CA1

November 10, 2022

1 Data Analysis - Coursework 1 (35%)

1.1 Short Style Data Analysis Questions

1.1.1 Deadline Friday week 6, 2pm.

1.2 Instructions

This coursework assesses learning outcomes from **Chapters 1 - 4** of the course.

These assessments are equivalent to an exam: - Submit your work via Turn-It-In on Learning Central. Note that you will need to upload your final notebook exported as a pdf. **Don't forget to execute all of your cells before you export the notebook to pdf**. You can constantly resubmit your turnitin document until the deadline. - The breakdown of the assessment criteria is provided in Learning Central under Assessment. - Don't worry about how your code looks - marks are not given for pretty code, but rather for the approach used in solving the problem, your reasoning, explanation and answer. - Please also take note of the University's policy on **plagiarism**, which is outlined in your student handbook.

1.3 QUESTION 1

[10 marks]

Answer:

In this question we are looking at the probability of passing or failing a number of modules. Since it only has two outcomes, pass/fail, we can treat it as binomial, meaning we can use the following equation,

$$B_{N,\theta}(\nu) = \binom{N}{\nu}\theta^{\nu}(1-\theta)^{(N-\nu)}$$

where N = the total number of trials, ν = the number of successes (modules passed), and θ = the probability of getting a success.

We are told that $\theta = 0.8f$, this would make the binomial equation,

$$B_{N,0.8f}(\nu) = \binom{N}{\nu}(0.8f)^{\nu}(1-0.8f)^{(N-\nu)}$$

where f is the fraction of weekly live sessions that a student attends.

1.3.1 a)

A student attending all the session swill have $f = 1$ resulting in $\theta = 0.8$.

$$\Rightarrow B_{N,0.8}(\nu) = \binom{3}{\nu}(0.8)^{\nu}(1-0.8)^{(3-\nu)}$$

(i) Passing all three modules would mean $\nu = 3$.

$$\Rightarrow B_{N,0.8}(3) = \binom{3}{3}(0.8)^{3}(1-0.8)^{(3-3)}$$

(ii) Failing one module and passing two modules would mean $\nu = 2$.

$$\Rightarrow B_{N,0.8}(2) = \binom{3}{2}(0.8)^{2}(1-0.8)^{(3-2)}$$

(iii) Failing two modules and passing one module would mean $\nu = 1$.

$$\Rightarrow B_{N,0.8}(1) = \binom{3}{1}(0.8)^{1}(1-0.8)^{(3-1)}$$

(iv) Passing no modules would mean $\nu = 0$.

$$\Rightarrow B_{N,0.8}(0) = \binom{3}{0}(0.8)^{0}(1-0.8)^{(3-0)}$$

[137]:
```
from scipy.stats import binom
import numpy as np

f = 1
N = 3
theta = 0.8*f

nu = 3
p_passing_three = binom.pmf(nu, N, theta)
```

```
print('(i) The probability of passing all three modules is {:.2f}'.
 ↪format(p_passing_three))

nu = 2
p_passing_two = binom.pmf(nu, N, theta)
print('(ii) The probability of passing two and failing one modules is {:.2f}'.
 ↪format(p_passing_two))

nu = 1
p_passing_one = binom.pmf(nu, N, theta)
print('(iii) The probability of passing one and failing two modules is {:.2f}'.
 ↪format(p_passing_one))

nu = 0
p_passing_no_modules = binom.pmf(nu, N, theta)
print('(iv) The probability of failing all three modules is {:.2f}'.
 ↪format(p_passing_no_modules))
```

```
(i) The probability of passing all three modules is 0.51
(ii) The probability of passing two and failing one modules is 0.38
(iii) The probability of passing one and failing two modules is 0.10
(iv) The probability of failing all three modules is 0.01
```

1.3.2 b)

[138]:
```
total_of_probs = p_passing_three + p_passing_two + p_passing_one +␣
 ↪p_passing_no_modules

print('The sum of all the probabilities is {:.2f}'.format(total_of_probs))
```

```
The sum of all the probabilities is 1.00
```

1.3.3 c)

A student attending half of the weekly sessions will have $f = 0.5$, resulting in $\theta = 0.4$,

$$\Rightarrow B_{N,0.4}(\nu) = \binom{3}{\nu}(0.4)^{\nu}(1-0.4)^{(3-\nu)}$$

(i) Passing all three modules would mean $\nu = 3$.

$$\Rightarrow B_{N,0.4}(3) = \binom{3}{3}(0.4)^{3}(1-0.4)^{(3-3)}$$

(ii) Failing one module and passing two modules would mean $\nu = 2$.

$$\Rightarrow B_{N,0.4}(2) = \binom{3}{2}(0.4)^2(1-0.4)^{(3-2)}$$

(iii) Failing two modules and passing one module would mean $\nu = 1$.

$$\Rightarrow B_{N,0.4}(1) = \binom{3}{1}(0.4)^1(1-0.4)^{(3-1)}$$

(iv) Passing no modules would mean $\nu = 0$.

$$\Rightarrow B_{N,0.4}(0) = \binom{3}{0}(0.4)^0(1-0.4)^{(3-0)}$$

[139]:
```
f = 0.5
N = 3
theta = 0.8*f

nu = 3
p_passing_three = binom.pmf(nu, N, theta)

print('The probability of passing all three modules is {:.2f}'.
 ↪format(p_passing_three))

nu = 2
p_passing_two = binom.pmf(nu, N, theta)

print('The probability of passing two and failing one modules is {:.2f}'.
 ↪format(p_passing_two))

nu = 1
p_passing_one = binom.pmf(nu, N, theta)

print('The probability of passing one and failing two modules is {:.2f}'.
 ↪format(p_passing_one))

nu = 0
p_passing_no_modules = binom.pmf(nu, N, theta)

print('The probability of failing all three modules is {:.2f}'.
 ↪format(p_passing_no_modules))
```

```
The probability of passing all three modules is 0.06
The probability of passing two and failing one modules is 0.29
The probability of passing one and failing two modules is 0.43
The probability of failing all three modules is 0.22
```

1.3.4 d)

In order to have a 50% chance of passing all three modules ($\nu = 3$), we know that,

$$B_{N,0.8f}(3) = \binom{3}{3}(0.8f)^3(1-0.8f)^{(3-3)} = 0.50$$

$$\Rightarrow B_{N,0.8f}(3) = (1)(0.8f)^3(1-0.8f)^{(0)} = 0.50$$

$$\Rightarrow B_{N,0.8f}(3) = (0.8f)^3(1) = 0.50$$

$$\Rightarrow (0.8f)^3 = 0.50$$

this can then be rearange for f,

$$f = \frac{\sqrt[3]{0.50}}{0.8}$$

[140]:
```
p = 0.5
f = np.cbrt(p) / 0.8

print('In order to have a 50% chance of passing all three, f needs to be {:.
↪2f}'.format(f))
```

```
In order to have a 50% chance of passing all three, f needs to be 0.99
```

1.3.5 e)

It isnt a very realistic way of modeling the probability that the studetns will pass the modules because this assumes that passing or failing each module is independent from each other, when in reality they most likely arent completely independent.
There are also more factors that go into passing a module than the attendance of the student.
The f value also doesnt allow for nuance in attendence, a student could fully attend one class and only attend the other classes half of the time, this would give give a reduced likelyhood of passing the 100% attendance class compared to if you just apply the model to passing the 100% attendance class.

1.4 QUESTION 2

[20 marks]

Answer:

H_0: M23 and D3 are equally good markers for the disease.

H_a: M23 is a better marker for the disease than D3.

At a 95% confidence interval there is a significant level of $\alpha = 0.05$.

Since the test is has two possible results, they either test positive or they test negative, then it can be treated as a binomial,

$$B_{N,\theta}(\nu) = \binom{N}{\nu}\theta^{\nu}(1-\theta)^{(N-\nu)}$$

where N = the total number of trials, ν = the number of successes (number that test possitive), and θ = the probability of getting a success.
From the question we know that $N = 7$, $\nu = 7$, and $\theta = 0.65$

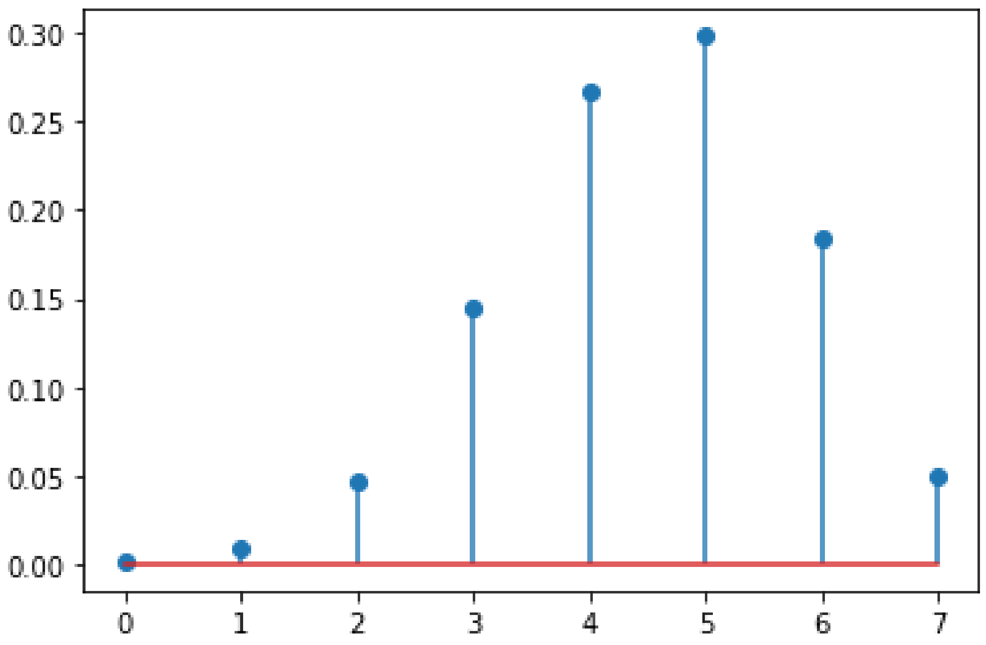
[141]:
```
from scipy.stats import binom
import scipy.stats as st
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


theta = 0.65 #prob
N = 7   # No of trials
nu = 7
x = np.arange(0,nu+1,1)

ans = binom.pmf(x, N, theta)
print('The probability of 7 successes in 7 trials = {:.3f}'.format(ans[N]))
```

```
The probability of 7 successes in 7 trials = 0.049
```

$0.049 < 0.05$, therefore we can reject the null hypothesis that M23 and D3 are equally good markers for the disease.

The data is discrete so it is best to do a stem plot to graphically visualise the data.

[142]:
```
_ = plt.stem(x,ans, use_line_collection = True)
```



While from the p-value we can reject the null hypothesis, the p-value is very close to α. There is also an issue with the sample size of 7 for the D3 marker. This is very small and a much larger sample size should be tested before any conclusions can be made.
There also isnt much information about the D3 marker. We arent told how often it appears in the regular population. It could be possible that everyone has the D3 marker, so then anyone with the disease would of course have the marker.

1.5 QUESTION 3

[15 marks]

Answer:

The problem is essentially pass/fail for the CPUs and as a result we can treat it as a binomial distribution. Since N will be large we can assume that it will tend to a normal distribution.
Given $\theta = 0.50$, $SE = 0.025$, Find N.
Assume a 95% Confidence Interval with a standard deviation of 1.96σ.

$$SE = \frac{\sigma}{\sqrt{N}}$$

$$\sigma = \sqrt{N\theta\left(1-\theta\right)}$$

$$\hat{\nu} = N\theta$$

The equation for σ and $\hat{\nu}$ can be substituted into the equation for SE.

$$\Rightarrow SE = \frac{\sqrt{\hat{\nu}\,(1-\theta)}}{\sqrt{N}}$$

This can be rearanged to find N, and since N has to be a whole number of trails then we use the ceiling function.

$$\Rightarrow N = \left\lceil \frac{\hat{\nu}\,(1-\theta)}{SE^2} \right\rceil$$

If we give a defective CPU the value of 0 and a working CPU a value of 1, then we can assume that, given $\theta = 0.50$, the mean $\hat{\nu} = 0.5$.

$$\therefore SE = \left\lceil \frac{1.96 \times \sqrt{0.5\,(1-0.5)}}{\sqrt{N}} \right\rceil$$

$$\Rightarrow N = \left\lceil \frac{1.96^2 \times 0.25}{0.025^2} \right\rceil = 1537$$

[143]:
```python
import numpy as np

theta = 0.5
error = 0.025
std = 1.96*np.sqrt(theta*(1-theta)) #95% CI sqrt(Nt(1-t))
n = np.ceil((std/error)**2)

print('The minimum number of CPUs that need to be tested is {:.0f}'.format(n))
```

```
The minimum number of CPUs that need to be tested is 1537
```

1.6 QUESTION 4

[25 marks]

Answer:

The question is asking what is the porbability of someone being a democrat given that they are against the change to the law.
Using the following definitions,
$D = Democrat$,

$R = Republican$,
$I = Independent$,
$A = Against$,
$A^c = For(NotAgainst)$,
this means we are looking for $P(D|A)$.

The question gives us the probabilities of being for the change in the law given each party,

$$P(A^c|D) = 0.80$$
$$P(A^c|R) = 0.35$$
$$P(A^c|I) = 0.10$$

We are also given the probabilities of voting for each party,

$$P(D) = 0.40$$
$$P(R) = 0.36$$

We can calulate $P(I)$ from,

$$P(I) = 1 - P(D) - P(R)$$

We can also calculte the probability of being against the change in the law given each party as follows,

$$P(A|D) = 1 - P(A^c|D)$$
$$P(A|R) = 1 - P(A^c|R)$$
$$P(A|I) = 1 - P(A^c|I)$$

In order to calculte $P(D|A)$ we can use bayes theorem which is defined as,

$$P(X|Y) = \frac{P(Y|X) * P(X)}{P(Y)}$$

From this, the probability of being a democrat given you are against the change in teh law would be,

$$P(D|A) = \frac{P(A|D) * P(D)}{P(A)}$$

We already know $P(A|D)$ and $P(D)$, so we only need to calculate $P(A)$. This can be calculated using,

$$P(A) = (A \cap D) + (A \cap R) + (A \cap I)$$

To find the $P(X|Y)$ we can rearange the following equation,

$$P(X|Y) = \frac{P(X \cap Y)}{P(Y)}$$

$$\Rightarrow P(X \cap Y) = P(X|Y) * P(Y)$$

[144]:
```
p_for_given_rep = 0.35
p_for_given_ind = 0.1
p_for_given_dem = 0.8

p_dem = 0.4
p_rep = 0.36
p_ind = 1 - p_dem - p_rep

p_against_given_dem = 1 - p_for_given_dem
p_against_given_rep = 1 - p_for_given_rep
p_against_given_ind = 1 - p_for_given_ind

p_against_and_dem = p_against_given_dem * p_dem
p_against_and_rep = p_against_given_rep * p_rep
p_against_and_ind = p_against_given_ind * p_ind

p_against = p_against_and_dem + p_against_and_ind + p_against_and_rep


p_dem_given_against = (p_against_given_dem*p_dem) / p_against

print('P(D|A) = {:.2f}'.format(p_dem_given_against))
```

```
P(D|A) = 0.15
```

1.7 QUESTION 5

[30 marks]

Answer:

1.7.1 Data Visualisation

To help decide what tests shouls and shouldnt be applied to the data, it is best to visualise the data first to give an idea of what the data is like. This can be done through a combination of the python libraries `pandas`, `matplotlib`, and `numpy`.

[145]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

person = np.arange(1, 11, 1)
hundred = [11, 12, 12, 13, 13, 15, 11, 16, 11, 12]
fifteenhundred = [270, 300, 230, 260, 270, 230, 260, 240, 270, 260]

dict = {   'Person': person,
           '100m': hundred,
           '1500m': fifteenhundred }

df = pd.DataFrame(dict)
df
```

[145]:

```
   Person  100m  1500m
0       1    11    270
1       2    12    300
2       3    12    230
3       4    13    260
4       5    13    270
5       6    15    230
6       7    11    260
7       8    16    240
8       9    11    270
9      10    12    260
```

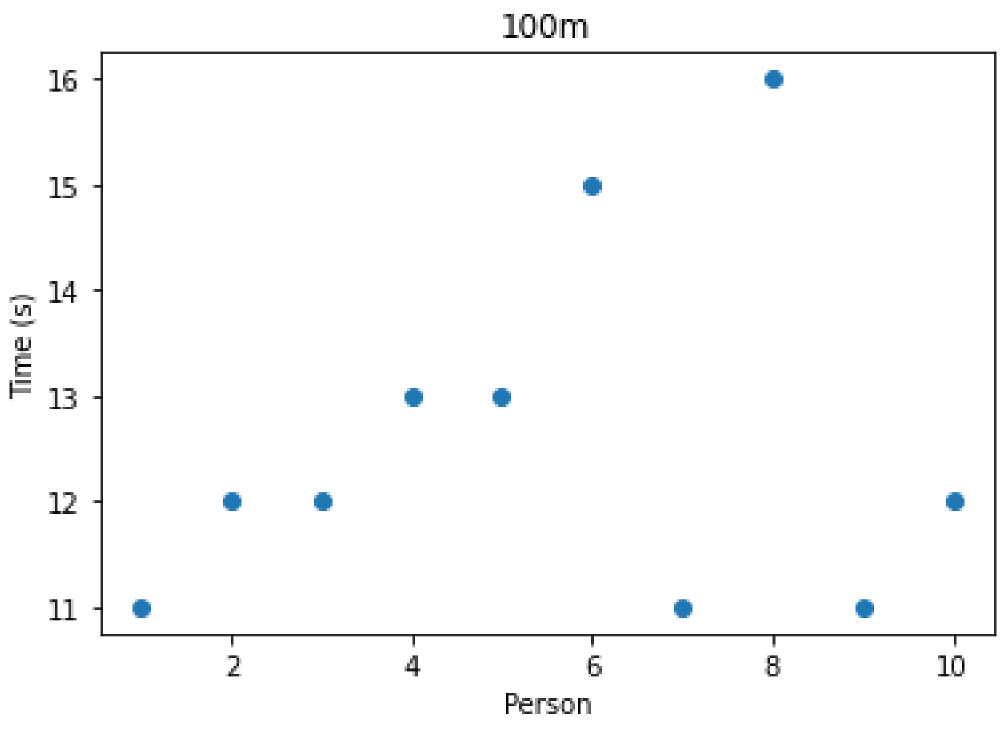
[146]:

```
print(df[['100m', '1500m']].describe())
```

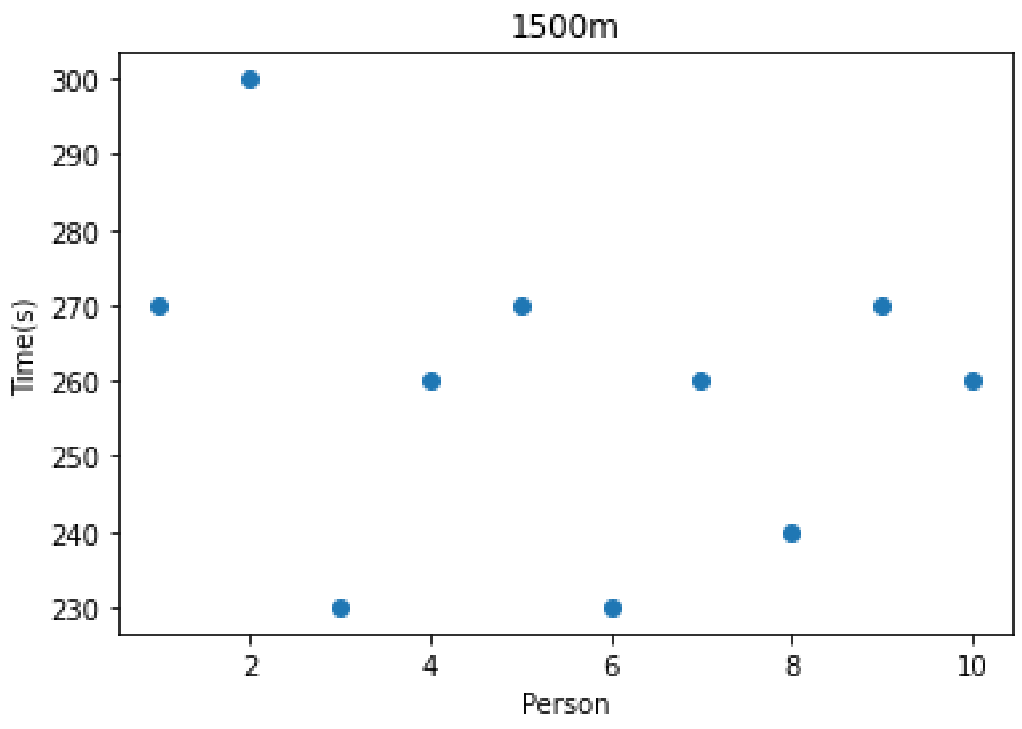
```
            100m       1500m
count  10.000000   10.000000
mean   12.600000  259.000000
std     1.712698   21.317703
min    11.000000  230.000000
25%    11.250000  245.000000
50%    12.000000  260.000000
75%    13.000000  270.000000
max    16.000000  300.000000
```

[147]:

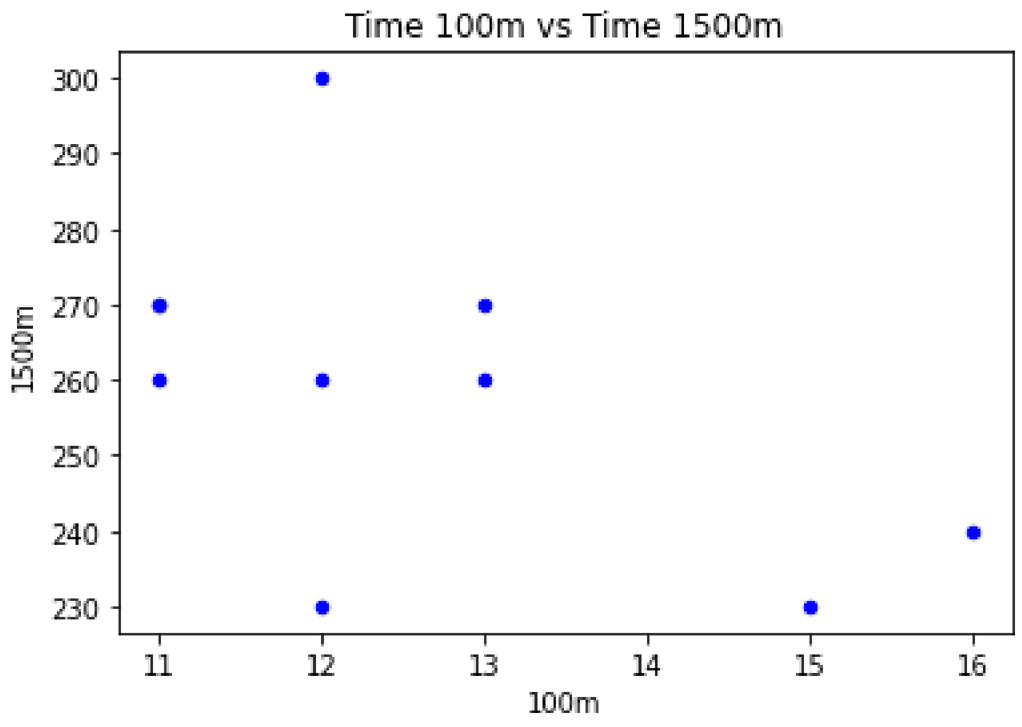
```
plt.title('100m')
plt.xlabel('Person')
plt.ylabel('Time (s)')
plt.scatter(df['Person'], df['100m'])
plt.show()
```



[148]:
```
plt.title('1500m')
plt.xlabel('Person')
plt.ylabel('Time(s)')
plt.scatter(df['Person'], df['1500m'])
plt.show()
```



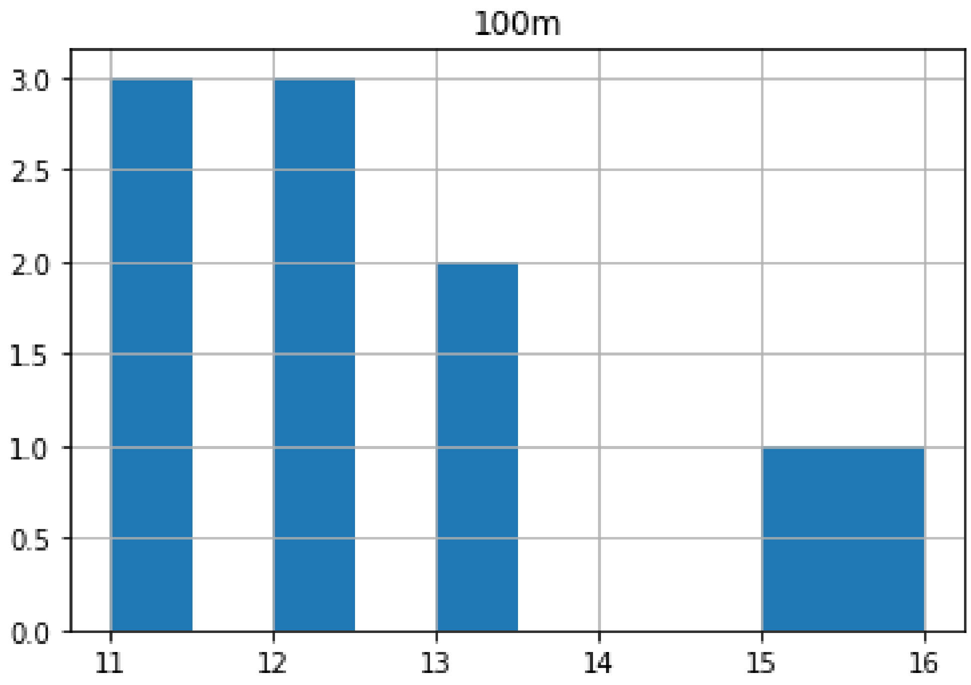
[149]:
```
_ = df.plot(kind='scatter', x='100m', y='1500m', color = 'blue', title = 'Time␣
↪100m vs Time 1500m')
```



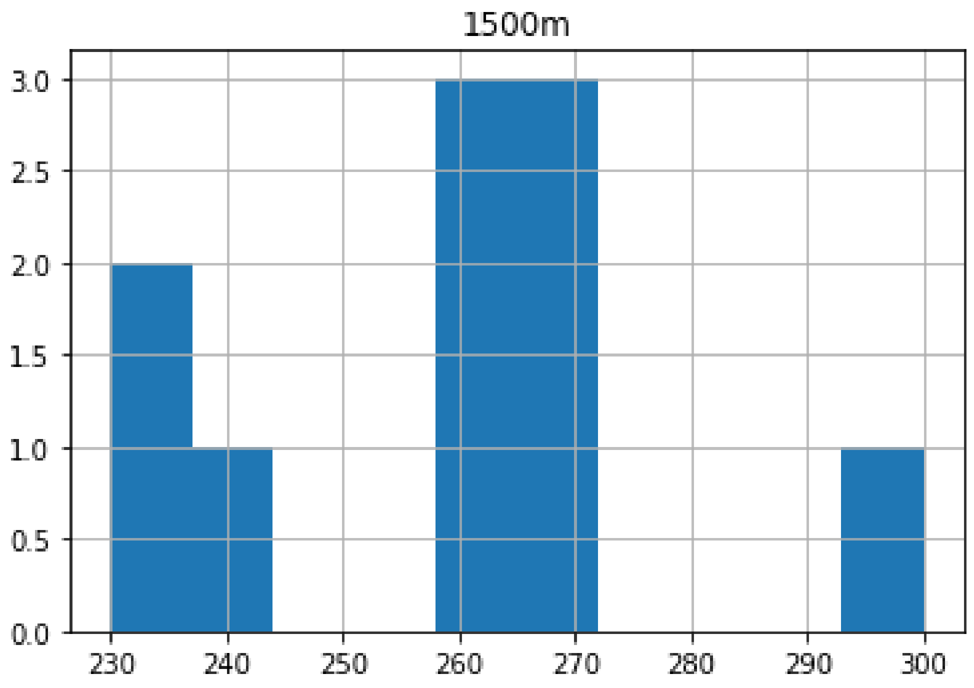
It is likely that the data is non-monotonic based on a visual analysis of the plotted data. The sample size being somewhat small so it is difficult to make a final conclusion on whether the data is monotonic or non-monotonic. The data could be monotonic with a few outliers.

The data also doesnt appear to be normally distributed, this ban be confirmed by ploting the data into histograms.

[150]:
```
_ = df.hist(column='100m')
```



[151]:
```
_ = df.hist(column='1500m')
```



The data doesnt appear to be normally distributed.

1.7.2 Tests

Since the data is not Gausian, Pearsonr cannot be used, and this along with the data being non-catagorical, it also means it isn't appropriate to do a chi-squared test. Therefore, alternatively, we can try and use the Spearman's rank test.

The equation for spearmans rank rest is as follows,

$$\rho_s = \frac{\sum_{i=1}^{N} R(x_i)\,R(y_i) - N(N+1)^2/4}{\sqrt{\sum_{i=1}^{N} R(x_i)^2 - N(N+1)^2/4}\sqrt{\sum_{i=1}^{N} R(y_i)^2 - N(N+1)^2/4}}$$

However, in the case that there are no duplicates in the data, the above expression reduces to

$$\rho_s = 1 - \frac{\sum_{i=1}^{N} [R(x_i) - R(y_i)]^2}{N\,(N^2-1)}$$

To see if the reduced equation can be used, it is possible to check both data sets for duplicates.

```
[152]: df.duplicated(['100m'])
```

```
[152]: 0    False
       1    False
       2     True
```

```
3    False
4     True
5    False
6     True
7    False
8     True
9     True
dtype: bool
```

[153]:
```
df.duplicated(['1500m'])
```

[153]:
```
0    False
1    False
2    False
3    False
4     True
5     True
6     True
7    False
8     True
9     True
dtype: bool
```

Both sets of data have duplicates and therefore it is not possible to use the reduced equation.

It is important to outline the null hypothesis and the alternative hypothesis when hypothesis testing. In the case of Spearman's rank,

H_0: There is no correlation between the data sets.

H_a: There is a correlation between the data sets.

At a 95% confidence interval there is a significant level of $\alpha = 0.05$.

[154]:
```
from scipy.stats import rankdata

rank_100m = rankdata(df['100m'])
rank_1500m = rankdata(df['1500m'])
N = len(rank_100m)
N_term = (N/4)*(N+1)**2


r_top = np.sum(rank_1500m * rank_100m) - N_term
r_bl = np.sum((rank_1500m)**2) - N_term
r_br = np.sum((rank_100m)**2) - N_term

r = r_top / (np.sqrt(r_bl)*np.sqrt(r_br))
print('Spearman\'s rho = {:.2f}'.format(r))
```

```
Spearman's rho = -0.47
```

This can be compared to the critical value of 0.564 which can be found in the Spearmans ρ table, for $n = 10$ and a confidance interval of 95% such that $\alpha = 0.05$.

$|-0.474| < 0.564$, therefore we cant reject the null hypothesis that there is no correlation between the two data sets.

A test can also be done to test each data set for outliers in the data, this can potentially help in confirming our earlier assumption that the data is non-monotonic. If there are little to no outliers then we can assume that the data is non-monotonic based on our previous visual interpretation.

[155]:
```
# https://github.com/aprilypchen/depy2016/blob/master/DePy_Talk.ipynb
# This function finds the outliers that are outside of 1.5IQR.
# It returns the value of the outlier, and where it is in the dataset
def find_outliers_tukey(x):
    q1 = np.percentile(x, 25)
    q3 = np.percentile(x, 75)
    iqr = q3-q1
    floor = q1 - 1.5*iqr
    ceiling = q3 + 1.5*iqr
    outlier_indices = list(x.index[(x < floor)|(x > ceiling)])
    outlier_values = list(x[outlier_indices])

    return outlier_indices, outlier_values

out_ind, out_val = find_outliers_tukey(df['100m'])
print('Number of outliers in the 100m data set =',len(np.sort(out_val)))

out_ind, out_val = find_outliers_tukey(df['1500m'])
print('Number of outliers in the 1500m data set =',len(np.sort(out_val)))
```

```
Number of outliers in the 100m data set = 1
Number of outliers in the 1500m data set = 0
```

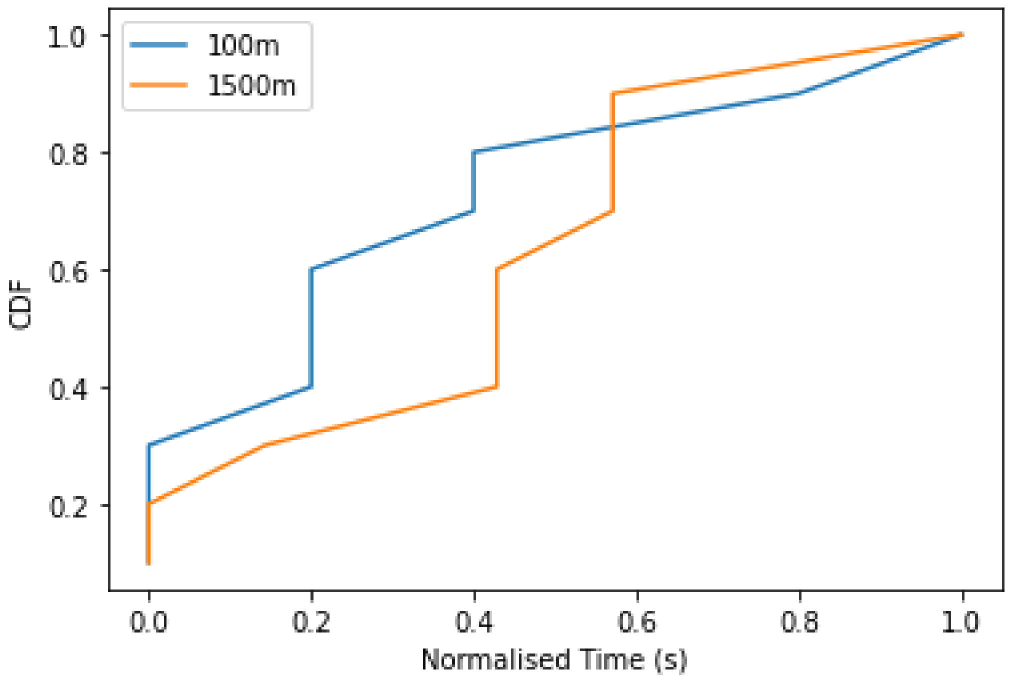
In the case of the 100m data set there is one outliers and in the case of the 1500m data set there are no outliers. This helps in implying that the data sets are non-monotonic.

A good test for non-parametric data is the Kolmogorov–Smirnov (KS) test.
The first step for a KS test is to plot the CDF of the data. In order to do this with our data sets, we need to normalise the time so that we can compare the underlying distributions. This can be done by dividing each data set by its max value.

The KS test is definded by hypotheses,
H_0: The data sets come from the same underlying distribution
H_a: The data sets do not come from the same underlying distribution

[156]:
```
def cdf(x):
    xs = np.sort(x)
    ys = np.arange(1, len(xs)+1)/float(len(xs))
    return xs, ys
```

```
a_t, a_cdf = cdf(df['100m'])
b_t, b_cdf = cdf(df['1500m'])

a_t = a_t - np.min(a_t)
normalised_a_t = a_t/np.max(a_t)
b_t = b_t - np.min(b_t)
normalised_b_t = b_t/np.max(b_t)

plt.plot(normalised_a_t, a_cdf, label='100m')
plt.plot(normalised_b_t, b_cdf, label='1500m')

plt.legend()
plt.ylabel('CDF')
plt.xlabel('Normalised Time (s)')
plt.show()
```



If the data set is large, D can be approximated by

$$D = c(\alpha)\sqrt{\frac{n_1 + n_2}{n_1 \times n_2}}$$

where c(α) = 1.36 at 95% CI, which can be found in a KS coefficient table.

```
[157]: D = 1.36*np.sqrt((len(df['100m'])+len(df['1500m']))/
        ↪(len(df['100m'])*len(df['1500m'])))
       print('D = {:.2f}'.format(D))
```

```
D = 0.61
```

This can be compared to the critical value 0.7 which can be found in the KS table.

$0.61 < 0.7$, therefore we cant reject the null hypothesis that the data sets come from the same underlying distribution.

This value for D can be compared to the `scipy.stats.ks_2samp` test result.

[158]:
```
from scipy.stats import ks_2samp

ks_2samp(normalised_a_t, normalised_b_t)
```

[158]:
```
KstestResult(statistic=0.5, pvalue=0.16782134274394334)
```

\$0.17 > 0.05 \$, the p-value is greater than α at a 95% confidence interval, therefore we cant reject the null hypothesis that the data sets come from the same underlying distribution.

The D values, while similar, are slight different between the two methods, this is probably because we cant use the equation

$$D = c(\alpha)\sqrt{\frac{n_1 + n_2}{n_1 \times n_2}}$$

because we dont have a large enough sample size.