Lab 1

This is the first lab to let you get familiar with the python environment and basic exploratory data analysis (EDA) methods.

Today you will learn EDA by some case study. Before the case study, we will review some basic knowledge about Python, Numpy, and Pandas.

Why Python for Data Analysis? ¶

Python has powerful libraries for data loading, visualization, statistics, natural language processing, image processing, and more. This vast toolbox provides data scientists with a large array of general- and special-purpose functionality. One of the main advantages of using Python is the ability to interact directly with the code, using a terminal or other tools like the Jupyter Notebook, which we'll look at shortly.

Essential Libraries and Tools

For data analysis and machine learning, array and tabular data are the fundamental data structures we use. NumPy and Pandas are Python libraries designed for processing these data structures.

Numpy

Numpy provide basic operation for array. The core functionality of Numpy is the ndarray class, a multidimensional (n-dimensional) array. All elements of the array must be of the same type. A NumPy array looks like this:

Creating ndarrays

In [1]:

```
import numpy as np
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
arr1
```

Out[1]:

```
array([6. , 7.5, 8. , 0. , 1. ])
```

In [2]:

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)
arr2
```

Out[2]:

```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

arr2 has two dimensions with shape inferred from the data. We can confirm this by inspecting the ndim and shape attributes:

In [3]:
```
arr2.ndim
```

Out[3]:
```
2
```

In [4]:
```
arr2.shape
```

Out[4]:
```
(2, 4)
```

np.array will creat a data type for the data. The data type is stored in a special dtype metadata object;

In [5]:
```
arr1.dtype
```

Out[5]:
```
dtype('float64')
```

In [6]:
```
arr2.dtype
```

Out[6]:
```
dtype('int32')
```

We can change the data type by

In [7]:
```
float_arr = arr2.astype(np.float64)
float_arr.dtype
```

Out[7]:
```
dtype('float64')
```

In [8]:
```
int_arr = arr1.astype(np.int32)
int_arr.dtype
```

Out[8]:
```
dtype('int32')
```

Arithmetic with Numpy Arrays

Arrays are important because they enable you to express batch operations on data without writing any for loops. NumPy users call this vectorization. Any arithmetic operation between equal-size arrays applies the operation element-wise:

In [9]:
```
# Matrix Addition
arr2 + arr2
```

Out[9]:
```
array([[ 2,  4,  6,  8],
       [10, 12, 14, 16]])
```

In [10]:
```
# Matrix Subtraction
arr2 - arr2
```

Out[10]:
```
array([[0, 0, 0, 0],
       [0, 0, 0, 0]])
```

```
In [11]: # Compute the element-wise reciprocal of the NumPy array arr2
         1 / arr2

Out[11]: array([[1.        , 0.5       , 0.33333333, 0.25      ],
                [0.2       , 0.16666667, 0.14285714, 0.125     ]])
```

```
In [12]: # Multiply each element of the NumPy array arr2 by 2
         arr2 * 2

Out[12]: array([[ 2,  4,  6,  8],
                [10, 12, 14, 16]])
```

```
In [13]: # Raise each element of the NumPy array arr2 to the power of 2
         arr2 ** 2

Out[13]: array([[ 1,  4,  9, 16],
                [25, 36, 49, 64]])
```

More detail about numpy overview can be found at https://numpy.org/doc/stable/user/index.html#user (https://numpy.org/doc/stable/user/index.html#user)

Pandas

Pandas is a Python library for data analysis. It is built around a data structure called the DataFrame. A pandas DataFrame is a table, similar to an Excel spreadsheet. pandas provides a great range of methods to modify and operate on this table; in particular, it allows SQL-like queries and joins of tables. In contrast to NumPy, which requires that all entries in an array be of the same type, pandas allows each column to have a separate type (for example, integers, dates, floating-point numbers, and strings). Here is a small example of creating a DataFrame using a dictionary:

```
In [14]: import pandas as pd
         # create a simple dataset of people

         data = {'Name': ["John", "Anna", "Peter", "Linda"],
                 'Location' : ["New York", "Paris", "Berlin", "London"],
                 'Age' : [24, 13, 53, 33]}
         data_pandas = pd.DataFrame(data)
         print(data_pandas)

             Name  Location  Age
         0   John  New York   24
         1   Anna     Paris   13
         2  Peter    Berlin   53
         3  Linda    London   33
```

There are several possible ways to query this table. For example:

In [15]:

```
# Select all rows that have an age column greater than 30
print(data_pandas[data_pandas.Age > 30])
```

```
    Name Location  Age
2  Peter   Berlin   53
3  Linda   London   33
```

Case Study

Now let's start our case study.

Library used

In [16]:

```
import pandas as pd
import numpy as np
import os
```

Reading the data & exploring the data

Reading and basic exploratory data analysis (EDA)

auto.csv: Gas mileage, horsepower, and other information for cars.

In [17]:

```
path = "D:\CityU\SDSC5002\week 2 tutorial" # Replace with your file path.
file_name = "auto.csv"
file_path = os.path.join(path, file_name)
df = pd.read_csv(file_path)
#mpg:miles per gallon
#cylinders:Number of cylinders between 4 and 8
#displacement: Engine displacement (cu. inches)
#horsepower:Engine horsepower
#weight:Vehicle weight (lbs.)
#acceleration:Time to accelerate from 0 to 60 mph (sec.)
#year:Model year
#origin:Origin of car (1. American, 2. European, 3. Japanese)
#name:Vehicle name
```

In [18]:

```
# Let's take a look at the structure of the data for the first 10 rows.
df.head(10)
```

Out[18]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
5	15.0	8	429.0	198	4341	10.0	70	1	ford galaxie 500
6	14.0	8	454.0	220	4354	9.0	70	1	chevrolet impala
7	14.0	8	440.0	215	4312	8.5	70	1	plymouth fury iii
8	14.0	8	455.0	225	4425	10.0	70	1	pontiac catalina
9	15.0	8	390.0	190	3850	8.5	70	1	amc ambassador dpl

In [19]:

```
# Let's do descriptive statistics
df.describe()
```

Out[19]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year
count	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000
mean	23.445918	5.471939	194.411990	104.469388	2977.584184	15.541327	75.979592
std	7.805007	1.705783	104.644004	38.491160	849.402560	2.758864	3.683737
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000
25%	17.000000	4.000000	105.000000	75.000000	2225.250000	13.775000	73.000000
50%	22.750000	4.000000	151.000000	93.500000	2803.500000	15.500000	76.000000
75%	29.000000	8.000000	275.750000	126.000000	3614.750000	17.025000	79.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000

In [20]:

```
# check variable information and missing value information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 392 entries, 0 to 391
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64  
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    int64  
 4   weight        392 non-null    int64  
 5   acceleration  392 non-null    float64
 6   year          392 non-null    int64  
 7   origin        392 non-null    int64  
 8   name          392 non-null    object 
dtypes: float64(3), int64(5), object(1)
memory usage: 27.7+ KB
```

In [21]:

```
# We will learn how to tune the visulization parameters to make publication-ready figur
# calculate different measures of central tendency (mean, median and mode) of Year
print ("Mean:",df['year'].mean()) # df['year'].mean() is the mean of df['Year']
print ("Median",df['year'].median())  # df['year'].median() is the median of df['Year']
print ("Mode",df['year'].mode()) # df['year'].mode() is the mode of df['Year'], the mos
```

```
Mean: 75.9795918367347
Median 76.0
Mode 0    73
Name: year, dtype: int64
```

In [22]:

```
# calculate different measures of dispersion or variability of year (range, IQR, varian
print("The max, min and range of Age are ",df['year'].max(),df['year'].min(),df['year']
```

```
The max, min and range of Age are  82 70 12 , respectively
```

In [23]:

```
# calculate 75% percentile(Q3) and 25% percentile(Q1) quantile and the interquartile ra
df['year'].quantile([.25,0.75])
```

Out[23]:

```
0.25    73.0
0.75    79.0
Name: year, dtype: float64
```

In [24]:

```
print ("The variance of Year is: %.1f"%(df['year'].var()))
print ("The standard deviation of Year is: %.1f"%(df['year'].std()))
```

```
The variance of Year is: 13.6
The standard deviation of Year is: 3.7
```

In [25]:

```
# Count the frequency of each Year.
pd.value_counts(df['year'])
# the first column is the age and the second column is the count
```

Out[25]:

```
year
73    40
78    36
76    34
75    30
82    30
70    29
79    29
72    28
77    28
81    28
71    27
80    27
74    26
Name: count, dtype: int64
```