Practical 4: solutions

Jumping Rivers

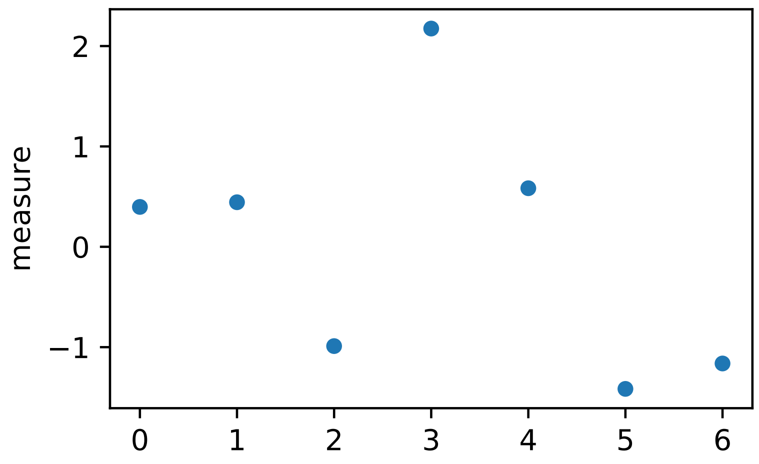
In this question, we are going to use a `for` statement to loop over a large data set and construct some scatter plots. To generate the data, run the following piece of code

```
import jrpyprogramming

exper = jrpyprogramming.datasets.experiment.load_data()

# change the column names
exper.columns = ["measure", "time", "treat"]
```

The data frame `exper` represents an experiment, where we have ten treatments: $A, B, \ldots, J$ and measurements at some time points. We want to create a scatter plot of measurement against time, for each treatment type.

2. First we create a scatter plot of one treatment:

```
group = exper[exper.treat == "A"]

import matplotlib.pyplot as plt

group.plot.scatter(x = "time", y = "measure")
```



3. To generate a scatter-plot for each treatment, we need to iterate over the different treatment types:

```
for i in exper.treat.unique():
    group = exper[exper.treat == i]
    group.plot.scatter(x = "time", y = "measure")
    plt.show()
    input("Hit enter for next plot")
```

- What does `exper.treat.unique()` give?

```
print('It gives all the unique treatments.')
```

```
## It gives all the unique treatments.
```

- In the `for` loop, what variable is changing? What are it's possible values?

```
print('The treat variable is changing. \
It goes through the different treatments.')
```

```
## The treat variable is changing. It goes through the different treatments.
```

- What does the `input()` function do?

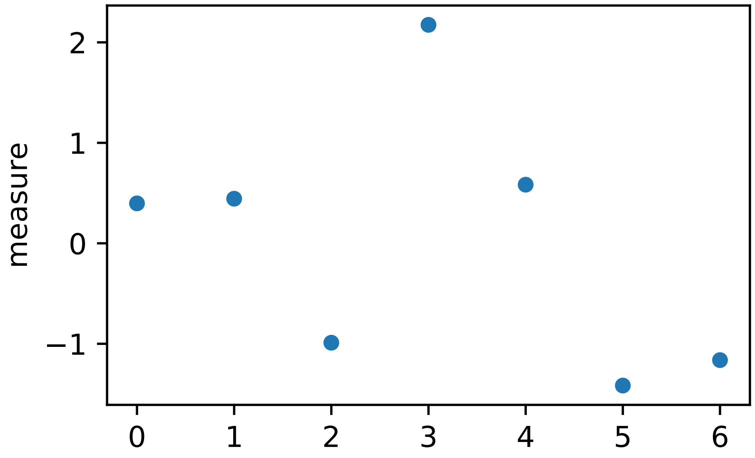
```
print(' It halts execution, waits for user input')
```

```
##  It halts execution, waits for user input
```

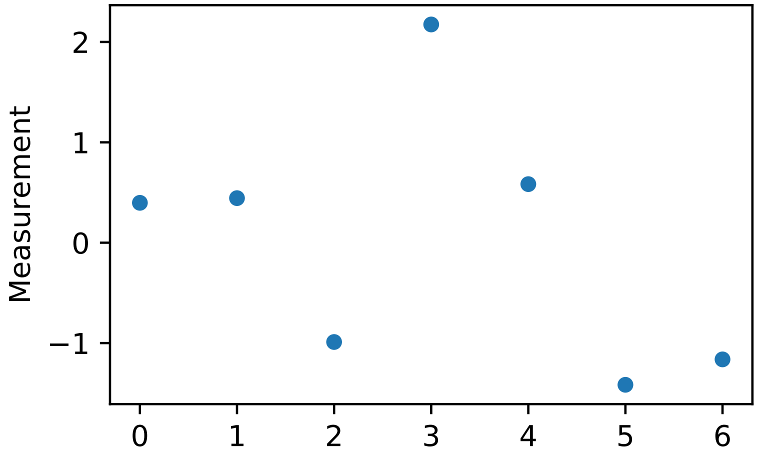
Questions

1. The default axis labels aren't great. So we can change the x-axis label using `plt.xlab()`:

```
group.plot.scatter(x = "time", y = "measure")
plt.xlabel("Time")
```



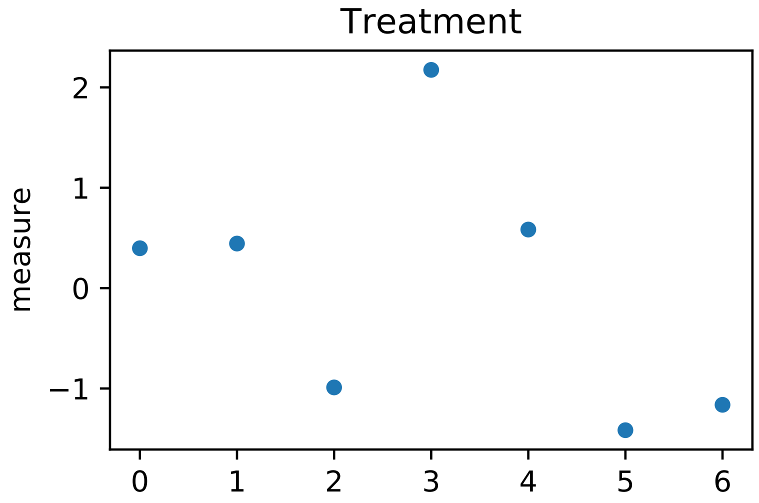
```
group.plot.scatter(x = "time", y = "measure")
plt.xlabel("Time")
plt.ylabel("Measurement")
```



Use the `ylab` argument to alter the y-axis label.

2. To add a title to a plot we use the `plt.title()` argument, viz:

```
group.plot.scatter(x = "time", y = "measure")
plt.title("Treatment")
```



3. We can concatentate strings/characters using `+`,

```
"Treatment "+"A"
```

```
## 'Treatment A'
```

Rather than have a static title, make the title of each plot display the treatment type.

```
group.plot.scatter(x = "time", y = "measure")
plt.xlabel("Time")
plt.ylabel("Measurement")
plt.title("Treatment "+ i)
plt.show()
```

4. The y-axis range should really be the same in all graphics. Add a `ylim` argument to fix the range.
   Hint: Work out the range before the `for` loop.

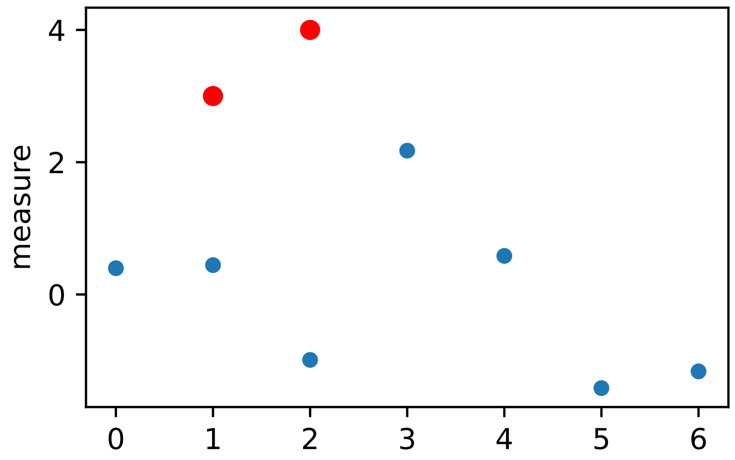
```
ymin = exper.measure.min()
ymax = exper.measure.max()

group.plot.scatter(x = "time", y = "measure")
plt.xlabel("Time")
plt.ylabel("Measurement")
plt.title("Treatment "+i)
plt.ylim([ymin, ymax])
plt.show()
```

5. At each iteration, use the `print()` function to print the average measurement level across all time points.
   Hint: You will have to convert the mean to a string, to do this use the `str()` function.

```
##Within the for loop have the line
print("The mean is "+str(group.measure.mean()))
```

6. On each graph, highlight any observations with a blue point if they are larger than the mean + standard deviations or less than the mean - standard deviations. Store your graph as a variable, then use the `.scatter()` method to plot additional points.
   Hint: You don't need `if` statements here. Just subset your data frame and pass this new data frame to the points function. For example, to highlight the points (1,3) and (2, 4) we use the command:

```
p = group.plot.scatter(x = "time", y = "measure")
p.scatter([1,2],[3,4], color = "red")
```



```
p = group.plot.scatter(x = "time", y = "measure")
plt.xlabel("Time")
plt.ylabel("Measurement")
plt.title("Treatment "+i)
plt.ylim([ymin, ymax])

## calculate outliers
measures = group.measure
upper_lim = np.mean(measures) + np.std(measures)
lower_lim = np.mean(measures) - np.std(measures)
outliers = group[(group.measure < lower_lim) | (group.measure > upper_lim)]

# add additional outlier points to the plot
p.scatter(outliers.time, outliers.measure, color = "red")
```

7.. Put your code, i.e. the `for` loop and plotting commands, in a function which takes the data frame as an argument.

```
## FULL SOLUTION
def viewgraphs(exper):
    ymin = exper.measure.min()
    ymax = exper.measure.max()
    for i in exper.treat.unique():
        group = exper[exper.treat == i]
        p = group.plot.scatter(x = "time", y = "measure")
        plt.xlabel("Time")
        plt.ylabel("Measurement")
        plt.title("Treatment "+i)
        plt.ylim([ymin, ymax])
```

```
measures = group.measure
upper_lim = np.mean(measures) + np.std(measures)
lower_lim = np.mean(measures) - np.std(measures)
outliers = group[(group.measure < lower_lim) | (group.measure > upper_lim)]

p.scatter(outliers.time, outliers.measure, color = "red")
input("Hit enter for next plot")
plt.show()
```