM · Jan 19, 2020

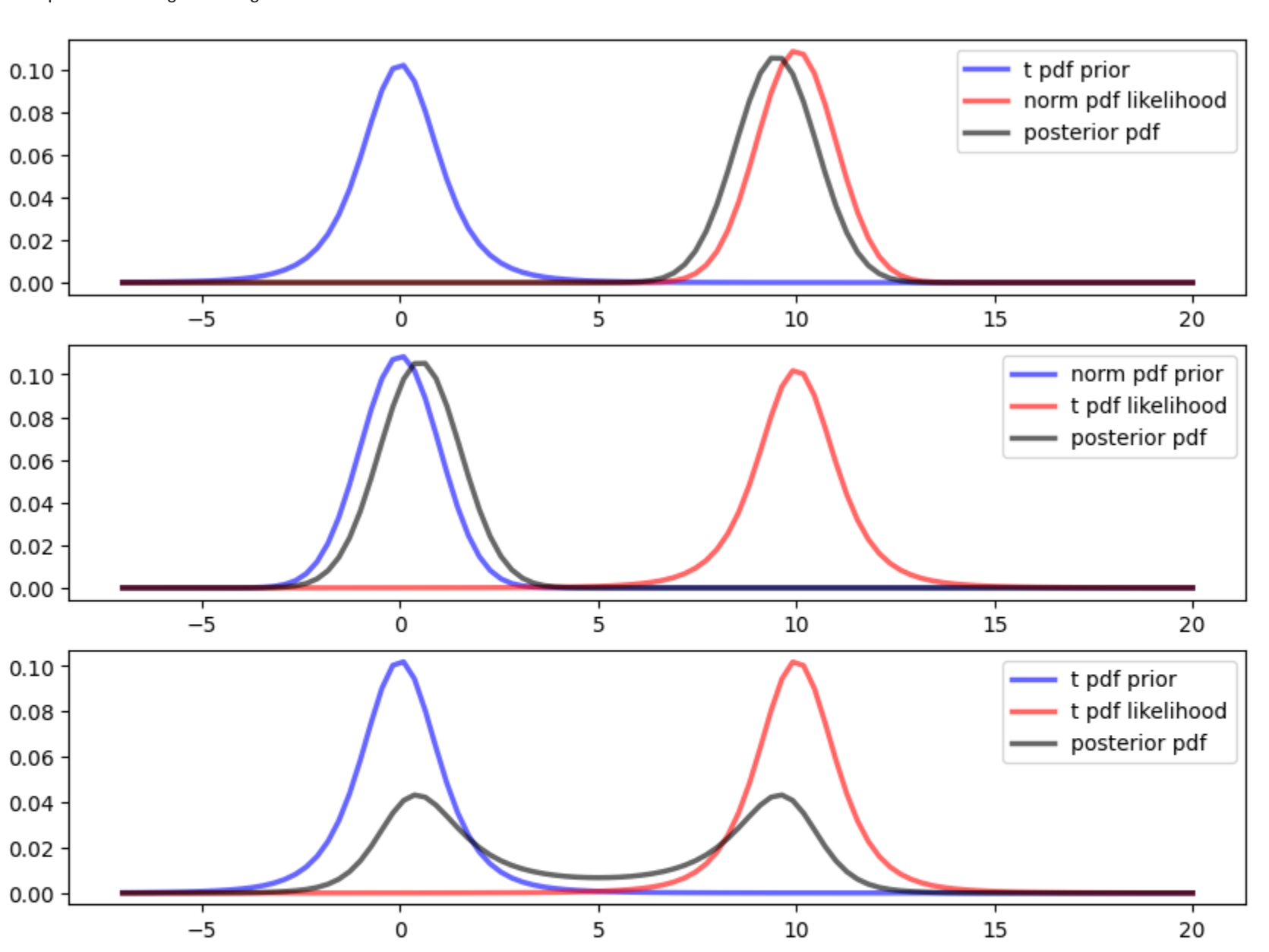
Read the full conversation on Twitter

134 Reply Copy link

Read 5 replies

In [8]:
```
x = np.linspace(-7,20, 100)
fig, ax = plt.subplots(4,1,figsize=(10,10))

prior = stats.t.pdf(x, df=4,loc=0, scale=1) #
prior /= sum(prior) #have to normalize pdf
likelihood = stats.norm.pdf(x,loc=10, scale=1)
likelihood /= sum(likelihood)

ax[0].plot(x, prior,'b-', lw=2.5, alpha=0.6, label='t pdf prior')
ax[0].plot(x, likelihood,'r-', lw=2.5, alpha=0.6, label='norm pdf likelihood')
ax[0].plot(x, np.multiply(prior,likelihood)/np.sum((np.dot(prior,likelihood))), 'k-', lw=2.5, alpha=0.6, label='posterior pdf')

ax[0].legend()
#swap likelihood and prior dists
prior = stats.norm.pdf(x,loc=0, scale=1)
prior /= sum(prior)
likelihood = stats.t.pdf(x,df=4, loc=10, scale=1)
likelihood /= sum(likelihood)
ax[1].plot(x, prior,'b-', lw=2.5, alpha=0.6, label='norm pdf prior')
ax[1].plot(x, likelihood,'r-', lw=2.5, alpha=0.6, label='t pdf likelihood')
ax[1].plot(x, np.multiply(prior,likelihood)/np.sum((np.dot(prior,likelihood))), 'k-', lw=2.5, alpha=0.6, label='posterior pdf')
ax[1].legend()

prior = stats.t.pdf(x,df=4, loc=0, scale=1)
prior /= sum(prior)
likelihood = stats.t.pdf(x,df=4, loc=10, scale=1)
likelihood /= sum(likelihood)
ax[2].plot(x, prior,'b-', lw=2.5, alpha=0.6, label='t pdf prior')
ax[2].plot(x, likelihood,'r-', lw=2.5, alpha=0.6, label='t pdf likelihood')
ax[2].plot(x, np.multiply(prior,likelihood)/np.sum((np.dot(prior,likelihood))), 'k-', lw=2.5, alpha=0.6, label='posterior pdf')
ax[2].legend()

prior = stats.norm.pdf(x,loc=0, scale=1)
prior /= sum(prior)
likelihood = stats.norm.pdf(x,loc=10, scale=1)
likelihood /= sum(likelihood)
ax[3].plot(x, prior,'b-', lw=2.5, alpha=0.6, label='norm pdf prior')
ax[3].plot(x, likelihood,'r-', lw=2.5, alpha=0.6, label='norm pdf likelihood')
ax[3].plot(x, np.multiply(prior,likelihood)/np.sum((np.dot(prior,likelihood))), 'k-', lw=2.5, alpha=0.6, label='posterior pdf')
ax[3].legend()
```

Out[8]:
```
<matplotlib.legend.Legend at 0x7fdb51d71b40>
```



Problem 4: Hubble was no Bayesian (4pts)

...but you can be!

In 1929, Edwin Hubble published a seminal paper, in which he compared the radial velocity of astronomical objects (i.e. how fast these objects move towards or away from us) with their distance. The former can be done pretty precisely with spectroscopy, the latter is much more uncertain.

He saw that the velocity increases with distance and speculated that this could be the sign of a cosmological expansion. This lead cosmologist to believe in the Big Bang theory.

Step 0:

Load the data from the file `hubble.txt` into an array with `numpy.genfromtxt`, and make again use of the arguments `names` and `dtype`. You should get 6 columns

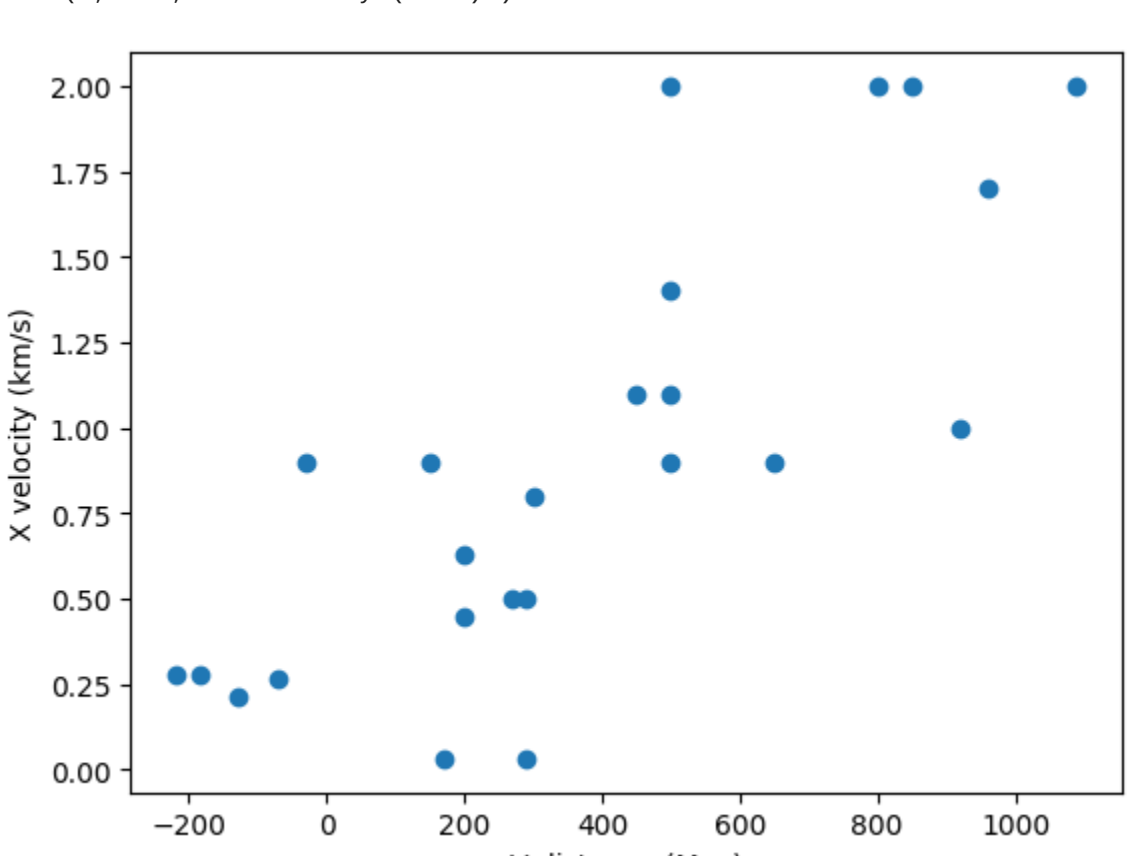
- `CAT`, `NUMBER`: These two combined give you the name of the galaxy.
- `R`: distance in Mpc
- `V`: radial velocity in km/s
- `RA`, `DEC`: equatorial coordinates of the galaxy

Make a scatter plot of R vs V (that means the independent variable is V). Don't forget labels and units...

In [9]:
```
data = np.genfromtxt('hubble_1.txt',dtype=None,names=True)

plt.scatter(data["V"],data["R"])
plt.xlabel("V distance (Mpc)")
plt.ylabel("X velocity (km/s)")
```

```
/tmp/ipykernel_2038/3122344751.py:1: VisibleDeprecationWarning: Reading unicode strings without specifying the encoding argument is deprecated. Set the encoding, use None for the s
ystem default.
  data = np.genfromtxt('hubble_1.txt',dtype=None,names=True)
```

Out[9]:
```
Text(0, 0.5, 'X velocity (km/s)')
```



Step 1 (1pt):

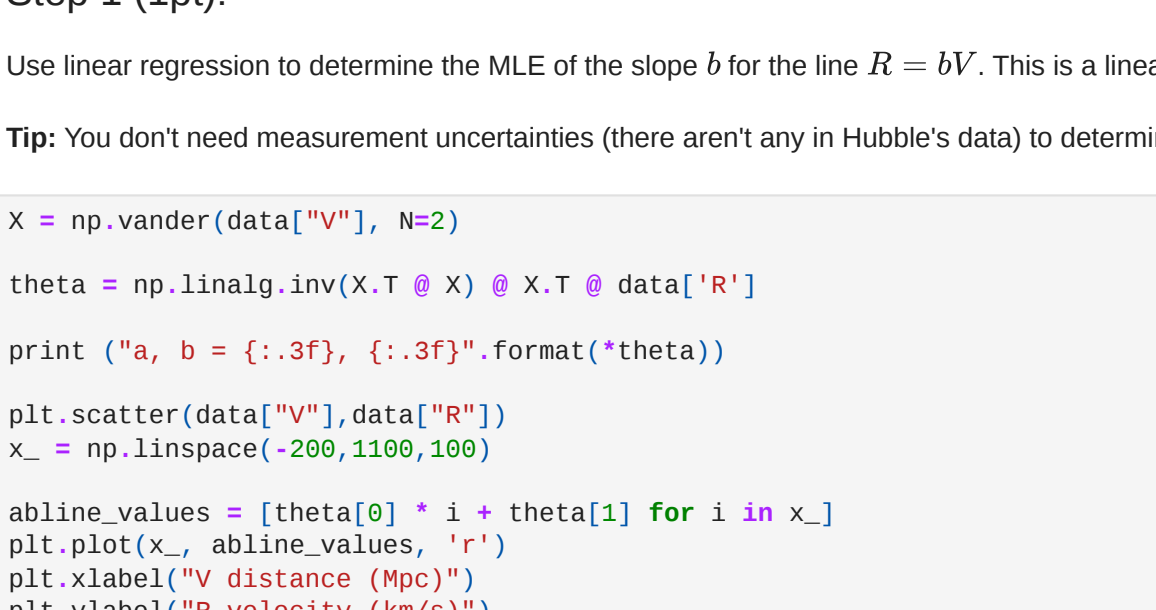
Use linear regression to determine the MLE of the slope b for the line $R = bV$. This is a linear model **with no intercept**. Print the MLE. Then, create a new version of the scatter plot by adding the MLE line.

Tip: You don't need measurement uncertainties (there aren't any in Hubble's data) to determine the MLE.

In [10]:
```
X = np.vander(data["V"], N=2)

theta = np.linalg.inv(X.T @ X) @ X.T @ data['R']

print ("a, b = {:.3f}, {:.3f}".format(*theta))

plt.scatter(data["V"],data["R"])
x_ = np.linspace(-200,1100,100)

abline_values = [theta[0] * i + theta[1] for i in x_]
plt.plot(x_, abline_values, 'r')
plt.xlabel("V distance (Mpc)")
plt.ylabel("R velocity (km/s)")
```

```
a, b = 0.001, 0.399
```

Out[10]:
```
Text(0, 0.5, 'R velocity (km/s)')
```



Step 2 (2pts):

The full Gaussian likelihood of the linear regression problem has a term for the intercept a, slope b, and *uncertainty* σ of R. We will assume that the uncertainties of all data points are identical. Adopt uninformative priors for all of the parameters $\theta = (a, b, \sigma)$.

Compute the log posterior on a reasonably fine grid of (a, b, σ), picking suitable limits for every parameter. Then marginalize out σ and plot the log posterior for the remaining parameters (a, b).

Tip: The function `scipy.special.logsumexp` is useful.

In [11]:
```
from scipy.special import logsumexp
from math import log, pi

A= np.linspace(-5,5,100) #intercept
B= np.linspace(-.01,.01,100) #slope
sigma = np.linspace(0.1,2,100)

X = data['V']
Y = data['R']

lp_mat = np.zeros((A.shape[0],B.shape[0],sigma.shape[0]))

#ok i think because you marginalize out theta in the end it doesn't need to be included as additional constant
def log_post(a,b,sigma,X,Y):
    return -len(X)/2 * log(2*np.pi*sigma**2) - 0.5 * np.sum((Y - (b*X + a))**2/sigma**2)


for ai in range(0,A.shape[0]):
    for bi in range(0,B.shape[0]):
        for si in range(0,sigma.shape[0]):
            lp_mat[ai,bi,si] = log_post(A[ai],B[bi],sigma[si],X,Y)
```

In [12]:
```
out = logsumexp(lp_mat,axis=2)
im = plt.imshow(out)
plt.xlabel('a')
plt.ylabel('b')
plt.xticks(np.linspace(0,99,6), labels=np.linspace(-5,5,6))
plt.yticks(np.linspace(0,99,6), labels=np.linspace(-0.01,0.01,6))
plt.colorbar(im)
```

Out[12]:
```
<matplotlib.colorbar.Colorbar at 0x7fdb5019a350>
```

Step 3 (1pt):

Use the function `sample_2d` below to draw samples from the 2D array of the posterior of (a, b). Create a final version of the scatter plot by adding the lines that correspond to these posterior draws.

Tip: When plotting, set the transparency `alpha` to values < 1, so that multiple draws of the same parameter pair become visually more important.

In [13]:
```
# taken from https://stackoverflow.com/questions/56017163
def sample_2d(dist2d, n=50, replace=True):
    """
    Given an array representing a 2D joint probability distribution p(x,y), return n index pairs (i_x, i_y) sampled
    according to that distribution
    """
    import numpy as np

    shape = dist2d.shape
    N = np.prod(shape)

    # generate the set of all x,y pairs represented by the pmf
    pairs=np.indices(dimensions=shape).T # here are all of the x,y pairs

    # make n random selections from the flattened pmf
    inds = np.random.choice(np.arange(N), p=dist2d.reshape(N), size=n, replace=replace)

    # inds is the set of n randomly chosen indicies into the flattened dist array...
    # therefore the random x,y selections
    # come from selecting the associated elements
    # from the flattened pairs array
    return pairs.reshape(-1,2)[inds]
```

In [14]:
```
#convert log post back to prob and normalize
out = logsumexp(lp_mat,axis=2)
out = out - logsumexp(out)
abs = sample_2d(np.exp(out)) # exponentiate

plt.scatter(data["V"],data["R"])
x_ = np.linspace(-250,1100,1000)
for n, ab in enumerate(abs):
    plt.plot(x_, x_*B[ab[1]]+ A[ab[0]], 'r',ls=':',alpha=0.5)
plt.xlabel('V distance (Mpc)')
plt.ylabel(r'R velocity ($\frac{km}{s})$')
plt.title('')
```

Out[14]:
```
Text(0.5, 1.0, '')
```