Statistics Guided Exercise

For using statistics with Python, we will be looking at the Pandas library. Pandas itself is build on top of another library, NumPy, and both have their own data structures. In this exercise, we will go over these data structures, and introduce you to Bokeh, which is a visualisation library you will be using in this exercise and the next for graphs and charts.

Pandas

Pandas is a Python library for data analysis in Python. It providies some useful functions and data structures for the collection and analysis of data. In particular, we will be making use of the **`DataFrame`** and **`Series`** classes.

A `DataFrame` object represents data in a series of rows (individual observations of data) and columns (features or variables) within those data. Each of those rows and columns can be extracted, and they then become a `Series`. We will work through an example to illustrate these concepts.

Importing Data

There are convenience functions to import data, such as **`read_json`** (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_json.html) and **`read_csv`** (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html) which, as their names suggest, will import data which is already in a particular format. For this example, we will import data from the MongoDB database we used in the exercise last week.

For this example, we will import the first 1000 documents (*MongoDB stores data records as BSON documents. BSON is a binary representation of JSON documents, though it contains more data types than JSON*), in the UK collection into a Pandas `DataFrame`. Run the cell below

In [1]:

```
# Convention is to import numpy and pandas with abbreviated names
# This means that instead of using pandas.read_csv, you would use pd.read_csv
import numpy as np
import pandas as pd

from bokeh.io import output_notebook, show
from bokeh.charts import *

# Import PyMongo, so that we can query some data
# 'mongodb://cpduser:M13pV5woDW@mongodb/health_data' is the location of the data we

from pymongo import MongoClient
client = MongoClient('mongodb://cpduser:M13pV5woDW@mongodb/health_data')
db = client.health_data

cursor = db.uk.find({}).limit(1000)

# Unfortunately, Pandas does not support PyMongo objects for import, so we need to c
listy = list(cursor)

# Create a Pandas DataFrame with the list object as a parameter
first_1000 = pd.DataFrame(listy)
```

Now we have our imported data in a `DataFrame` object. Like any other Python object, it has a collection of attributes and methods which we can use. We will go over some here, but see the documentation (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html) for a full list. We'll start by seeing what the data looks like by calling the `head()` function on the data:

In [2]:

```
# Filtering the data in the DataFrame to only return rows where RatingValue < 3
first_1000[first_1000['RatingValue'] < 3]
```

Out[2]:

	AddressLine1	AddressLine2	AddressLine3	AddressLine4	BusinessName	BusinessType
3	NaN	30-34 Holmrook Road	Preston	NaN	A L Salam	Retailers - other
19	NaN	53A Fylde Road	Preston	NaN	Ale Emporium	Pub/bar/nightclub
64	NaN	130 Church	Preston	NaN	Ayaan's Miami	Takeaway/sandwich shop

In [3]:

```
# We can also create a DataFrame which has only some columns
three_columns = first_1000[['RatingValue', 'FHRSID', 'PostCode']]

print('DataFrame with COLUMNS only:\n\n', three_columns.head())

# Or some rows
# Print the Dataframe with rows 950 to 960, could be any number of rows
print("\nDataFrame with a selection of ROWS:")
first_1000[950:960]
```

```
DataFrame with COLUMNS only:

    RatingValue  FHRSID PostCode
0          3.0   90105  PR1 7BE
1          5.0  479725  PR1 7BY
2          5.0  479676  PR1 3BT
3          2.0  135423  PR1 6SR
4          5.0  759083      NaN

DataFrame with a selection of ROWS:
```

Using the existing `first_1000` DataFrame, try and create a dataset which outputs the columns FHRSID, PostCode, LocalAuthorityName, with any establishment where `RatingValue < 3`

In [2]:

```
# YOUR CODE HERE
three_columns = first_1000[first_1000['RatingValue'] < 3][['FHRSID', 'PostCode', 'Lo
print('DataFrame with COLUMNS only:\n\n', three_columns)
```

```
DataFrame with COLUMNS only:

      FHRSID PostCode LocalAuthorityName
3    135423  PR1 6SR            Preston
19   479641  PR1 2XQ            Preston
64   335295  PR1 3BT            Preston
85   448032  PR1 4DX            Preston
108  454884  PR1 1PX            Preston
120   51647  PR1 6XH            Preston
127  448082  PR1 2XQ            Preston
180  369063  PR2 6NH            Preston
189   69893  PR1 8JD            Preston
206   86973  PR1 1PX            Preston
230  637915  PR1 5LD            Preston
265  335352  PR1 8RQ            Preston
272  121600  PR1 1TS            Preston
273  133196  PR2 2DX            Preston
276  335380  PR1 7BE            Preston
292  335301  PR2 6BU            Preston
327  479591  PR1 7AT            Preston
350  201373  PR2 2DU            Preston
398  629593  PR1 5NU            Preston
426  479519  PR1 4SS            Preston
432  115804  PR1 5HH            Preston
459  479764  PR1 2AR            Preston
469  479921  PR1 4ST            Preston
474  768229  PR2 1AU            Preston
518  137209  PR1 4SU            Preston
561  850075  PR1 7JN            Preston
575  479767  PR2 1XN            Preston
632  201370  PR1 5EA            Preston
648  448112  PR1 7JS            Preston
654   70877  PR1 5QS            Preston
663   97856  PR1 8HJ            Preston
665  708327  PR1 7RA            Preston
687  335282  PR1 3BQ            Preston
700  335364  PR1 3YH            Preston
704  726619  PR1 5XA            Preston
719  479766  PR1 2EE            Preston
752  121893  PR2 6YS            Preston
787   92213  PR4 0BJ            Preston
821  136094  PR1 6EY            Preston
827  670391  PR1 6QY            Preston
828   78740  PR2 3XA            Preston
829  510354  PR1 2UP            Preston
856  479530  PR1 4TA            Preston
863  479853  PR1 5RY            Preston
868  479778  PR1 5TR            Preston
871  108298  PR1 3YH            Preston
896  136644  PR1 6QA            Preston
912   83211  PR1 5AS            Preston
925  335327  PR2 9UP            Preston
936   94388  PR1 2AR            Preston
954  479819  PR1 5XA            Preston
```

A `DataFrame` is an object in the `Pandas` library, but in addition we have the `Series` object, a collection of which makes up the `DataFrame`.

Many of the operations we can perform on a `Series` can also be performed on a `DataFrame`. It is a `Series` object which we will be using this week.

It is possible to perform an operation on each element in the `Series`, as well as call functions which require all of these such as `mean()`.

In [ ]:

```
print(first_1000['RatingValue'].head(), '\n')
print(first_1000['RatingValue'].head() * 100, '\n')
print(first_1000['RatingValue'].head() * 23 > 100)
```

As well as being a part of a `DataFrame`, it is possible to create a `Series` from a list type object, for example see the code below:

In [ ]:

```
s = pd.Series([8,6,2,7,9,6])
print(type(s))
print(s)
```

Create a `Series` object `rating_series` which contains the `RatingValue` column from the `first_1000` `DataFrame` object.

Then display descriptive statistics from that object (mean, median, mode etc). You can see the full list of available functions in [the documentation (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html)](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html)

In [3]:

```
# YOUR CODE HERE
rating_series = pd.Series(first_1000['RatingValue'])
mean = rating_series.mean()
median = rating_series.median()
mode = rating_series.mode()

print(mean)
print(median)
print(mode)
```

```
4.29613733906
5.0
0    5.0
dtype: float64
```

Bokeh Charts

Bokeh uses Pandas data structures as the basis for its charts. We will now place the data structure we just generated onto various types of charts.

In Bokeh, there is a class **`Plot`** which is the basis for the visualisations we will consider in more detail next week. A **`Chart`** is a *subclass* of **`Plot`**, which is designed to allow the generation of common charts with the minimum amount of code.

At its simplest, these charts simply accept an object such as a `DataFrame` or `Series`, and will output a chart. For example, a simple bar chart looks as follows:

In [13]:

```
# To display the chart in the notebook, we need to run this function, otherwise cal
output_notebook()
# Create a simple bar chart with made up data
bar_chart = Bar(pd.Series([8,6,2,7,9,3]))
print(type(bar_chart))
# Display the bar chart
show(bar_chart)
```

(http://bokeh.pydata.org) Loading BokehJS

```
<class 'bokeh.charts.chart.Chart'>
```

There are other charts which you can use such as a Histogram, Line graph, and scatter plot. View the Bokeh **user guide for charts (http://bokeh.pydata.org/en/latest/docs/reference/charts.html)** to see the options available to customise the chart created above. In addition it is possible to customise the display.

Randomness

A feature which NumPy supports is that of generating random numbers. This is important, for example, in generating a random sample from a population. The randomness, however, is not *truly* random, but rather pseudo-random, i.e., it will generate predictable values based on the initial *seed* that it accepts.

This feature means that if we know the seed, we can reproduce the results we wish to share provided that we have the same data, which is a desirable property. Though this may sound counter-intuitive it allows others to run our code using the same seed and they will get the same output, the code can then be run using a different seed value.

Consider the scenario where you want to populate an array with random data, you can use the `numpy.random.randint` function as below:

In [ ]:

```
# import numpy.random
import numpy as np
# The numbers generated will include the low value
low = 0
# The numbers generated will not include the high value, but will go up to high - 1
high = 10
np.random.randint(low, high, size=10)
```

Create a loop with 10 iterations, where each iteration prints a randomly generated array of size 10. Notice how each array has set of different values.

In [ ]:

```
# YOUR CODE HERE
for ri in range(0,10):
    print(np.random.randint(low, high, size=10))
```

For many situations, this is desirable. However, where we want to be able to reproduce, e.g., sample sizes, we want our samples to be reproducible. To do this, we use the `RandomState` class in NumPy, where we specify our seed, as follows:

In [4]:

```
# Run this cell several times - observe the outcome
rs = np.random.RandomState(543210)
j = rs.randint(low, high, size=10)
print(j)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
 last)
<ipython-input-4-aa0d0d3ff07e> in <module>()
      1 # Run this cell several times - observe the outcome
      2 rs = np.random.RandomState(543210) #씨드벨류 543210을 넣으면 사용자가
전달한 시드값을 기반으로 함
----> 3 j = rs.randint(low, high, size=10)
      4 print(j)

NameError: name 'low' is not defined
```

In the cell below, generate the same loop as before, except this time instantiate the `RandomState` object to a value of 123456:

In [ ]:

```
# YOUR CODE HERE
rs = np.random.RandomState(123456)
j = rs.randint(low, high, size=10)
print(j)
```

IQR and Outliers

In the videos, Sergej talked about "outliers" in a dataset. In this worksheet, we'll give a slightly more detailed definition about what exactly they are, and the effect they can have on data.

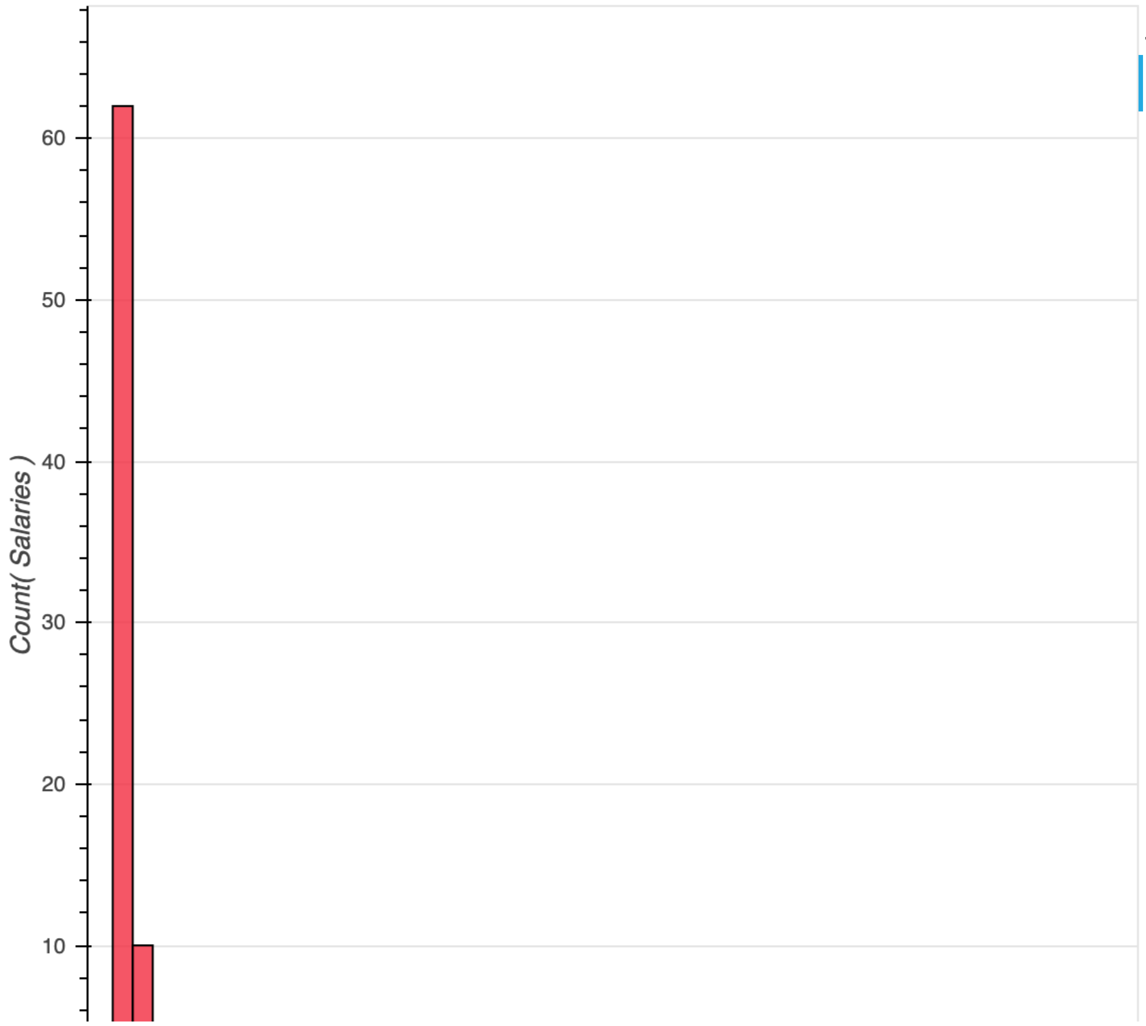
An outlier is a value which is atypical of the rest of the dataset. For example, consider this set of data from [searches on the UK income tax calculator (https://www.incometaxcalculator.org.uk/average-salary-uk.php)](https://www.incometaxcalculator.org.uk/average-salary-uk.php). If we draw a distribution of them, we will notice a big difference in the values:

In [4]:

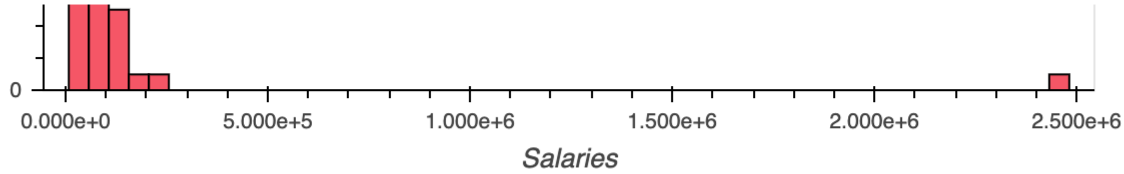
```
import pandas as pd
import numpy as np
from bokeh.charts import Histogram, output_notebook, show
from bokeh.models import Axis

salaries_list =  [30000,18000,25000,20000,40000,50000,35000,45000,22000,60000,24000,
             16000,100000,21000,26000,15000,32000,19000,17000,70000,27000,55000,185(
             36000,65000,42000,38000,12000,2481300,75000,33000,19500,43000,48000,12(
             17500,90000,34000,29000,16500,11000,31000,150000,37000,13000,22500,520(
             44000,200000,39000,46000,110000,27500,21500,47000,23500,15500,41000,265
             20500,14500,130000,250000,24500,28500,72000,140000,32500,8000,53000,95(


salaries = pd.DataFrame({'Salaries': salaries_list})
print('the max is %f' % (salaries.max()))

from bokeh.plotting import figure, show, output_file
output_notebook()
hist = Histogram(salaries, bins=50)
# Show absolute number on axis rather than E notation:
xaxis = hist.select(dict(type=Axis))[0]
xaxis.formatter.use_scientific = False
show(hist)
print(salaries.count())
```

```
the max is 2481300.000000
```

(http://bokeh.pydata.org) BokehJS successfully loaded.



(http




```
Salaries    80
dtype: int64
```

You will notice, the massive outlier on the right, where the person in question earns nearly £2.5 million. It makes it very difficult to get the chart to display anything useful, and has a significant effect on our data. For example, see the code below which shows the difference in number between the mean and the median:

In [5]:

```
print('The mean is: %f, and the median is %f' %
      (salaries.mean(), salaries.median()))
```

```
The mean is: 76371.250000, and the median is 31500.000000
```

Task: Remove the highest value from the dataset and see how this changes the mean and the median.

In [6]:

```
# YOUR CODE HERE
salaries = salaries.loc[salaries['Salaries'] < 2000000]
print('The mean is: %f, and the median is %f' % (salaries.mean(), salaries.median()))
```

```
The mean is: 45929.113924, and the median is 31000.000000
```

Moving the top value had a considerable effect on the mean value of the dataset, decreasing it by over £30,000, however the result is still quite a bit higher than the median. So although the £2.5 million figure is obviously an outlier, how can we define an outlier more concretely? To start, we will consider the interquartile range (IQR):

IQR

The IQR is calculated as follows:

1. Ordering the data by value
2. Taking the middle value from the *bottom* half of the data (lower quartile, known as Q1)
3. The median is known as Q2
4. Taking the middle value from the *top* half of the data (upper quartile, known as Q3)
5. The IQR is then calculated with Q3 - Q1

The Q1 and Q2 values are considered as the 25th and 75th percentiles, since they represent the values 25% and 75% through the ordered data. Luckily, there are functions within Pandas which allow the calculation of these percentiles, which provide us with the IQR: the `quantile` (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.quantile.html) function on a `DataFrame` which takes a float between 0 and 1 to get the appropriate percentile. For example, the median could be calculated as follows:

In [7]:

```
salaries.quantile(0.5)
```

Out[7]:

```
Salaries    31000.0
Name: 0.5, dtype: float64
```

Task: Calculate the IQR of the salaries data using Pandas. Print out the value of the upper and lower quartiles so you can check the answer is correct

In [8]:

```
# YOUR CODE HERE
upper = float(salaries.quantile(0.75))
lower = float(salaries.quantile(0.25))
iqr = upper - lower
print('Upper quartile:\t%f\nLower quartile:\t%f\nIQR:\t\t%f' % (upper, lower, iqr))
```

```
Upper quartile: 51000.000000
Lower quartile: 20250.000000
IQR:            30750.000000
```

Outliers

Having introduced the IQR, we can now consider what constitues an outlier. As a rule of thumb, an outlier can be defined as follows:

- `lower_quartile - (1.5 * IQR)`
- `upper_quartile + (1.5 * IQR)`

This is highly dependent on the data, and may not be appropriate for all situations, as is the decision of what to do with them. For the time being, we will simply exclude data which are outside these limits.

To do this, consider the following Pandas code, which excludes outliers from the salaries data. It uses a more complicated `.loc`, where it filters on two

In [9]:

```
# You don't need to write anything here
# Create the dataset again, rather than use the one with the top value taken out
salaries = pd.DataFrame({'Salaries': salaries_list})

salaries = salaries['Salaries'][(salaries['Salaries'] > (float(lower) - (iqr * 1.5)
                     & (salaries['Salaries'] < (float(upper) + (iqr * 1.5)))]

# salaries = salaries['Salaries'] < lower
# salaries = salaries.loc[salaries['Salaries'] <= lower]
print('The mean is: %f, and the median is %f' % (salaries.mean(), salaries.median()
print(salaries.count())
```

```
The mean is: 34202.816901, and the median is 28000.000000
71
```

The purpose of this exercise was to introduce the concept of an outlier, and how much of an effect it can have on data, and to give some practice using Pandas. There are many different ways that outliers could be defined, and circumstances where they could or should not be excluded.