MECH60017/MECH96038 STATISTICS COMPUTING TUTORIAL SHEET I

Instructions and relevant commands are written in **black** for MATLAB, in **blue** for Python and in **red** for R (if no Python or R commands are provided, that means they are not needed). The resulting figures presented are the ones obtained using MATLAB - these can differ slightly when using Python or R.

Plots can be a very powerful way to help reveal and understand patterns in data. In this tutorial you will develop your skills in MATLAB/Python/R and some useful packages/toolboxes for statistical data analysis. You will:

- **import data**
- **make some simple plots**
- **judge how these plots can help you to understand a real-world phenomenon.**

You are a time traveller who has arrived on 15th August 1985 at Yellowstone National Park, and just missed seeing the famous geyser, "Old Faithful", erupting. You would like to predict how long you need to wait to see the next eruption. Fortunately the park ranger has supplied you with some data on waiting times between eruptions since the 1st August 1985.

1 Importing data and summary statistics

In this exercise you will import the `faithful.csv` data file and make a boxplot of the waiting times, with labelled axes. You will also calculate some summary statistics for waiting time. The data file is on Blackboard (Tutorials/MATLAB Tutorials/Datasets).

There are several ways to import data files into MATLAB, Python or R. For this exercise you are working with text files. Read the MATLAB help file `https://uk.mathworks.com/help/matlab/import_export/ways-to-import-text-files.html`. For Python, you will have to import the `pandas` toolbox and read the relevant help file `https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html`. R requires loading the `readr` package; its documentation can be found in `https://cran.r-project.org/web/packages/readr/readr.pdf`.

Useful commands for this exercise are:

- `boxplot`/ `boxplot` (from the `seaborn` toolbox)/ `boxplot` - to produce a boxplot
- `xlabel`/ `xlabel` / `xlab` argument of the `boxplot` command - label x-axis
- `ylabel`/ `ylabel` / `ylab` argument of the `boxplot` command - label y-axis
- `max`/ `max` from the `NumPy` library / `max` – returns the maximum value of an array.

- `min`/ `min` from the `NumPy` library / min – returns the minimum value of an array.
- `mean`/ `mean` from the `NumPy` library / mean – returns the mean (average) value of the array.
- `median`/ `median` from the `NumPy` library / median – returns the median value of an array.
- `mode`/ `mode` from the `stats` module of the `SciPy` library / In R, the mode is not directly computable unless the user creates such a function (see `https://www.tutorialspoint.com/r/r_mean_median_mode.htm` for more details) – returns the mode value of an array.
- `std`/ `std` from the `NumPy` library / `sd` – returns the standard deviation of an array.
- `var`/ `var` from the `NumPy` library / `var` – returns the variance of an array.
- `sort`/ `sort` / `sort` – sorts the elements of the array in ascending order.
- `quantile`/ `quantile` from the `NumPy` library / `quantile` - returns quantiles of an array corresponding to specified cumulative probabilities.

Use `help` in MATLAB, Python or R to get information on how these commands work.

1. Download the `faithful.csv` file for this example to an appropriate local directory.
2. Open the `faithful.csv` file in a text editor. The data set includes information on all eruptions between 1st and 15th August, 1985 (up to the time you arrived). You should see three columns, headed "waiting time", "duration", and "day". Each row corresponds to a single eruption, and rows are ordered in time. For each eruption, the column labelled "waiting time" contains the waiting time in minutes from the end of the previous eruption to the start of the current eruption. The column labelled "duration" contains the duration in minutes of the current eruption. The column labelled "day" gives the day of the eruption.
3. Import the data in one of MATLAB, Python or R.
4. Recalling that you are attempting to predict how long to wait for the next eruption, calculate what you think would be useful sample statistics for "waiting". Some example commands are listed above - for more details see the help files.
5. Use the `boxplot` function to produce a boxplot of `waiting`.
6. Use the `boxplot` function to produce a boxplot of `waiting` by `day`.
7. Use `xlabel` and `ylabel` to label the axes appropriately. Remember to state units of measurement.
8. How are the ends of the whiskers determined? Can you change this?
9. What patterns in waiting time, if any, do you notice? What can you say about day-to-day variation? How long do you predict you need to wait for the next eruption?

You should produce a boxplot like the following:

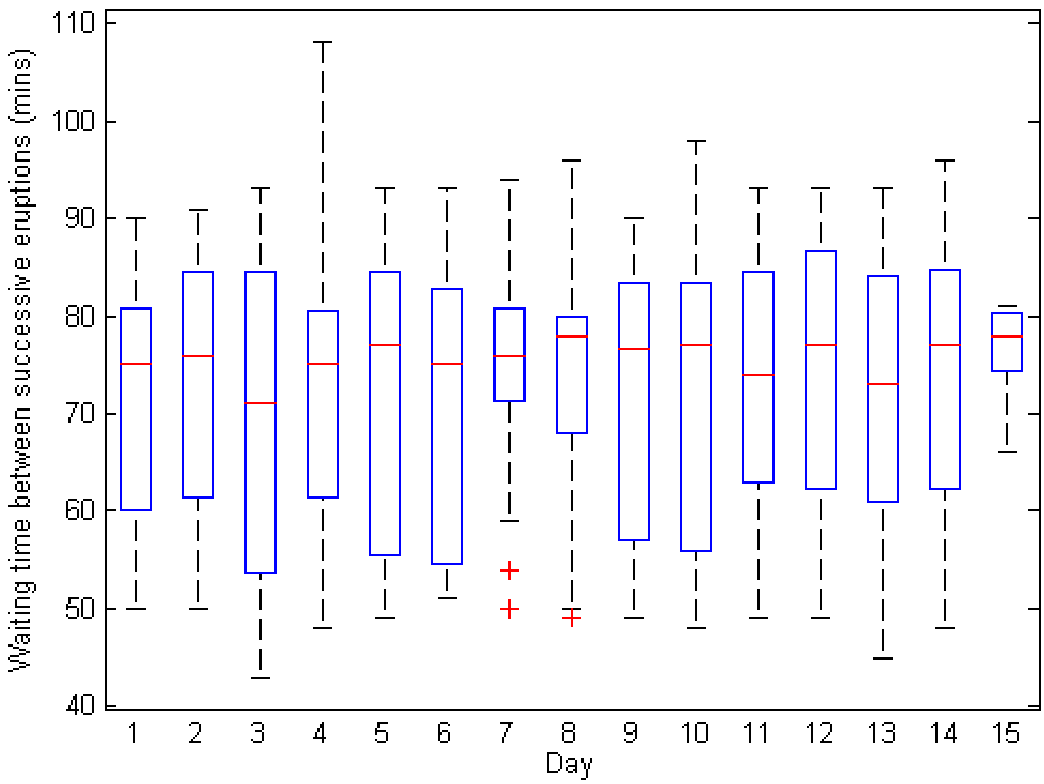

Figure 1: Boxplot of daily waiting time between successive eruptions.

2 Histograms and kernel density plots

In this example you will utilise the functions:

- `hold on` - to overlay plots
- `hist`/ `histogram` from the `NumPy` library / `hist` - to compute histograms
- `bar`/ `barplot` from the `Seaborn` library / `barplot` - to produce bar plots
- `linspace`/ `linspace` from the `NumPy` library / `seq` - to produce a vector of equally spaced values between some specified minimum and maximum value
- `ksdensity`/ `kdeplot` from the `Seaborn` library / `density` - to compute (and plot) a kernel density estimate of the population density
- `plot`/ `plot` - for a line plot of the kernel density estimate against bin.
- `legend` / `legend` from the `PyPlot` module of the `MatPlotLib` library / `legend` - add legend to a plot.

Use `help` in MATLAB, Python or R to get information on how these commands work.

1. Use `min` and `max` to compute the smallest and largest waiting times between eruptions.
2. Use `linspace`/ `linspace` / `seq` to compute centres of 10 binning intervals between the smallest and largest waiting times. Name the bin variable `bins`.
3. Use `hist`/ `histogram` / `hist` to compute the number of waiting times in each bin interval. Store the result in a variable called `freq`.
4. Make a histogram using the function `bar`/ `barplot` / `barplot`. As `waiting` is a continuous variable, your bars should have `WIDTH` set to 1 - see `help` file.
5. Compute the relative frequency and store in `relfreq`. Hint: rescale the frequencies - the area under the relative frequency histogram must be 1.
6. Make a relative frequency histogram using the function `bar`. Use `hold on` so you can overlay the kernel density estimate. If working in Python, you may use `distplot` from the `Seaborn` library to create this plot in one command!. In R, use the command `lines` after creating the histogram to overlay the kernel density estimate.
7. By default, the `ksdensity(X)` command estimates the kernel density at 100 points across the range of `X`. Use `ksdensity(waiting, bins)` to generate a kernel density estimate of the waiting times, where the kernel densities are estimated at the ten bin centres determined for the histogram.
8. Overlay the scaled kernel density estimate on top of the histogram using a line plot (function `plot`). If working in Python, you may use `distplot` from the `Seaborn` library to create this plot in one command!. In R, use the command `lines` after creating the histogram to overlay the kernel density estimate.
9. Repeat the above using 20 and 50 bin intervals. How many bins would you choose to best represent the distribution of waiting times? A commonly used criterion to determine the number of bins is Sturges' formula: $1 + \log_2 n$, where n is the sample size - do you think this is a good choice for these data?

10. Comment on the histogram and kernel density estimate versus the boxplot for the "Old Faithful" waiting times.

11. Using `ksdensity`/ `kdeplot`/ `density` with the default 100 estimated points, calculate kernel density smooths with different *bandwidths*. Use `hold on` to overlay the plots. Use `legend` to add a legend to the plot. **(Notice that different bandwidth values may be needed in Python or R to obtain the same plots as the ones depicted in the following page, obtained in MATLAB - that is because of the different kernels and default bandwidths chosen in each programming language).**

See if you can reproduce a set of histograms like the following:

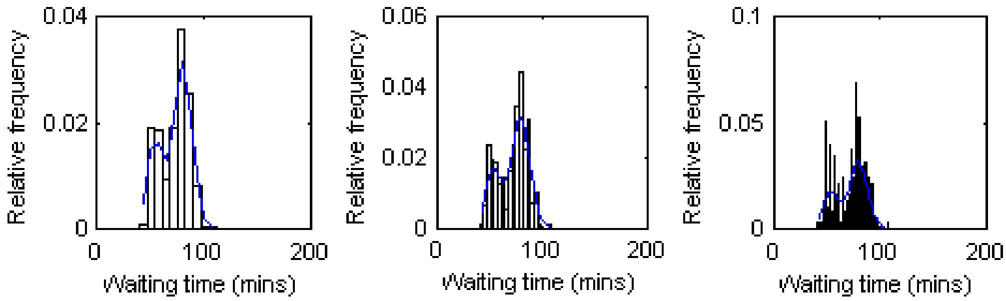

Figure 2: Histogram with kernel density estimate of waiting times between eruptions of Old Faithful, for 10, 20 and 50 bins.

And kernel density estimates:

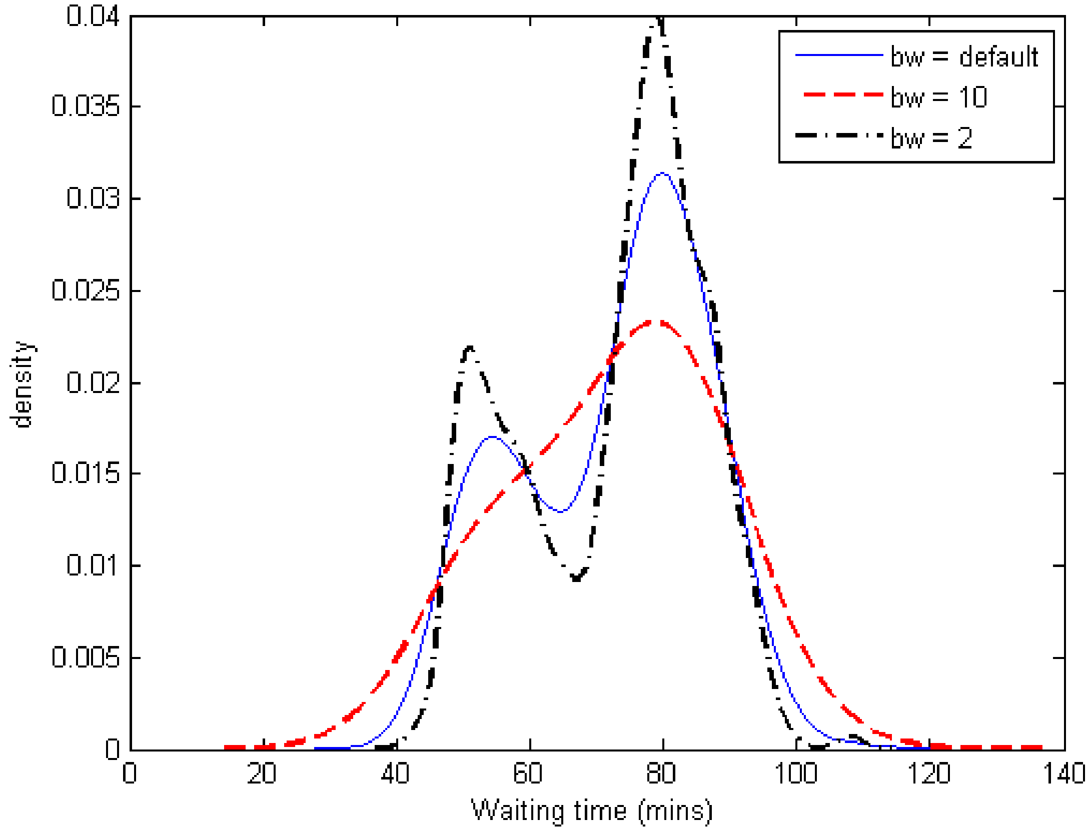

Figure 3: Kernel density estimates of waiting times between eruptions of Old Faithful (varying bandwidths).

3 Plotting consecutive eruption waiting times

Is there a pattern in waiting times over time? Plotting the waiting times in order may help. Here we plot consecutive waiting times for each day separately. You will use the functions:

- `find`/ `loc` / `which` - find elements in a vector or matrix matching a value
- `subplot` / `subplots` from the `PyPlot` module of the `MatPlotLib` library / `par(mfrow(...))` - to plot multiple plots on one figure
- `xlim` / `set(xlim=(·, ·))` / `xlim` argument of the `plot` function - to set x-axis limits
- `ylim` / `set(ylim=(·, ·))` / `ylim` argument of the `plot` function - to set y-axis limits
- `title` / `set(title="...")` / `main` argument of the `plot` function - to give a plot a title

For this exercise, you should revise the use of the `for` loops and `if/else/end` statements in MATLAB, Python or R.

1. Use a `for` loop and the `subplot`/ `subplots` / `par(mfrow(...))` command to create one plot of `waiting` for each day (1-15) on the same figure. Use `find`/ `loc` / `which` to identify each day (see section 5 in this tutorial for hints on using `find`/ `loc` / `which`).

   You should use `xlim` and `ylim` to ensure all your plots have the same axes and add a title specifying the day using `title` / `set(title="...")` / `main`.

   Use `set(gca, 'FontSize', 8)` to set the font size of a particular subplot.

See if you can reproduce a final plot like the following:
What do these plots suggest to you?

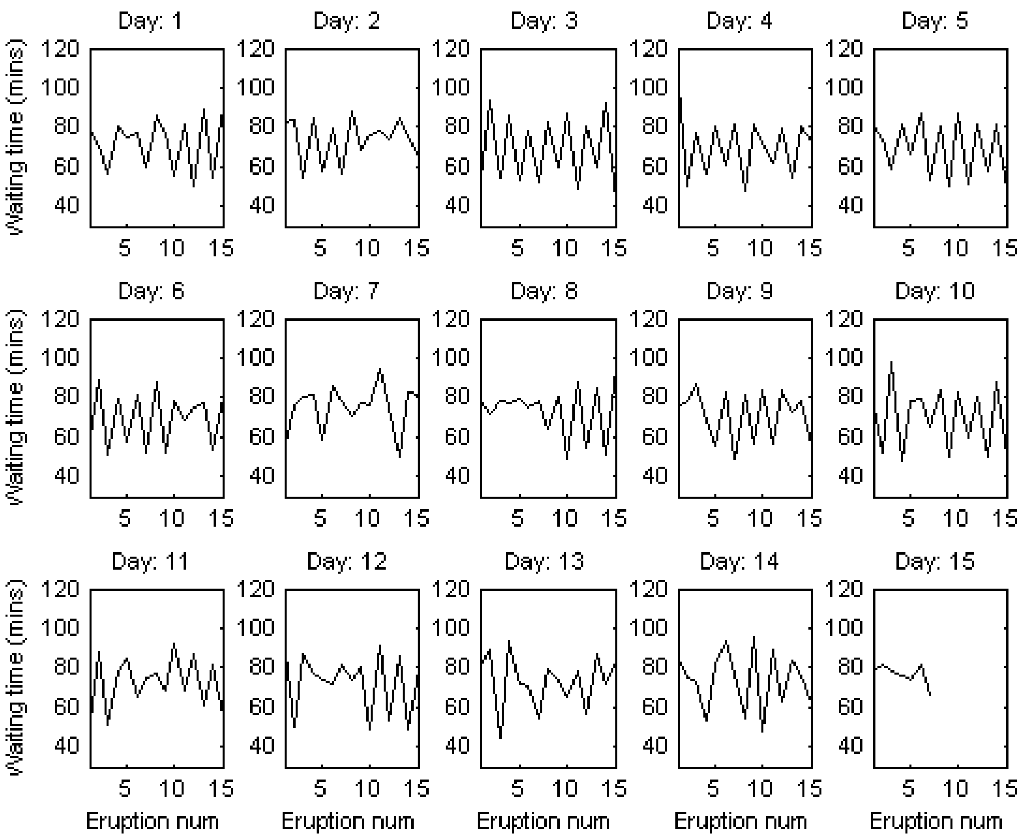

Figure 4: Consecutive waiting time between eruptions of Old Faithful, 1-15 August 1985.

4 Scatterplots and linear regression

Can you predict waiting time for an eruption using the duration of the previous eruption? A scatterplot is a useful visual aid. In this example you will utilise the functions:

- `regress` / `polyfit` from the `NumPy` library / `lm` - perform simple linear regression
- `length` / `len` / `length` - calculates length of a vector
- `ones` - creates vector or matrix containing the number 1
- `lagmatrix` / `lagmat` from the `tsa.tsatools` module of the `statsmodels` library / `lagmatrix` from the `tsutils` package - create lagged time series

1. Use `length` / `len` / `length` to calculate the number of data points and store this value in `n`.
2. Use the command `lagmatrix` / `lagmat` / `lagmatrix` to make a variable called `lagduration` containing the lagged value of `duration`.
3. Use the `plot` function to produce a scatter plot of `waiting` against `lagduration`.
4. Use `xlabel` and `ylabel` to label the axes appropriately.
5. Use the `regress` / `polyfit` / `lm` function to do a linear regression with `Y = waiting` and `X = [ ones(n, 1) lagduration ]` and store the regression coefficients in `B`. You will learn more about linear regression next term.

   Note that we need to use `ones(n, 1)` to create an $n \times 1$ vector of ones in order to have a constant coefficient in the linear regression (i.e. we are assuming $y = b_1 + b_2 x$ without the `ones` we would be assuming $y = b_1 x$). In Python, you can set the second argument of `polyfit` to `1` to indicate the inclusion of an intercept coefficient. In R, there is no need to specify the intercept, so just using `X = lagduration` will work, as `lm` assumes that an intercept term is included in the linear model.
6. Calculate a line of best fit using `waitingest = B(1) + B(2)*lagduration;`
7. Use the `plot` function to produce a line plot of the predicted waiting time `waitingest` against lagged duration `lagduration`. You will need to use `hold on` to overlay the line plot on your scatter plot. What does this plot suggest to you? Do you think the linear fit is appropriate? What is your predicted waiting time now? Is there other information you could use to improve your prediction?

You should produce a scatter plot like the following:

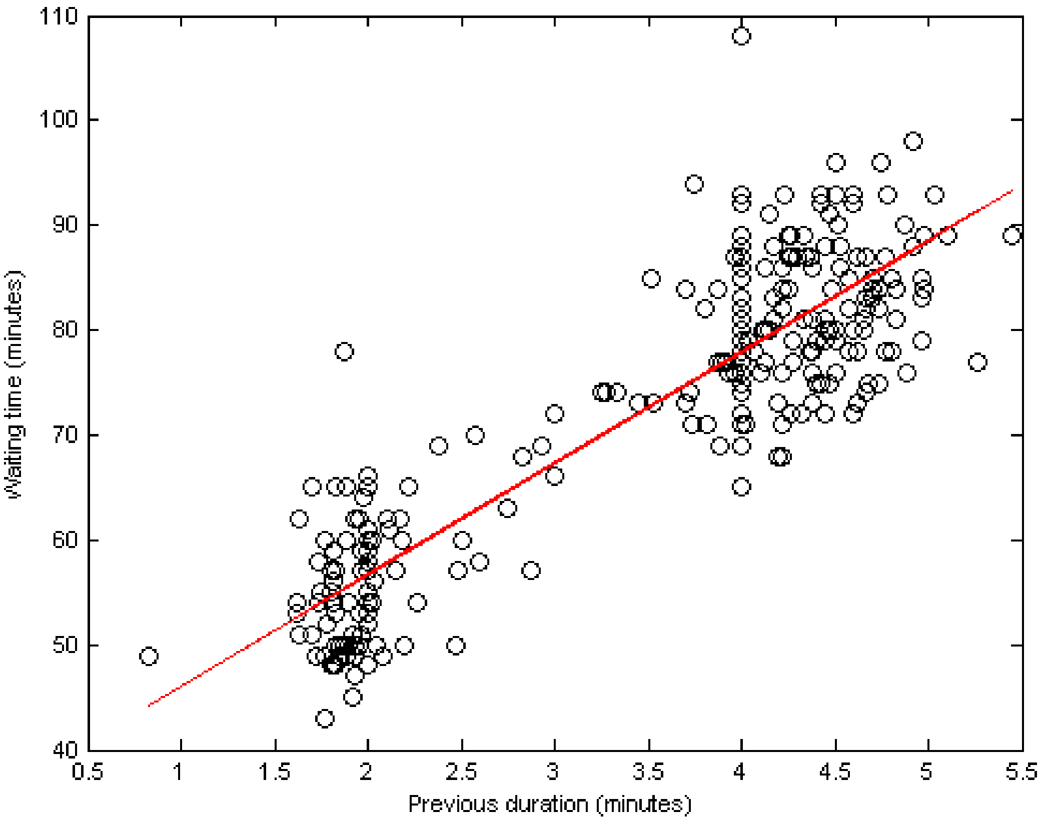

Figure 5: Scatter plot and linear regression fit for Old Faithful data.

5 K-means clustering

In this example you will use the functions:

- `kmeans` / `KMeans` from the `cluster` module of the `sklearn` library / `kmeans` - to do k-means clustering on two-dimensional data
- `find`/ `loc` / `which` - find elements in a vector or matrix matching a value

1. Use the lagged duration calculated in part 4 `lagduration`
2. Make a new scatter plot of `waiting` versus `lagduration`. How many distinct groups of eruptions can you see?
3. Use `kmeans` / `KMeans` / `kmeans` to cluster the data into 2 classes using:

   `C = kmeans([lagduration, waiting], 2);`
   `C = KMeans(n_clusters=K, random_state=0).fit(X)` (here `X` is a data frame including `lagduration` and `waiting` as its 2 columns).
   `C <- kmeans(cbind(lagmatrix[2:285], waiting[2:285]), centers=K)` (where `K` is the number of clusters).

   The variable `C` will contain numbers 1 to 2 corresponding to the class that that particular data point has been assigned to.

4. Using a `for` loop and the `find` function, find all the waiting times assigned to each of the two classes and overlay two new scatter plots in different colours.

   The following code fragment may be of use:

```
% define some colours
col{1} = 'r'; col{2} = 'g';

% for each class
for c = 1 : 2

    % find data with class equal to "c" and store locations in "loc"
    loc = find( C == c );

    % plot using "o" and no line and different colours
    plot(width(loc), weight(loc), 'color', col{c}, ...
     'marker', 'o', 'markersize', 4, 'LineStyle', 'None');

end
```

```
for i in range(K):
    label0 = X.loc[C.labels_==0]
    label1 = X.loc[C.labels_==1]


plt.scatter(x=label0['lagduration'], y=label0['waiting'], color = 'red')
plt.scatter(x=label1['lagduration'], y=label1['waiting'], color = 'green')
```

```
C <- kmeans(cbind(lagmatrix[2:285], waiting[2:285]), centers=K)

plot(y=waiting[2:285], x=lagmatrix[2:285], pch=16,
     col=as.factor(C$cluster))
```

See if you can reproduce a final plot like the one overleaf.

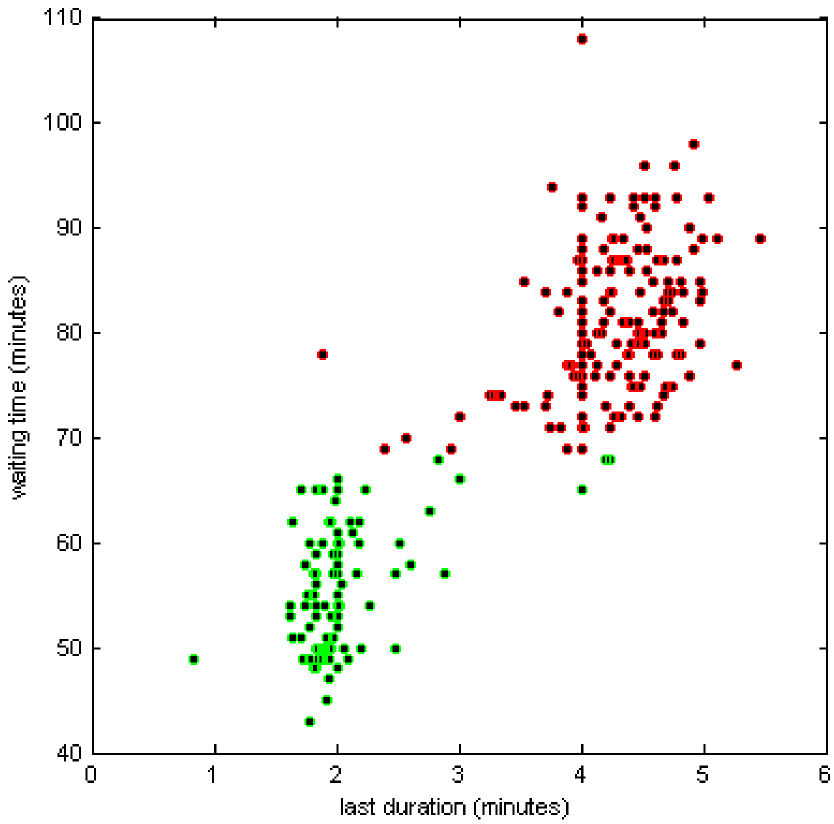

Figure 6: k-means clustering of waiting time data.

Repeat the exercise using a different number of clusters and comment.
Repeat the exercise using lagged waiting time and waiting time as your two variables.

6 Final Questions

1. Compare the information revealed by each graph. What is gained (or lost) by each representation?

2. How useful were the summaries of location for predicting the expected waiting time to the next eruption?

3. What is your final prediction of waiting time? Can you predict more than one waiting time ahead? Will your predictions apply today, and why/why not? Which graphical representation would best communicate your predictions? What other information would you provide?

4. You decide to stay in the park for the rest of the day to collect some more data and validate your prediction. The data are in `faithful15`. Import `faithful15` into MATLAB/Python/R. How close are your final predictions for the next waiting times in comparison with the actual values?

5. What have you learned about (a) variation and (b) making predictions?