Python for Data Science

Imagine there are two suspects who are making an exchange the authorities want to intercept, but getting an agent to the exchange location too soon would make them suspicious. The data we will use, which is freely available, is about traffic flow and was obtained from data.gov.uk. Using this data, and the functions provided in `magic.py`, your task is to determine the optimal time that the agent should leave from Southampton and travel up the M3 to Heathrow Airport.

The exercises below are not assessed, but will give you an introduction to using common Python data structures and methods. There's even a teaser for some statistical content, which will come in useful for week 4! The code in `magic.py` will also be helpful for you to look at some of the ways a Python script works.

In case you get stuck, the answers are available at 2. Python for Data Science - Answers.ipynb (2.%20Python%20for%20Data%20Science%20-%20Answers.ipynb)

Exercise 1

Please run the following code every time before starting the exercise. This helps make sure that all the helper functions are in place. You can run code by pressing `Crtl` + `Enter`, or `Shift` + `Enter` if you want to move onto the next cell.

In [8]:

```
from magic import *
```

Exercise 2a

Compute the amount of time it will take to get from Southampton to Heathrow using data from the first observed time (07:00). Use the `get_arrival_time` function in magic.py (magic.py) to help if you wish.

In [53]:

```
# N.B. model answers to the tasks in the 'Introduction Exercise' may be found in the
leaving_time = '07:00'

journeyData = get_data(1, 2013, 1, leaving_time)[0]
#비어있는 배열(array)에 할당

total_journey_time = 0

print('link\ttime taken for links')
print(journeyData, type(journeyData))

for d in journeyData:
    time_taken_per_link = journeyData[d]
    print(d, "\t", time_taken_per_link)
    total_journey_time += time_taken_per_link

print("Arrival time: %s" %get_arrival_time(leaving_time, total_journey_time))

# `2. Python for Data Science - Answers.ipynb`,  by following the link above


### END ANSWER
```

```
link    time taken for links
{1: 190.32, 2: 158.85, 3: 223.14, 4: 93.21, 5: 99.45, 6: 1342.67, 7: 8
7.55, 8: 264.51, 9: 257.65, 10: 387.22, 11: 310.45, 12: 623.01, 13: 71
6.88} <class 'dict'>
1        190.32
2        158.85
3        223.14
4        93.21
5        99.45
6        1342.67
7        87.55
8        264.51
9        257.65
10       387.22
11       310.45
12       623.01
13       716.88
Arrival time: 08:19:14
```

EXERCISE 2b:

Generalise your result so you are calculating the arrival time assuming you're leaving every 15 minutes between 05:00 to 09:00.

Using the code from EXERCISE 2a, create a function `calculate_arrivals`, which calculates the output for any provided inputs. The function should contain the parameters:

- `weekday` An integer, the day of the week
- `year` An integer between 2012 and 2014 inclusive
- `month` An integer between 1 and 12 inclusive
- `leaving_time` A string in the format HH:MM, e.g., 07:00

There are usually four weeks in the month, and the order of the data are not guaranteed, so you should not attempt to accurately determine the date at this point. The data returned will be for the first instance of the `weekday` found in the month

In [10]:

```
# Copy your answer from EXERCISE 1a here, and make it into a function called "calcu
# Use that to complete this exercise


# solution to Q2a, notice that get_arrival_time is returned and not printed
def calculate_arrivals(weekday, year, month, leaving_time):
    data = get_data(weekday, year, month, leaving_time)[0]
    total_time = 0
    #print('key\ttime taken (s)')

    for d in data:
        time_taken = data[d]
        #print(d, "\t", time_taken)

        total_time += time_taken

    return get_arrival_time(leaving_time, total_time)


### ANSWER

# set the leaving time to 05:00
leaving_time = '05:00'

# print headings
print('Departure time\tArrival time')

# using the range function to run through the loop 17 times, there are 4 hours betw
# each hour is divided into 4 sections (every 15 minutes) therefore 17 iterations a
print( calculate_arrivals(7, 2013, 1, '05:15'))

for lt in range(0, 17):
    arr = calculate_arrivals(7, 2013, 1, leaving_time)
    print("leaving: %s arriving: %s" % (leaving_time, arr))
    leaving_time = get_next_time(leaving_time)

### END ANSWER
```

```
Departure time  Arrival time
05:59:00
leaving: 05:00 arriving: 05:43:39
leaving: 05:15 arriving: 05:59:00
leaving: 05:30 arriving: 06:14:59
leaving: 05:45 arriving: 06:29:54
leaving: 06:00 arriving: 06:44:28
leaving: 06:15 arriving: 06:59:55
leaving: 06:30 arriving: 07:16:23
leaving: 06:45 arriving: 07:28:47
leaving: 07:00 arriving: 07:42:25
leaving: 07:15 arriving: 07:59:08
leaving: 07:30 arriving: 08:15:29
leaving: 07:45 arriving: 08:30:21
leaving: 08:00 arriving: 08:44:35
leaving: 08:15 arriving: 08:58:40
leaving: 08:30 arriving: 09:14:13
leaving: 08:45 arriving: 09:28:35
leaving: 09:00 arriving: 09:44:31
```

EXERCISE 2c:

Using the solution you produced for EXERCISE 2b, calculate the estimated arrival time for any day in the first week for the first six months, in a single year (2012 - 2014 inclusive). Return the results as a dictionary in the format:

`{leaving_time: [list of arrival times]}`, e.g. `{'08:15': ['08:57:04', '09:16:28', ...]}`

N.B. This might take a while to run! Build your solution gradually, so that when you test your code, it doesn't run the entire dataset.

In [27]:

```
# YOUR CODE HERE

def estimated_arrival_time():
    arrive_dict = {}
    #비어있는 사전 만들기 (출발시간이 키, 도착시간이 벨류)
    for w in range(1, 8):
        #7요일
        for y in (2012, 2013):
            #2012년도
            for m in range(1, 7):
                #6개월
                leaving_time = '05:00'
                #출발시각 5시 부터
                for leavingTime in range(0, 18):
                    #5시부터 15분짜리 섹션 17개 만들기 (9시 15분에 출발까지)
                    arr = calculate_arrivals(w, y, m, leaving_time)
                    #변수 arr에 할당은 calculate_arrivals사용

                    if not leaving_time in arrive_dict:
                    #출발시간이 사전에 들어있지 않은 경우에는
                        arrive_dict[leaving_time] = []
                        #사전에 출발시간 비워두기

                    arrive_dict[leaving_time].append(arr)
                    #계산한 arr변수를 arrive_dict 배열에 추가

                    leaving_time = get_next_time(leaving_time)
                    #루프 마지막이니 돌아야되니까 출발시각을 다음 출발시각으로 설정해주기
    return arrive_dict
    #사전으로 반환

arrive_dict = estimated_arrival_time()
#arrive_dict을 estimated_arrival_time로 지정

for arr_d in arrive_dict:
    #사전을 반복문으로 arrive_dict 안에 있는
    #설정해준 다음 출발시각을 반복하라는거지
    print("Leaving at time : %s\nArriving at time : %s\n" % (arr_d, str(arrive_dict[
    pass



### END ANSWER

#계산결과로 6개월간 첫 주의 7일 즉, 6개월 동안 매 달 첫 주 모든요일의 예상 도착시간을 계산 한 것
```

```
Leaving at time : 07:45
Arriving at time : ['08:25:32', '08:54:23', '08:50:23', '08:44:59', '0
8:31:23', '08:28:44', '09:08:25', '09:10:18', '08:50:51', '08:27:40',
'08:26:43', '09:03:17', '08:41:03', '09:00:41', '08:56:05', '08:37:5
1', '08:56:40', '08:25:55', '08:28:24', '08:57:40', '08:52:30', '08:4
0:09', '09:17:49', '08:44:22', '08:33:48', '08:38:50', '08:46:15', '0
8:41:30', '08:38:39', '08:38:45', '08:32:32', '08:55:17', '08:42:07',
'08:35:16', '08:40:25', '08:39:19', '08:41:26', '08:45:07', '09:00:0
8', '08:31:14', '08:46:18', '08:35:21', '08:32:39', '08:43:36', '08:5
5:57', '08:33:07', '08:46:39', '08:41:42', '08:32:41', '08:33:07', '0
8:31:58', '08:30:47', '08:30:50', '08:30:40', '08:31:59', '08:39:53',
'08:35:31', '08:30:29', '08:32:14', '08:35:17', '08:30:25', '08:28:4
7', '08:31:50', '08:28:36', '08:27:01', '08:28:04', '08:30:10', '08:2
```

```
5:50', '08:27:52', '08:27:49', '08:28:51', '08:31:30', '08:28:12', '0
8:29:27', '08:28:13', '08:26:16', '08:28:10', '08:27:53', '08:30:21',
'08:27:59', '08:30:50', '08:30:44', '08:26:55', '08:25:12']

Leaving at time : 07:30
Arriving at time : ['08:10:16', '08:39:19', '08:39:10', '08:30:49', '0
```

EXERCISE 3

We now have a considerable amount of data, but it's not very easy to read! On this amount of data it would be cumbersome to perform any calculations manually. Luckily, Python is good at automating these things employing libraries containing common descriptive statistics methods. We shall be using the NumPy and SciPy libraries.

The data calculated in EXERCISE 2b contains each leaving time, in the first week, of the first six months of a single year.

To calculate the optimal leaving time, we will be using linear regression to perform the analysis.

EXERCISE 3a

The first step is to determine the mean arrival time for each leaving time for each day in the first week.

Dealing with time data can be difficult, so to make calculations easier the arrival time is converted into the amount of seconds since midnight. To do this the function `get_time_in_seconds(time_str)` is used which performs this calculation with `time_str` being time in the format HH:MM:SS.

This will return two `array`s for each leaving time (as a string) with the mean arrival time (in seconds), e.g., `{'07:00': 25200}`. This is achieved using the `mean` function from the numpy library. For this to work the `numpy` library of functions must be imported which will allow use a special `numpy` data structure in arrays.

An array can be created from a list, by running `np.asarray`, as in the example below

In [14]:

```
# You don't have to write anything here

import numpy as np
#파이썬 모듈 넘파이를 가져옴

listy = [1, 2, 3, 4]
#바나나 일수도, 사과일수도 그냥 선언해주는거

array_obj = np.asarray(listy)
#위에서 할당한 listy를 배열로 할당했는데 np.asarray를 사용
#np.asarray(listy)는 np의 모듈 특징(특성)이거나 함수임
# => array_obj는 listy 내용을 사용하는 배열

print(listy, type(listy))
#타입은 리스트 타입이라는 뜻

print(array_obj, type(array_obj))
#타입은 임포트한 numpy에서 ndarray의 ad는 n개의 디멘션(차원) 배열임
np.mean(array_obj)
#평균
```

```
[1, 2, 3, 4] <class 'list'>
[1 2 3 4] <class 'numpy.ndarray'>
```

Out[14]:

```
2.5
```

EXERCISE 3b

Having calculated the mean values a visualisation of the output can be produced. To do this, we will be using the Python library `pyplot`, which is part of `matplotlib`. The output visualisation will be a scatterplot which has the leaving time as the predictor variable, and the arrival time as the response variable.

Use the `plt.scatter` function to create a scatterplot to visualise our times. This function takes as its first two parameters:

- A list of values for the X axis - predictor value, arrival time
- A list of values for the Y axis - response value, leaving time

In other words the arrival times are used to predict an optimal leaving time.

Use the values obtained from Exercise 3a to do this.

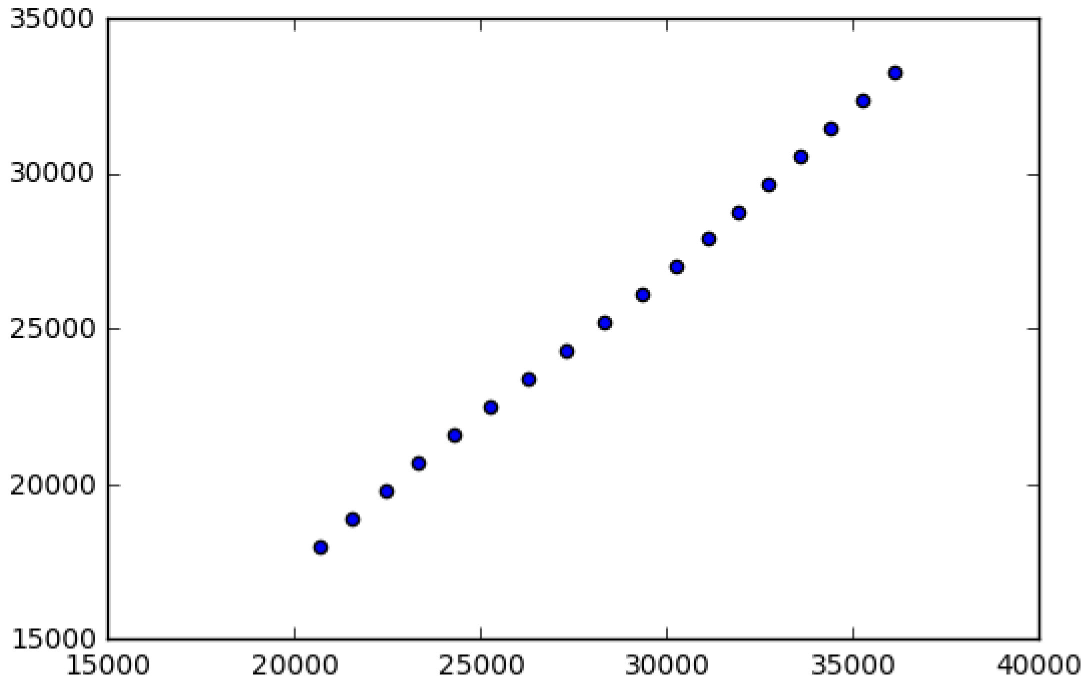
In [29]:

```
# Do the necessary imports
import matplotlib.pyplot as plt
%matplotlib inline

# YOUR CODE HERE

#다른 출발시간에 대한 도착시간, 도착시간의 값들을 써서 평균을 구하고 출발시간(0645등)에 적용

obj_loop = estimated_arrival_time()
#측정된 도착 시간 변수 만들기
departureValues_X = []
#x에 대한 출발값
responseMeanVaues_Y = []
#y에 대한 대응 평균값

for k in obj_loop.keys():
#키는 사전에만 있음, 리스트에는 멤버만 있음 -> 결국 사전 형태의 배열이 필요함
#obj_loop.keys()에는 (예상 도착시간으로 생성된?) (내생각은 출발시간) 키가 들어있고
    listNew = []
    #숫자로 변환된 도착시간들 넣을거임
    responseMeanVaues_Y.append(get_time_in_seconds(k))
    #responseMeanVaues_Y 리스트에 키에 반복된 값 넣기.

    for li in obj_loop[k]:
        #대응 평균값 Y 구하기
        #obj_loop[k] 에는 값(k) (출발?시간)이 들어있다.
        listNew.append(get_time_in_seconds(li))
        #비어있는 리스트 listNew에 출발?시간 get_time_in_seconds(li)값 (li = k) 넣어주기

    mean = np.mean(np.asarray(listNew))
    #도착시간들의 평균 값을 넣은 리스트생성
    departureValues_X.append(mean)
    #X에 삽입

departureValues_X = np.asarray(departureValues_X)
responseMeanVaues_Y = np.asarray(responseMeanVaues_Y)

print(departureValues_X)
print(responseMeanVaues_Y)

plt.scatter(departureValues_X, responseMeanVaues_Y)
### END ANSWER %
```

```
[ 31085.79761905  30241.30952381  28322.0952381   26263.13095238
  20659.79761905  32733.96428571  23334.54761905  31886.69047619
  22429.1547619   34378.29761905  21536.61904762  29322.45238095
  33552.38095238  25231.05952381  35239.51190476  36094.30952381
  27295.55952381  24256.29761905]
[27900 27000 25200 23400 18000 29700 20700 28800 19800 31500 18900 261
00
 30600 22500 32400 33300 24300 21600]
```

Out[29]:

```
<matplotlib.collections.PathCollection at 0x7fa9baaa57f0>
```



EXERCISE 3c

Having done this, we now need to calculate the regression equation, so we can determine the optimal time for leaving. The graph may look a bit raw at the moment, but that's okay. We will fix that soon!

First, we still need to find the equation, plot the line on the graph, and solve it for 09:00.

To solve the equation, we can use the `stats.linregress(first_list, second_list)` in scipy. Run this now, placing the output into a variable `fx` with the same parameters as you used for the scatter plot, and look at the output:

In [32]:

```
# YOUR CODE HERE
from scipy import stats
#stats는 모듈, scipy는 서브 모듈 또는 함수

fx = stats.linregress(departureValues_X, responseMeanVaues_Y)
#변수 fx를 선언하고 stats.linreqress에 출발값 x와 대응평균값 y 할당

print("slope\t\t", fx.slope)
#기울기 구하기 .slope는 scipy의 기능인 stats의 특성
print("intercept\t", fx.intercept)
#절편출력
#직선 Y = mx+c , c는 절편 m은 기울기



### END ANSWER
```

```
slope           0.975768820109
intercept       -2206.19277639
```

The output of the function is an object with different values. These can be accessed using the dot syntax, i.e., `m = fx.slope`. We can use these to plot `y = f(x)`, by evaluating `mx + c` as the second parameter in the `plot` function. Perform the following steps:

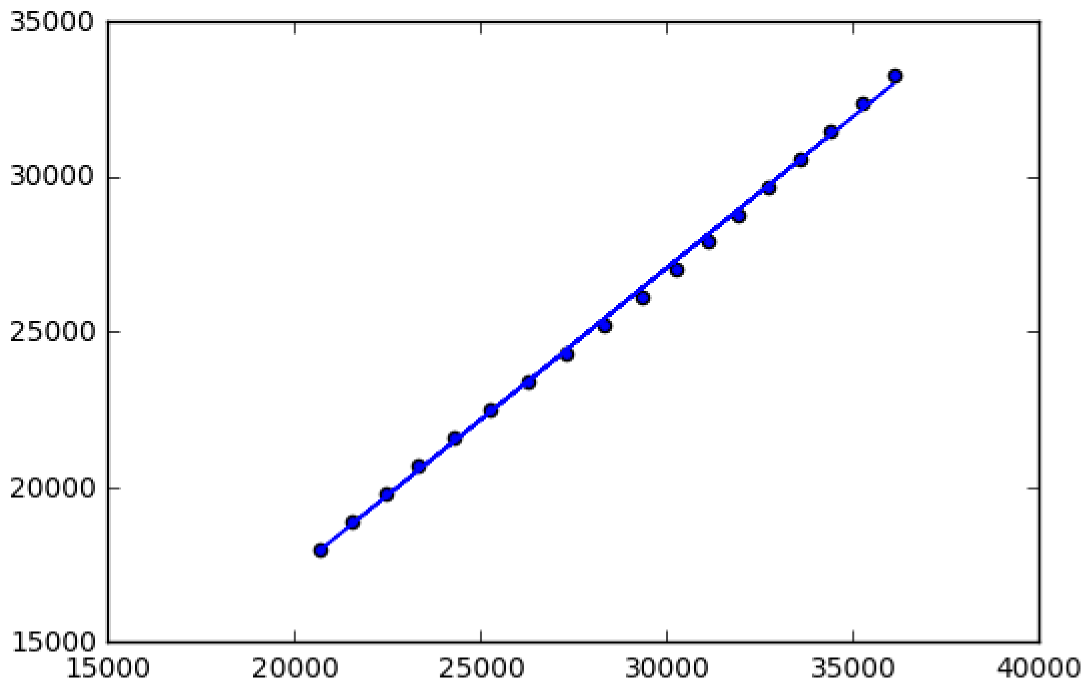
- Create a variable `m` from `fx.slope`
- Create a variable `c` from `fx.intercept`
- Plot this equation onto the graph as `plt.plot(leaving_time, m * x + c, '-')`

In [45]:

```
# YOUR CODE HERE
#방적식 mx+c로 출발시간을 플롯으로 나타낼 것입니다.
m = fx.slope
c = fx.intercept

plt.scatter(departureValues_X, responseMeanVaues_Y)

plt.plot(np.asarray(departureValues_X), m * np.asarray(departureValues_X) + c, '-')
#plt.plot(np.asarray(departureValues_X), m * np.asarray(departureValues_X) + c, '-',
#plt의 특성인 plot의 값을, np.asarray를 사용할거라 다시 선언!
#plt는 위에서 import matplotlib.pyplot as plt %matplotlib inline 이름을 다시 사용한거임

### END ANSWER
```

Out[45]:

```
[<matplotlib.lines.Line2D at 0x7fa9b822bb00>]
```



```
Now we have a scatter plot, a regression equation, and both are displayed on a
graph.  We can use the graph to get an idea for when we should leave, but we can
also solve the equation, and this is our next step.  We already have an equation,
so we can substitute the values in to get the answer.

All we need to do is:

* Get the value for 09:00 in seconds rather than the more general case used for
the plot
* Get the answer from the equationn
* Convert the answer from seconds to a time
```

In [46]:

```
# YOUR CODE HERE
#특정 시간에 도착하려면 언제 출발하는게 제일 좋을지 알아내려고 함
#9시에 도착하려면 언제 출발하는게 좋을 지 최적의 '출발시간'을 초단위로 계산할거임!

y = m * get_time_in_seconds('09:00') + c
#변수 y에 m *get_time_in_seconds + C를 곱해줌

print(get_time_in_str(y))



### END ANSWER

#순서 : 기반 데이터 data,gov.uk에서 다운로드 얻어와서
# 데이터를 정제한 다음, 특정 기간에 해당하는 데이터를 가져오고,
# 그 필요했던 데이터를 이용하여 실제로 출발하기 가장 좋은 시간을 구했다.
```

```
08:10
```

EXERCISE 4 (Optional)

We have an answer, so we could go home. However, we need to show this to our manager, so ideally we'll get the graph to look a bit better. Here are some improvements you could make. Reading documentation about functions is an important skill, so the instructions here are brief and you are expected to read through the documentation at the links. If you have trouble, try StackOverflow (https://stackoverflow.com), or ask one of the demonstrators.

- Label the chart and the axes docs (http://matplotlib.org/users/pyplot_tutorial.html#id3)
- Change the range of the axes using `xlim` and `ylim` docs (http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.xlim)
- Change the axes from seconds to a datetime string. For this we use `xticks` and `yticks` functions, which require an array of labels, e.g., `['07:00', '07:15'.....]` docs (http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.xticks)
- Increase the size of the plot. You will need to specify a `plt.figure` before drawing the plot. This will specify `figsize=(width,height)` as as parameter
- Look at other examples of customisations you can use, for example here (http://chrisalbon.com/python/matplotlib_simple_scatterplot.html)

Another Solution?

There are many ways you could try and solve this problem. We chose linear regression as an example. If you've finished this, try and think of another way to solve it, and implement that instead!

In [ ]: