Python, Jupyter, Pandas

Notebook adapted from HKUST's VISLAB COMP4462 Lab Notebook

In this tutorial, we will learn about the basics of Python through an interactive programming environment (REPL, read-evaluate-print-loop), which is this instantiation of a Jupyter notebook you are looking at.

Jupyter notebook is widely used in the data science community. Google provides Colab, an installation-free and widely accessible version of Jupyter notebook. To save troubles of setting up Jupyter notebook, you can view this notebook on Google Colab.

All the original materials of the HKUST tutorial are hosted on this GitHub repository.

Python and Jupyter Basics

First Jupyter shortcut: shift+enter

Press shift+enter to run a "cell", a cell is a code block that runs separately from the other code blocks. The first time you run a cell on Google Colab, it will take a few seconds to create a runtime for you to run code on the remote server, which has separate computational power, RAM and storage from your laptop or desktop.

Sometimes, a code block contains only one statement, sometimes multiple statements, that"s why we call them "blocks". And the content is not necessarily Python statements, it can be shell commands as well, as we will see, keep reading!

Print

```
# Print

print('Hello, welcome to VDS!')
```

```
Hello, welcome to VDS!
```

```
################################################################################
# TODO:                                                                        #
# Print your own name below!                                                   #
################################################################################
```

```
print('Benito Alvares')

################################################################################
#                                END OF YOUR CODE                              #
################################################################################
```

```
Benito Alvares
```

The Python exercises below are optional (you're welcome to do them, but they are not mandatory). Go ahead and scroll to Pandas below.

Basic Data Types

You have seen a string above when trying to print "Hello". We will see more data types:

- String, e.g. "Hello, world!"
- Numeric, e.g. 4462
- Booleans, e.g. True, False
- None, None is the null/undefined in Python
- Parsing string to numberic, e.g. int("123"), float("123.45")

Declaring a variable in Python is just assigning a value to a new variable; it will automatically detect whether the variable is declared; if not, create a new variable with the value assigned to it; otherwise, update the value.

```
# Variables and arithmetic

pi = 3.1415
degree = 60
radian = degree * pi / 180
```

If the last statement of a cell is an expression (i.e. not an assignment statement), Jupyter notebook will print the evaluated value of the expression automatically.

Like printing the `radian` variable is to just write `radian`

```
radian
```

```
1.0471666666666668
```

```
# Or any expressions

radian * 180 / pi
```

```
60.00000000000001
```

```
################################################################################
# TODO:                                                                        #
# Try to calculate the radian of 45 degrees!                                   #
################################################################################

radian =45 * pi / 180
radian

################################################################################
#                                END OF YOUR CODE                              #
################################################################################
```

```
0.785375
```

List, tuple and dictionary

List

For simplicity, you can treat list in Python as array in other languages. In Python, the use of list is much more common than array.

```
# List

visual_channels = ['position', 'length', 'angle', 'area', 'color', 'curvature']
visual_channels
```

```
['position', 'length', 'angle', 'area', 'color', 'curvature']
```

```
# Accessing items in a list is just like accessing an array with square
# brackets []

'Most effective channel: ' + visual_channels[0]
```

```
'Most effective channel: position'
```

```
# To get a slice of a list

visual_channels[1:3]
```

```
['length', 'angle']
```

```
# To append an item to a list, we use the "append" method
```

```
visual_channels.append('shape')
visual_channels
```

```
['position', 'length', 'angle', 'area', 'color', 'curvature', 'shape']
```

```
# To concatenate two list, just add them up. Remember to assign the sum to a new
# variable

more_visual_channels = visual_channels + ['color hue', 'color saturation', 'color luminance']
more_visual_channels
```

```
['position',
 'length',
 'angle',
 'area',
 'color',
 'curvature',
 'shape',
 'color hue',
 'color saturation',
 'color luminance']
```

```
# To sort a list, we use "sort"

more_visual_channels.sort()
more_visual_channels
```

```
['angle',
 'area',
 'color',
 'color hue',
 'color luminance',
 'color saturation',
 'curvature',
 'length',
 'position',
 'shape']
```

```
# To remove an item from a list, we use "remove"

visual_channels.remove('length')
visual_channels
```

```
['position', 'angle', 'area', 'color', 'curvature', 'shape']
```

```
# To remove an item by its index

del visual_channels[2]
visual_channels
```

```
['position', 'angle', 'curvature', 'shape']
```

```
################################################################################
# TODO:                                                                        #
# Try to create two lists of colors, then concatenate them, and sort in        #
# alphabetical order.                                                          #
# The two lists are                                                            #
# red, green, blue                                                             #
# cyan, magenta, yellow, black                                                  #
################################################################################
list1 = ['red', 'green', 'blue']
list2 = ['cyan', 'yellow', 'magenta','black']

combinedlist = list1 + list2

combinedlist.sort()

combinedlist
################################################################################
#                                 END OF YOUR CODE                             #
################################################################################
```

```
['black', 'blue', 'cyan', 'green', 'magenta', 'red', 'yellow']
```

Tuple

```
# Tuple

tutorial = ('Python', 'Jupyter notebook', ['Pandas', 'altair'])
tutorial
```

```
('Python', 'Jupyter notebook', ['Pandas', 'altair'])
```

```
# Unpacking tuple
# It is like assigning values to variable position by position, Be careful that
# the number of items in the tuple needs to match the number of variables

language, tool, libraries = tutorial
print('Programming language: ', language)
print('Running on: ', tool)
print(f'Libraries: {libraries[0]}, {libraries[1]}')
```

```
Programming language:  Python
Running on:  Jupyter notebook
Libraries: Pandas, altair
```

Dictionary

```
# Dictionary

types_of_data = {
    'ordered': [1, 2, 3, 4, 5],
    'categorical': ['strawberry', 'apple', 'orange']
}
types_of_data
```

```
{'categorical': ['strawberry', 'apple', 'orange'], 'ordered': [1, 2, 3, 4, 5]}
```

```
# Accessing entries in a dictionary by keys. It is like accessing an array with
# indexes other than integers

types_of_data['categorical']
```

```
['strawberry', 'apple', 'orange']
```

```
# Modifying an entry in a dictionary can be simply done by assigning a value

types_of_data['categorical'] = ['strawberry', 'apple', 'orange', 'banana']


################################################################################
# TODO:                                                                        #
# Try to modify the values of the "ordered" entry to 1, 10, 100, 1000!         #
################################################################################
types_of_data['ordered'] = [1, 10, 100, 1000]

types_of_data
################################################################################
#                                END OF YOUR CODE                              #
################################################################################
```

```
{'categorical': ['strawberry', 'apple', 'orange', 'banana'],
 'ordered': [1, 10, 100, 1000]}
```

Function

```
# Define a function

def int2hex (n):
  return ('0x%0.2X' % n)[2:]

int2hex(255)
```

```
'FF'
```

```
# Define a function with named arguments

def rgb2hex (red=0, green=0, blue=0):
  return '#' + int2hex(red) + int2hex(green) + int2hex(blue)

print(rgb2hex(64,224,208))
print(rgb2hex(green=255))
```

```
#40E0D0
#00FF00
```

```
################################################################################
# TODO:                                                                        #
# Try to create a function that returns hex code in tuple!                     #
# Replace the "..." with your code.                                            #
# Get the hex code of RGB 64, 224, 208 and unpack it in variables r, g and b   #
################################################################################

def rgb2hex_tuple (R, G, B):
  return (int2hex(R), int2hex(G), int2hex(B))

R, G, B = rgb2hex_tuple(64, 224, 208)

print(R,G,B)
################################################################################
#                                END OF YOUR CODE                              #
################################################################################
```

```
40 E0 D0
```

Logic flow and loop

```
# If statement

color = 'cyan'
hex = ''

if color is 'cyan':
  hex = rgb2hex(0, 255, 255)
elif color is 'magenta':
  hex = rgb2hex(255, 0, 255)
elif color is 'yellow':
  hex = rgb2hex(255, 255, 0)
else:
  hex = rgb2hex(255, 255, 255)

hex
```

```
    '#00FFFF'
# Loop through items in a list

cmyk_colors = ['cyan', 'magenta', 'yellow', 'black']

for color in cmyk_colors:
  print(color)
```

```
cyan
magenta
yellow
black
```

```
# Loop through items in a list by index

for i in range(len(cmyk_colors)):
  print(i, cmyk_colors[i])
```

```
0 cyan
1 magenta
2 yellow
3 black
```

```
################################################################################
# TODO:                                                                        #
# Try to print the hex code of all the CMYK colors through using a for loop.   #
# RGB values of CMYK colors:                                                   #
# Cyan:    (0, 255, 255)                                                       #
# Magenta: (255, 0, 255)                                                       #
# Yellow:  (255, 255, 0)                                                       #
# Black:   (255, 255, 255)                                                     #
################################################################################

for color in cmyk_colors:
  if color is 'cyan':
    hex = rgb2hex(0, 255, 255)
  elif color is 'magenta':
    hex = rgb2hex(255, 0, 255)
  elif color is 'yellow':
    hex = rgb2hex(255, 255, 0)
  else:
    hex = rgb2hex(255, 255, 255)
  print(color, hex)

################################################################################
#                                END OF YOUR CODE                              #
################################################################################
```

```
    cyan #00FFFF
    magenta #FF00FF
```

List/dictionary comprehension

```
# Make a list by looping through another list
# *color is spread operator, equivalent to rgb2hex(color[0], color[1], color[2])

colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255)]

[rgb2hex(*color) for color in colors]
```

```
['#FF0000', '#00FF00', '#0000FF']
```

```
# Make a dictionary by looping through another list

color_names = ['red', 'green', 'blue']

color_dictionary = {color_names[i]:rgb2hex(*colors[i]) for i in range(len(colors))}

color_dictionary['red']
```

```
'#FF0000'
```

```
################################################################################
# TODO:                                                                        #
# Try to make a color dictionary for CMYK colors.                              #
# RGB values of CMYK colors:                                                   #
# Cyan:    (0, 255, 255)                                                       #
# Magenta: (255, 0, 255)                                                       #
# Yellow:  (255, 255, 0)                                                       #
# Black:   (255, 255, 255)                                                     #
################################################################################

colors_cmyk = [(0, 255, 255), (255, 0, 255), (255, 255, 0),(255, 255, 255) ]

color_dictionary_cmyk = {cmyk_colors[i]:rgb2hex(*colors_cmyk[i]) for i in range(len(colors_cm
color_dictionary_cmyk
################################################################################
#                                 END OF YOUR CODE                             #
################################################################################
```

```
{'black': '#FFFFFF',
 'cyan': '#00FFFF',
 'magenta': '#FF00FF',
 'yellow': '#FFFF00'}
```

Files

```
with open('colors.csv', 'w') as f:
  f.write('color,r,g,b,hex\n')
  f.write('red,255,0,0,#FF0000\n')
  f.write('green,0,255,0,#00FF00\n')
  f.write('blue,0,0,255,#0000FF\n')


with open('colors.csv', 'r') as f:
  lines = f.readlines()
  for line in lines:
    print(line.split(',')[0])
```

```
color
red
green
blue
```

Date and Time

Date and time are very common in data visualization. In Python, the standard library provides rich set of tools to manipulate date and time.

```
# Import datetime library

from datetime import datetime, timedelta


# Get the current date and time (in UTC time)

now = datetime.now()
now
```

```
datetime.datetime(2020, 9, 12, 19, 4, 39, 389165)
```

```
# Access properties of the datetime object

now.year, now.month, now.day, now.hour, now.minute, now.second
```

```
(2020, 9, 12, 19, 4, 39)
```

```
# Create a datetime object with a specific date

project_presentation = datetime(2019, 10, 3)
project_presentation
```

```
datetime.datetime(2019, 10, 3, 0, 0)
```

```
# Add two days to a datetime object

project_presentation + timedelta(days=2)
```

```
datetime.datetime(2019, 10, 5, 0, 0)
```

```
################################################################################
# TODO:                                                                        #
# Try to create a datetime object with date 27th October, 2019!                 #
# Then add two days to it.                                                     #
################################################################################

test = datetime(2019, 10, 27) + timedelta(days=2)
test
################################################################################
#                                END OF YOUR CODE                              #
################################################################################
```

```
datetime.datetime(2019, 10, 29, 0, 0)
```

Pandas Basics

New Section

Upload dataset

In the following, we will explore the Pokemon dataset, you can download the dataset from the tutorial material repository on GitHub (link is occasionally down; if so, use the kaggle link in the next sentence). Thanks to Rounak Banik for preparing The Complete Pokemon Dataset.

If you are using Google Colab, you should be able to see an arrow on the left edge of the browser window, it hides the Table-of-conents; Code-snippets; Files panel. Open it and switch to the "Files" tab, where you can see the files in the current directory. Click upload and upload the dataset you have downloaded.

Pandas Dataframe

```
# Import pandas, and assign it to its common alias "pd"

import pandas as pd
```

```
df_pokemon = pd.read_csv('pokemon.csv')
```

```
df_pokemon.head()
```

	pokedex_number	name	japanese_name	base_total	attack	defense	sp_attack	sp
0	1	Bulbasaur	Fushigidaneフシギダネ	318	49	49	65	
1	2	Ivysaur	Fushigisouフシギソウ	405	62	63	80	
2	3	Venusaur	Fushigibanaフシギバナ	625	100	123	122	
3	4	Charmander	Hitokageヒトカ	309	52	43	60	

```
df_pokemon.describe()
```

	pokedex_number	base_total	attack	defense	sp_attack	sp_defense	s
count	801.000000	801.000000	801.000000	801.000000	801.000000	801.000000	801.00
mean	401.000000	428.377029	77.857678	73.008739	71.305868	70.911361	66.33
std	231.373075	119.203577	32.158820	30.769159	32.353826	27.942501	28.90
min	1.000000	180.000000	5.000000	5.000000	10.000000	20.000000	5.00
25%	201.000000	320.000000	55.000000	50.000000	45.000000	50.000000	45.00
50%	401.000000	435.000000	75.000000	70.000000	65.000000	66.000000	65.00
75%	601.000000	505.000000	100.000000	90.000000	91.000000	90.000000	85.00
max	801.000000	780.000000	185.000000	230.000000	194.000000	230.000000	180.00

```
df_pokemon.sort_values('hp', ascending=False).head()
```

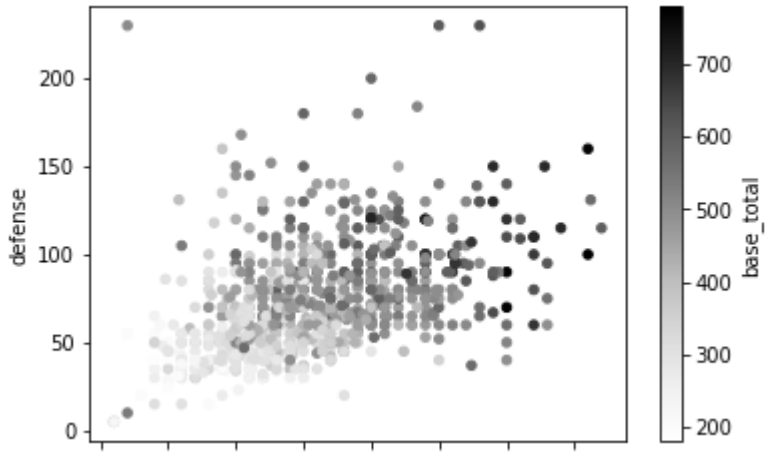
	pokedex_number	name	japanese_name	base_total	attack	defense	sp_attack	s
241	242	Blissey	Happinasハピナス	540	10	10	75	
112	113	Chansey	Luckyラッキー	450	5	5	35	
798	799	Guzzlord	Akuzikingアクジキング	570	101	53	97	
717	718	Zygarde	Zygarde (10% Forme)ジガルデ	708	100	121	91	

Plot with Pandas

Hint: Try to press "Tab" key right after you have typed the open parenthesis, e.g. `df_pokemon.plot.scatter(` . It will show the function signature, so that you can see what arguments to pass into the function.

```
df_pokemon.plot.scatter(x='attack', y='defense', c='base_total')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f914b05e358>
```



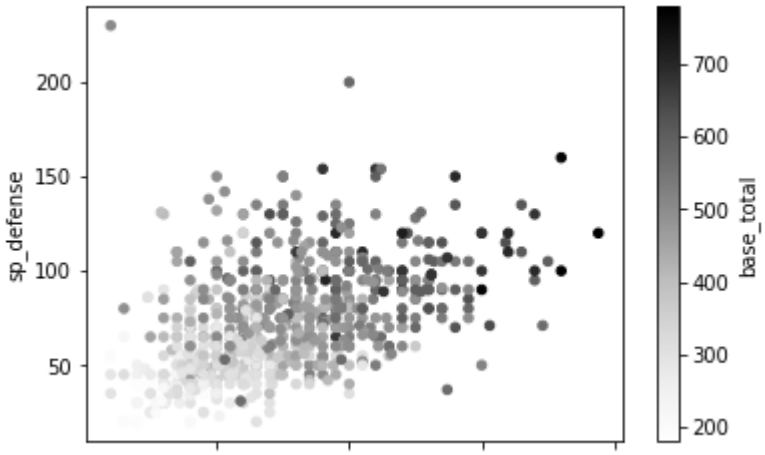

```
################################################################################
# TODO:                                                                        #
# Try to use sorting to find the 5 lowest defense pokemons.                    #
################################################################################

df_pokemon.sort_values('defense', ascending=True).head()

################################################################################
#                                END OF YOUR CODE                              #
################################################################################
```

	pokedex_number	name	japanese_name	base_total	attack	defense	sp_attack
439	440	Happiny	Pinpukuピンプク	220	5	5	15
112	113	Chansey	Luckyラッキー	450	5	5	35
241	242	Blissey	Happinasハピナス	540	10	10	75
237	238	Smoochum	Muchulムチュール	305	30	15	85

```
################################################################################
# TODO:                                                                        #
# Try to plot the relationship between sp_attack and sp_defense.               #
################################################################################
```

```
df_pokemon.plot.scatter(x='sp_attack', y='sp_defense', c='base_total')

################################################################################
#                                END OF YOUR CODE                              #
################################################################################
```

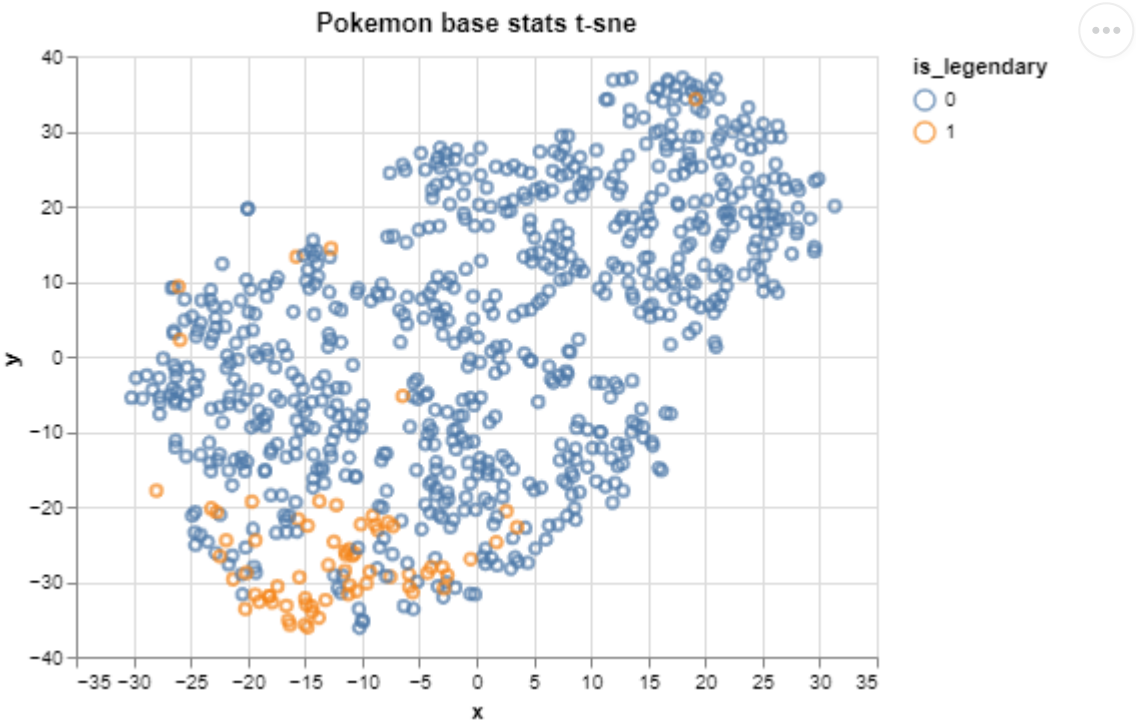
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f914ab389e8>
```



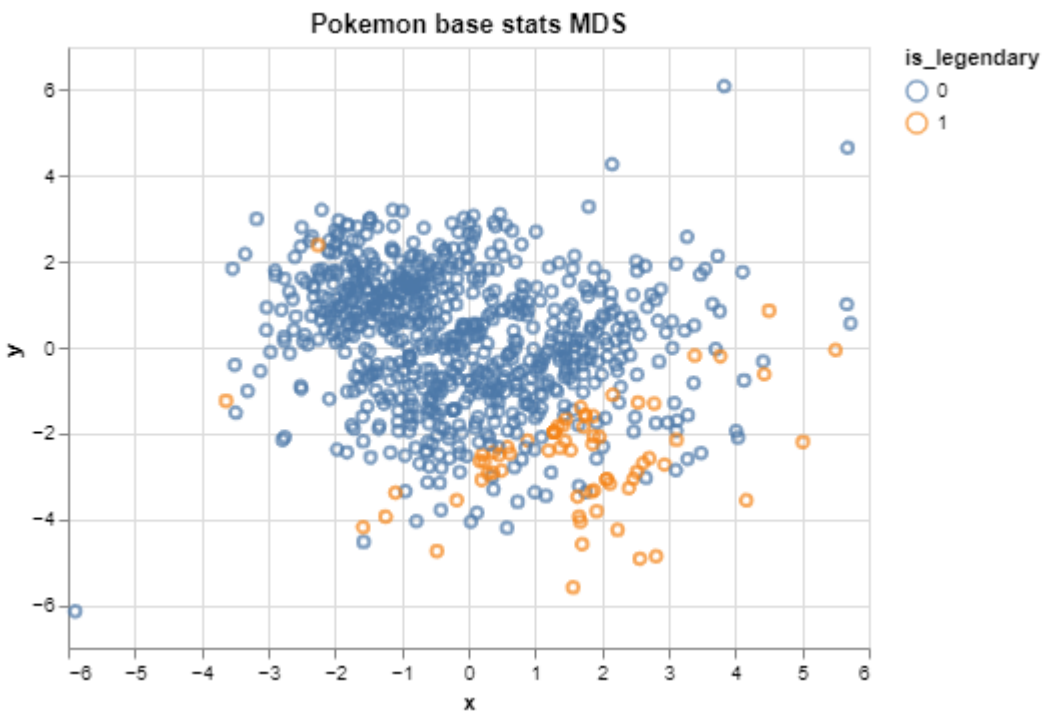
Dimension reduction with scikit-learn

```
from sklearn.manifold import TSNE, MDS
import altair as alt


# Extract values from dataframe

X = df_pokemon[['attack', 'defense', 'sp_attack', 'sp_defense', 'speed', 'hp']].values


# Normalize

X = (X - X.mean())/ X.std()


# t-sne algorithm,
# t-sne is a randomized algorithm, the argument random_state is the random seed,
# it is set to 801 for reproducibile plotting, you may change the seed and see
# how the plot changes

X_embedded = TSNE(n_components=2, random_state=150).fit_transform(X)


# Concat results to dataframe

df_pokemon['x'] = X_embedded[:, 0]
df_pokemon['y'] = X_embedded[:, 1]
```

```
# Plot with altair

alt.Chart(df_pokemon, title='Pokemon base stats t-sne').mark_point().encode(
    x='x',
    y='y',
    color='is_legendary:N',
    tooltip=['name', 'attack', 'defense', 'sp_attack', 'sp_defense', 'speed', 'hp', 'is_legen
)
```




```
##########################################################################################
# TODO:                                                                                  #
# Try to perform dimension reduction with MDS and plot the result!                       #
##########################################################################################

MyX_embedded = MDS(n_components=2, random_state=150).fit_transform(X)
df_pokemon['x'] = MyX_embedded[:, 0]
df_pokemon['y'] = MyX_embedded[:, 1]
alt.Chart(df_pokemon, title='Pokemon base stats MDS').mark_point().encode(
    x='x',
    y='y',
    color='is_legendary:N',
    tooltip=['name', 'attack', 'defense', 'sp_attack', 'sp_defense', 'speed', 'hp', 'is_legen
)


##########################################################################################
#                                    END OF YOUR CODE                                    #
##########################################################################################
```


Pokemon base stats MDS
is_legendary
0
1
y
x
6
4
2
0
−2
−4
−6
−6
−5
−4
−3
−2
−1
0
1
2
3
4
5
6