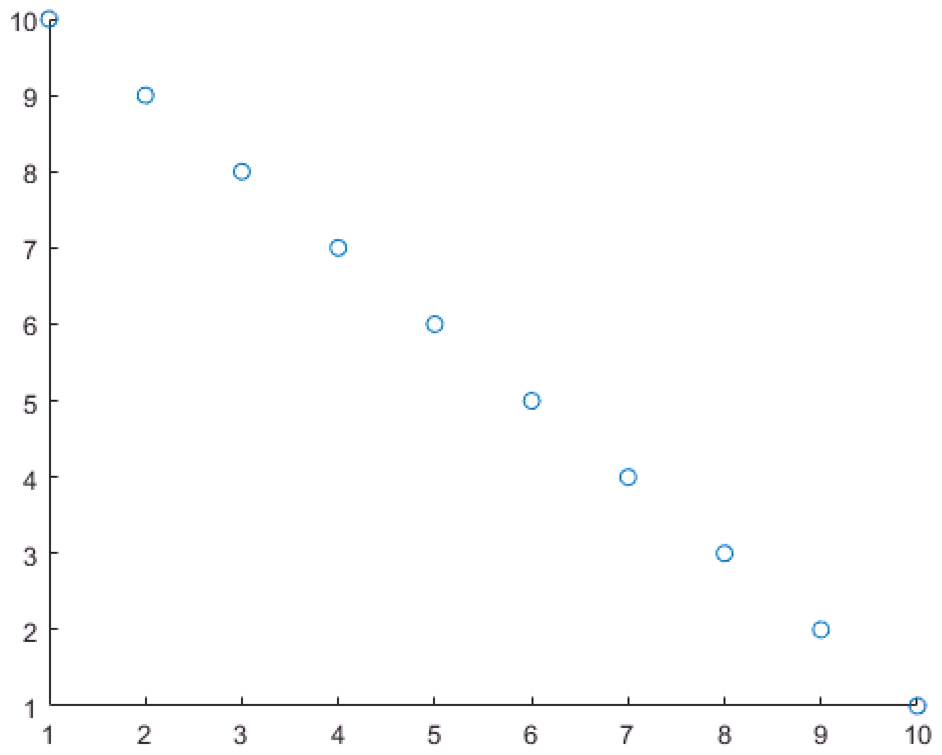
Fitting the Data

We can make the observations from scatter plots & histograms more formal by looking at correlation values and pdf curves.

Correlation

The function `corr(x,y)` gives the pairwise correlation coefficient between the column vectors `x` and `y` (they have to be columns).

```
x= 1:1:10;
y= 10:-1:1;

scatter(x,y)
```



```
[r, p]=corr(x',y')
```

```
r = -1
p = 0
```

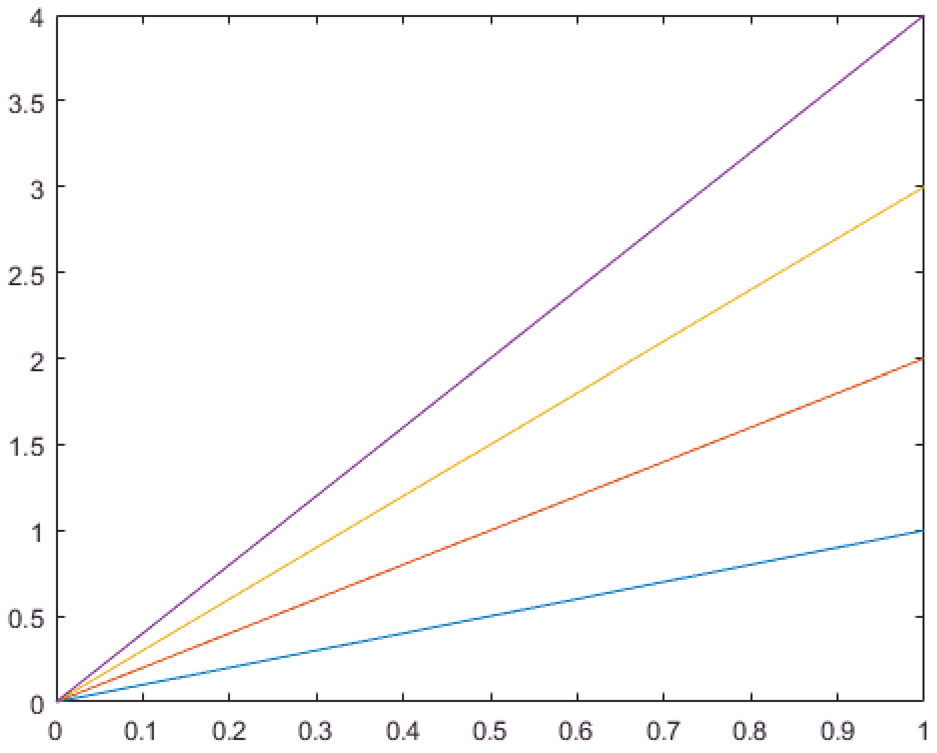
We can also get the coveted p-value by specifying two outputs. DISCLAIMER this sort of test isn't suitable for all data. Consult your friendly neighbourhood statistician if you're not sure.

The `corr` function can also be used to get a matrix of pairwise correlation coefficients of every column in a matrix. Let's create a matrix X with different linear functions in each column.

```
%time
t= 0:0.01:1;
Nloop= 4;

X= zeros(length(t), Nloop);

for idx= 1:Nloop;
    X(:,idx)= idx.*t
end
```

```
X =
         0         0         0         0
    0.0100         0         0         0
    0.0200         0         0         0
    0.0300         0         0         0
    0.0400         0         0         0
    0.0500         0         0         0
    0.0600         0         0         0
    0.0700         0         0         0
    0.0800         0         0         0
    0.0900         0         0         0
      ⋮

X =
         0         0         0         0
    0.0100    0.0200         0         0
    0.0200    0.0400         0         0
    0.0300    0.0600         0         0
    0.0400    0.0800         0         0
    0.0500    0.1000         0         0
    0.0600    0.1200         0         0
    0.0700    0.1400         0         0
    0.0800    0.1600         0         0
    0.0900    0.1800         0         0
      ⋮

X =
         0         0         0         0
    0.0100    0.0200    0.0300         0
    0.0200    0.0400    0.0600         0
    0.0300    0.0600    0.0900         0
    0.0400    0.0800    0.1200         0
    0.0500    0.1000    0.1500         0
    0.0600    0.1200    0.1800         0
    0.0700    0.1400    0.2100         0
    0.0800    0.1600    0.2400         0
    0.0900    0.1800    0.2700         0
      ⋮

X =
         0         0         0         0
    0.0100    0.0200    0.0300    0.0400
    0.0200    0.0400    0.0600    0.0800
    0.0300    0.0600    0.0900    0.1200
    0.0400    0.0800    0.1200    0.1600
    0.0500    0.1000    0.1500    0.2000
    0.0600    0.1200    0.1800    0.2400
    0.0700    0.1400    0.2100    0.2800
    0.0800    0.1600    0.2400    0.3200
    0.0900    0.1800    0.2700    0.3600
      ⋮
```

```
plot(t, X)
```



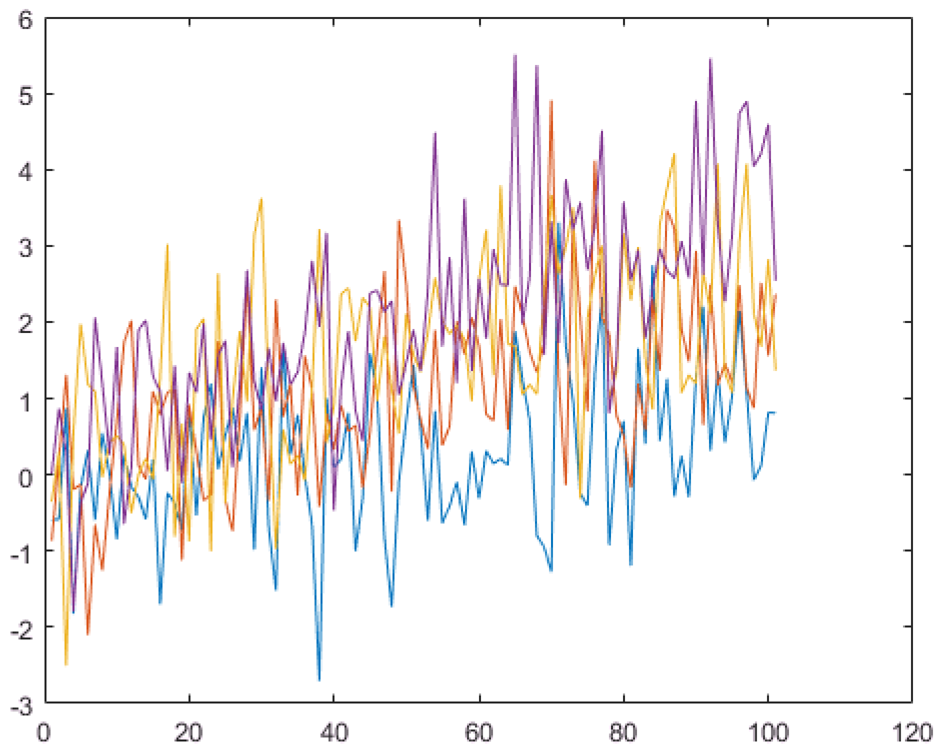
And look at the correlation coefficients

```
[rho, p]= corr(X)
```

```
rho =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
p =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Challenge

```
% CHALLENGE
% What happens to the correlation coefficients
% if you add Gaussian noise
% to the lines in the matrix X?


% HINT: randn(m,n) will create a m-by-n matrix
% of Gaussian noise with mean 0 and std 1.

% Create Gaussian noise the same size as X
g_noise=randn(size(X));

% Add noise to X
New_X=X+g_noise;
% Plot the 4 lines
figure; plot(New_X)
```



```
% Get correlation
[rhop, pp]=corr(New_X)
```

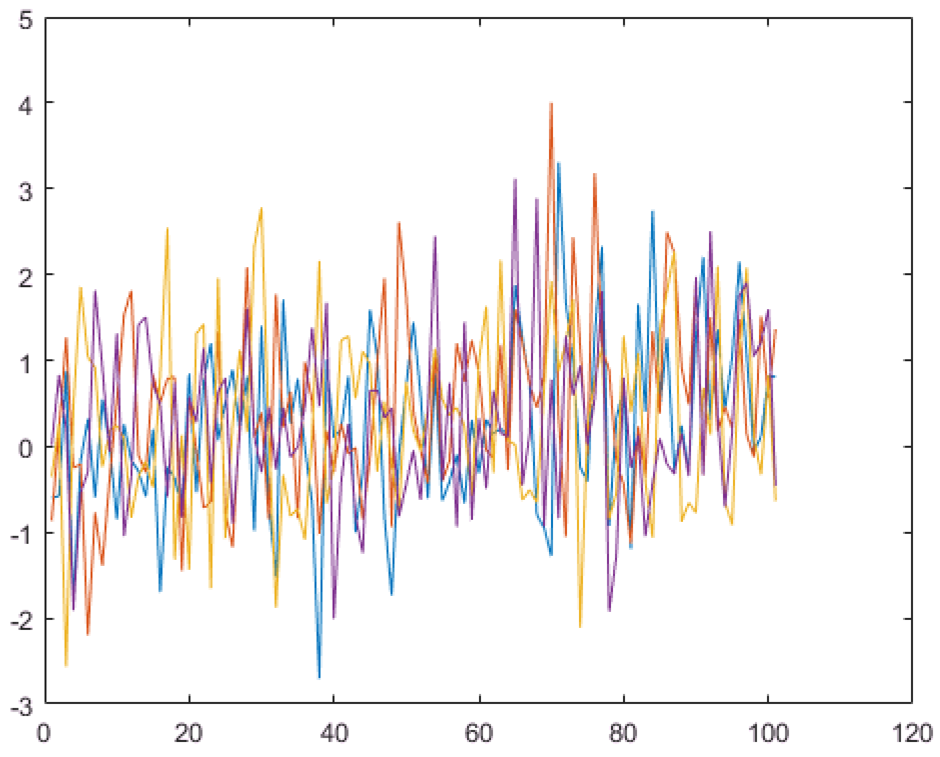
```
rhop =
    1.0000    0.2097    0.1529    0.3154
    0.2097    1.0000    0.2623    0.4486
    0.1529    0.2623    1.0000    0.4081
    0.3154    0.4486    0.4081    1.0000
pp =
    1.0000    0.0353    0.1269    0.0013
    0.0353    1.0000    0.0080    0.0000
    0.1269    0.0080    1.0000    0.0000
    0.0013    0.0000    0.0000    1.0000
```

```
[rho, p]=corr(New_X, 'type', 'Spearman')
```

```
rho =
    1.0000    0.2376    0.1637    0.3069
    0.2376    1.0000    0.2068    0.4357
    0.1637    0.2068    1.0000    0.4303
    0.3069    0.4357    0.4303    1.0000
p =
    1.0000    0.0169    0.1019    0.0019
    0.0169    1.0000    0.0382    0.0000
    0.1019    0.0382    1.0000    0.0000
    0.0019    0.0000    0.0000    1.0000
```

Challenge

```
% CHALLENGE
% What about non-linear data?
% Test the correlation with the function

clear x
for idx= 1:Nloop;
    x (:,idx)= t .^ idx;
end

New_x=x+g_noise;
figure; plot(New_x)
```



```
[rho, p]= corr(New_x)
```

```
rho =
    1.0000    0.1445   -0.0050    0.1368
    0.1445    1.0000   -0.0184    0.1347
   -0.0050   -0.0184    1.0000    0.0342
    0.1368    0.1347    0.0342    1.0000
```

-

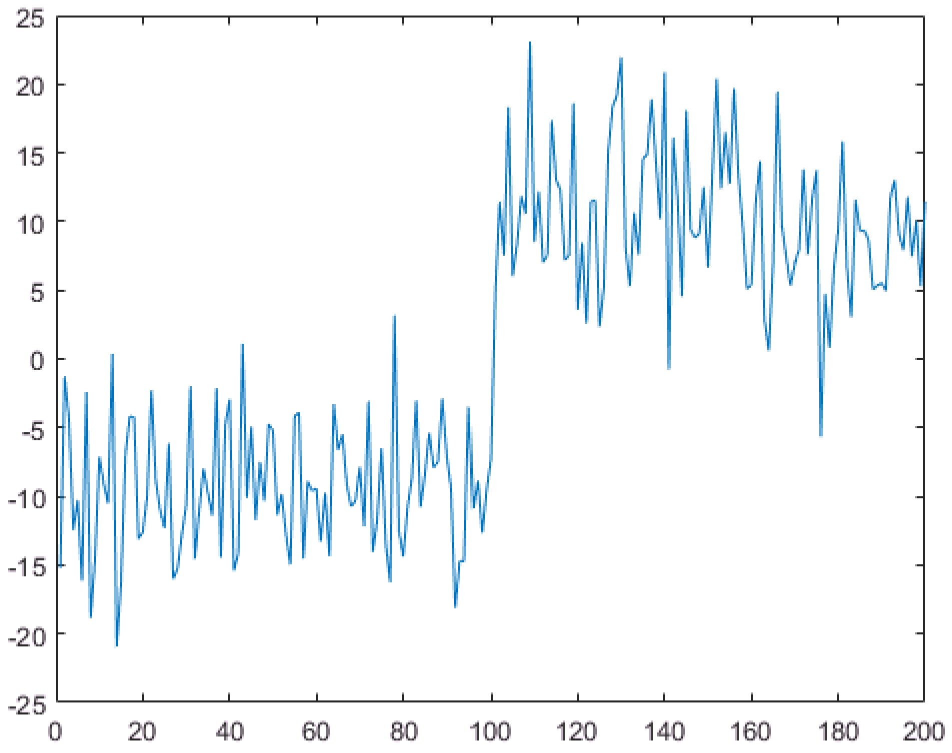
```
p =

    1.0000    0.1494    0.9607    0.1725
    0.1494    1.0000    0.8550    0.1792
    0.9607    0.8550    1.0000    0.7343
    0.1725    0.1792    0.7343    1.0000
```

```
% Try using the rank correlation instead using
% corr(X,'type','Spearman')

% EXTENSION
% What about if you make the functions in X
% sinusoidal instead of linear?

% EXTENSION
% Check the correlation betweeen pedestrian counts at
% Bourke street from your previous code
```

Clustering

The function `kmeans` is a good "quick and dirty" way to segment data into groups. We can make a toy example vector, x with two peaks in the histogram.

```
% make a vector of random numbers with
% half centred around -10 and half around 10.
X= [-10+5*randn(1,100), 10+5*randn(1,100)];
clf; plot(X)
```



```
% use kmeans to find the clusters
idx= kmeans(X',2);
unique(idx);
```

We can plot the data separately using the cluster index

```
clf; plot(X(idx==1), 'b')
hold on
plot(X(idx==2), 'r')
```

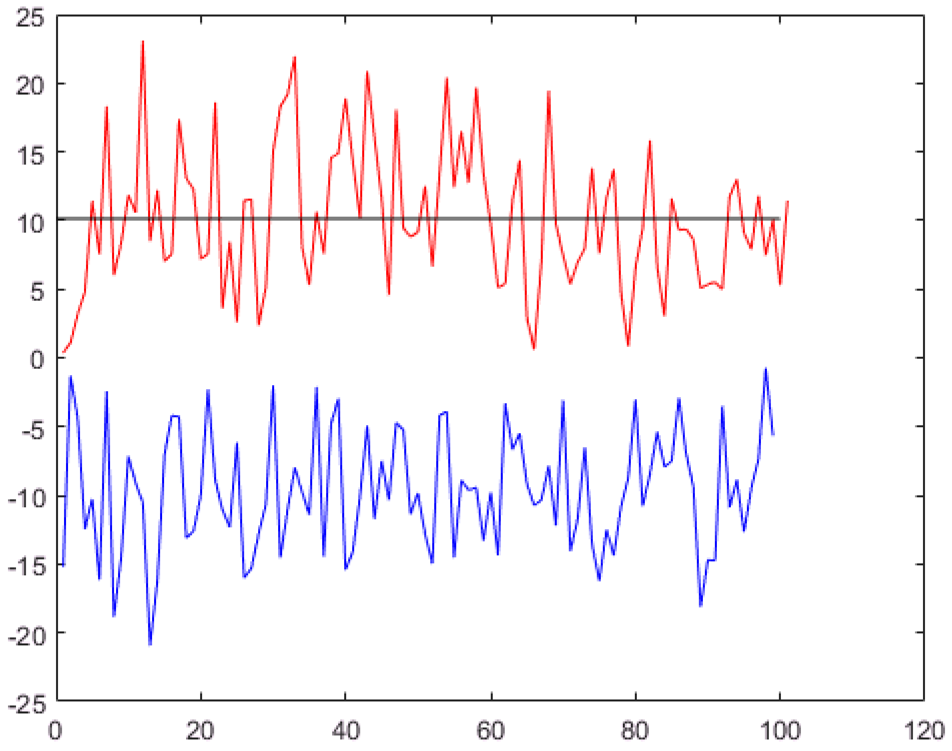
We can also get the centre point of each cluster

```
% a second output includes the centre points
[idx, c]= kmeans(X',2);
c
```

```
c =
    10.1280
    -9.6313
```

```
% we can plot a line through the middle using the
% line function
```

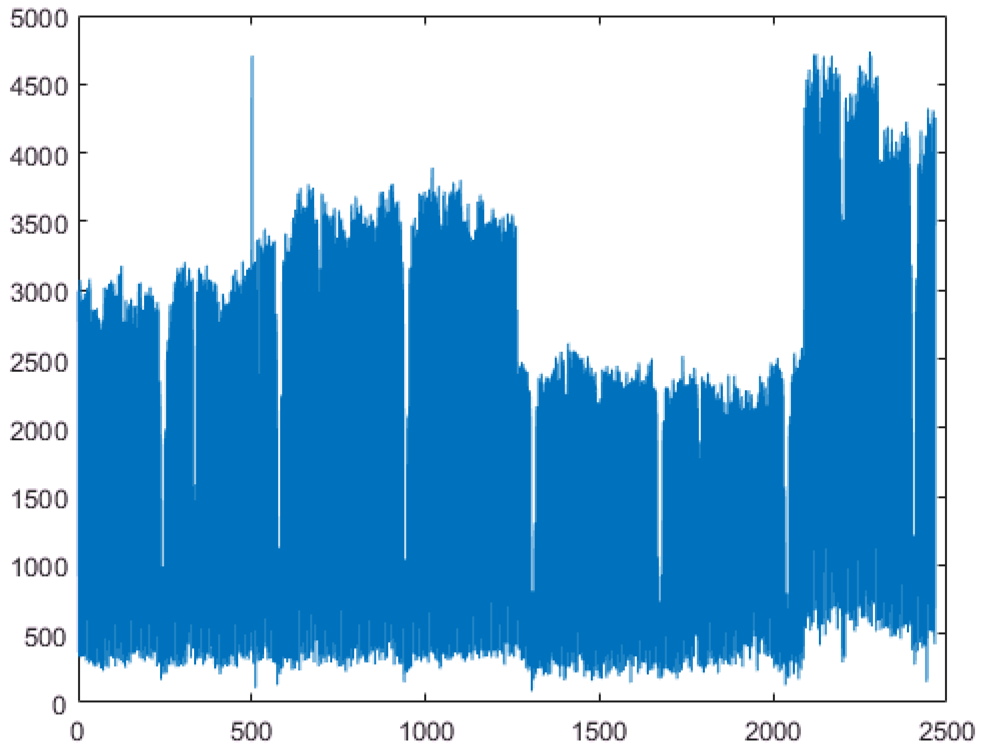
```
line([0 100], [c(1) c(1)], 'color', 'k')
```



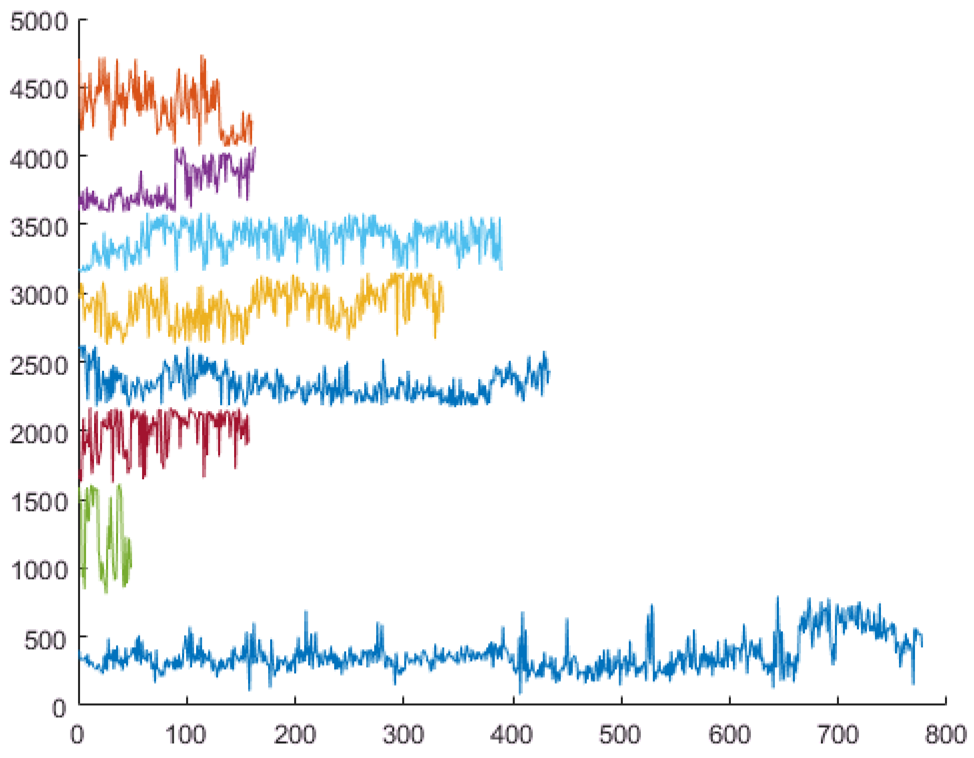
```
line([0 100], [c(2) c(2)], 'color', 'k')
```

Challenge

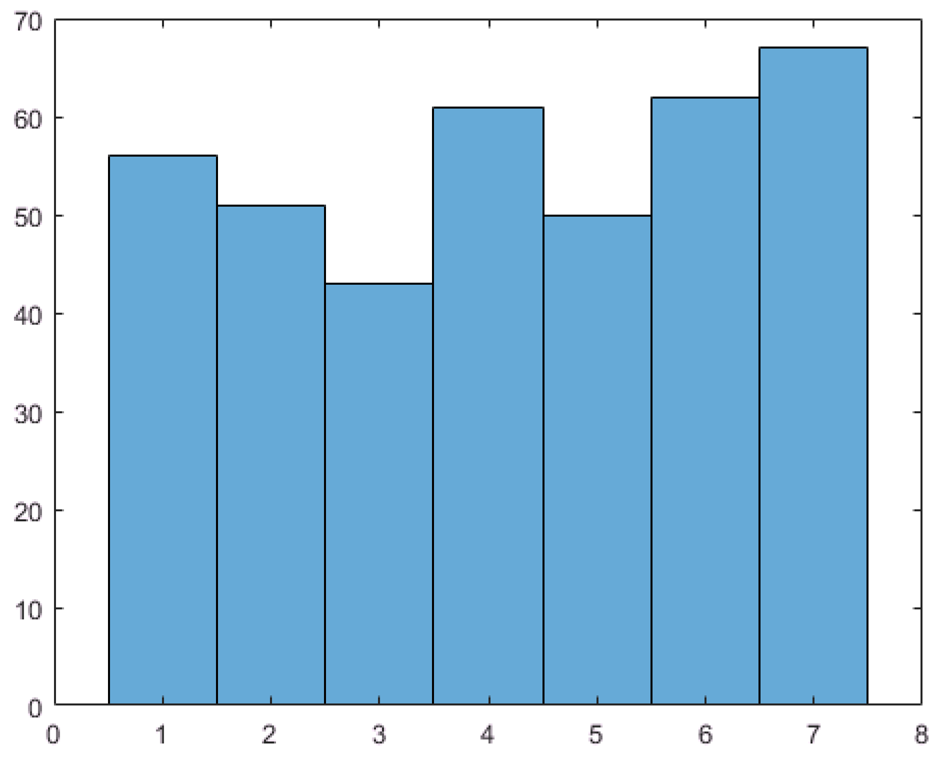
```
%% CHALLENGE
% Look up kmeans in the help documentation and use
% it to cluster the pedestrian
% counts at Flinders street at 8am (Flinders8)
% into two or more groups.
clf; plot(Flinders8)
```



```
rng(1)
[idx, c]= kmeans(Flinders8,8);


figure; hold on
for n=1:8;
plot(Flinders8(idx==n))
end
```



```
% HINT your input should be the pedestrian
% counts from the previous challenge:

%% EXTENSION
% 1. Plot your two clusters as separate histograms
% 2. Include a vertical line on the graph marking the
% center of the cluster (the function line() will help)
%

%% EXTENSION
% Can you see any factors that are different between
% the clusters? Check the day of the week.
dayofweek=weekday(Dates_Flinders);
dayofidx1=dayofweek(idx==6);
figure; histogram(dayofidx1)
```

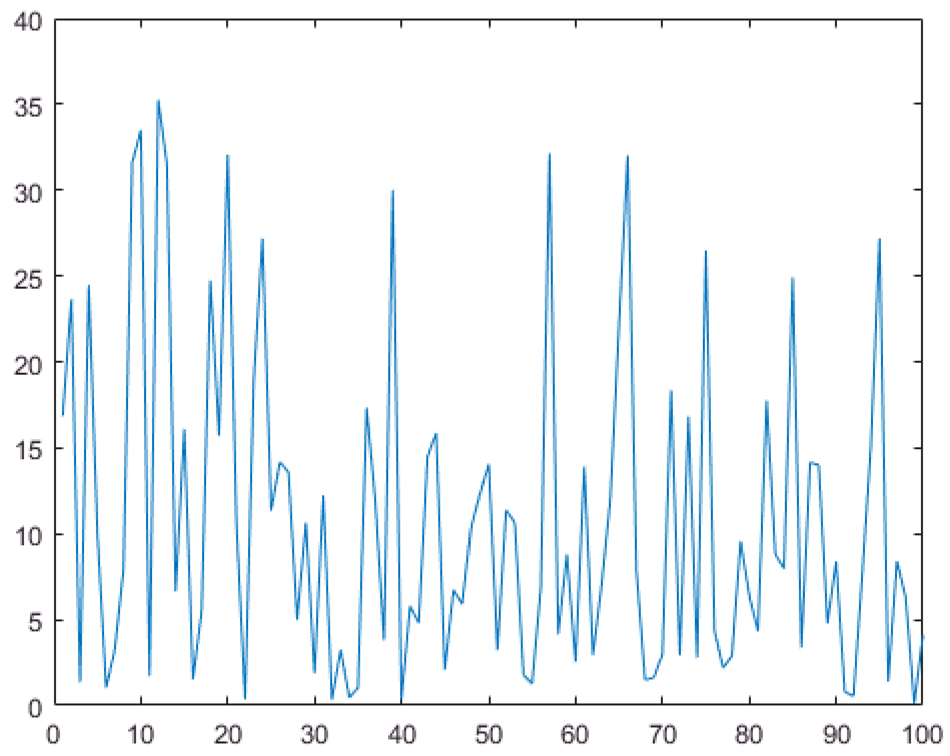


Probability

We can fit the probability density function to our histograms using a kernel-density estimation (`ksdensity`), or we can check the fit of specific distribution shapes, such as Gaussian, exponential, etc (`fitdist`).

Going to make some fake data using the `rand` command

```
rng default
x= -log(1-rand(1,100))/0.1;
figure; plot(x)
```



```
histogram(x, 'normalization', 'pdf'); hold on
```

Fit an exponential distribution

```
pd= fitdist(x', 'exp')
```

```
pd =
  ExponentialDistribution

  Exponential distribution
    mu = 10.7101   [8.88593, 13.1632]
```

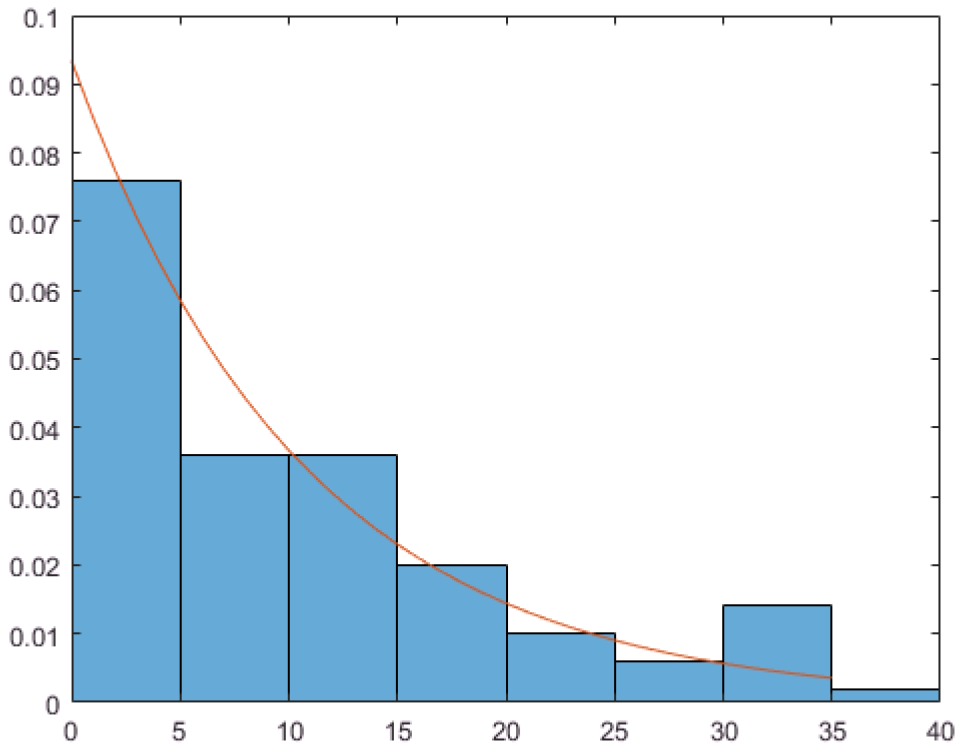
```
pd.mu
```

```
ans = 10.7101
```

```
x_vals= 0:1:max(x);
prob_dens= 1/pd.mu*exp(-1/pd.mu.*x_vals)
```

```
prob_dens =
    0.0934    0.0850    0.0775    0.0706    0.0643    0.0585    0.0533    0.0486 ⋯
```

```
plot(x_vals, prob_dens)
```



We get a distribution parameter and confidence bounds. The exponential distribution just has one parameter. You can always read more about the distribution by clicking the link in MATLAB.

We can check the estimate against the histogram

Challenge

```
% CHALLENGE
% Use fitdist to fit a normal distribution to
% each of the clusters from the previous challenge
% (flinders street on the weekend and weekday)
x_norm= 6+randn(1,5100);

% EXTENSION
% create a line plot of the estimated Gaussian
% distributions on top of the histograms
% HINT: you don't have to look up the equation for a
% Gaussian pdf to plot one, you can use the
% function normpdf()
```

There are heaps of distributions included in the standard `fitdist` function, most of which I have never even heard of... Search the fitdist function.