Introduction to Scientific Programming in Python

WhoAmI?

Ayse Bilge Gunduz

PhDc, Comp Eng @ YTU

Women Developers of Turkey

Women Hackerz

Woment Tech Network

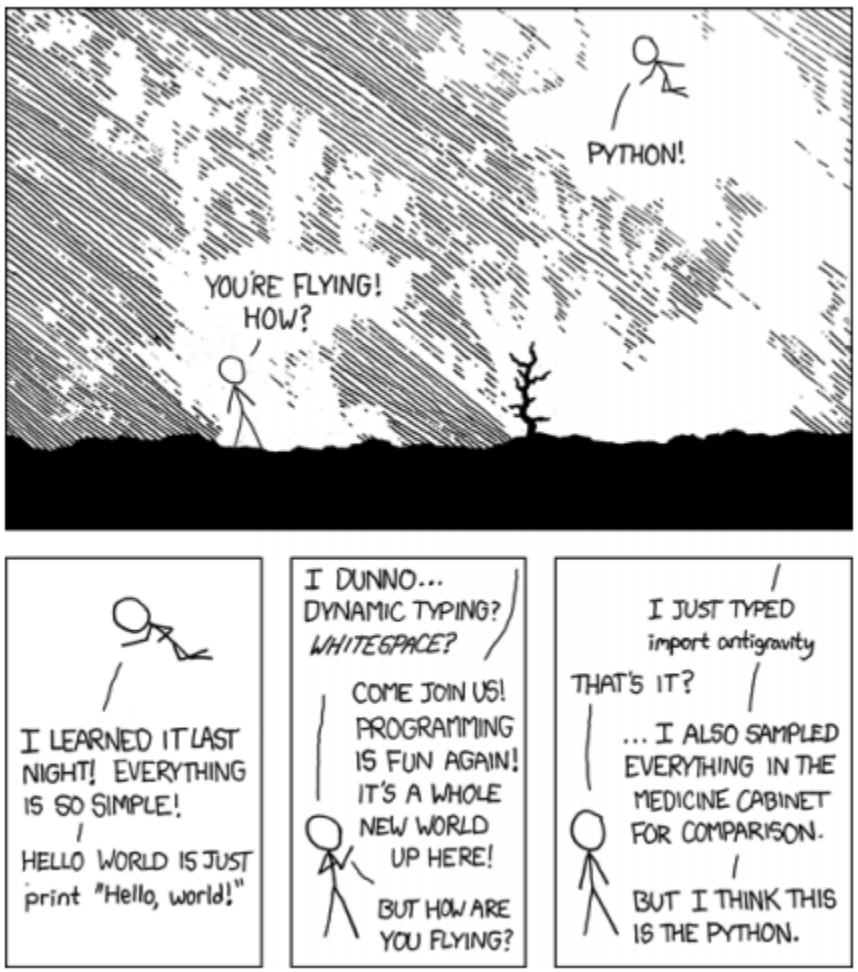
Application Security and Data Science Enthusiast

Muay Thai 🥊

@abilgegunduz

@aysebilgegunduz

Python



https://web.stanford.edu/~schmit/cme193/lec/lec1.pdf

Why Python3

Python 2.7 is end of life, and will not be maintained past January 1, 2020.

Working with Python 2.7 is a risk that you won't find sufficient support sooner or later.

NumPy

```
1 import numpy as np #importing let us use the lib
```

```
1 x=7
2
3 print(np.exp(x)) #1096.6331584284585
4 print(np.abs(x)) #7
5 print(np.pi)     #3.141592653589793
6 print(np.e)      #2.718281828459045
```

Output

```
1096.6331584284585
7
3.141592653589793
2.718281828459045
```

https://github.com/numpy/numpy

NumPy

Input

```
1 A = np.array([0, 1, 2, 3], dtype=np.float)  #create array
2 print(A)
3
4 print(A.ndim)   # number of dimensions, in Matlab `ndims(a)`
5 print(A.shape)  # shape, in Matlab `size(a)`
6 print(A.dtype)  # the data type of the array
```

Output

```
[0. 1. 2. 3.]
1
(4,)
float64
```

NumPy

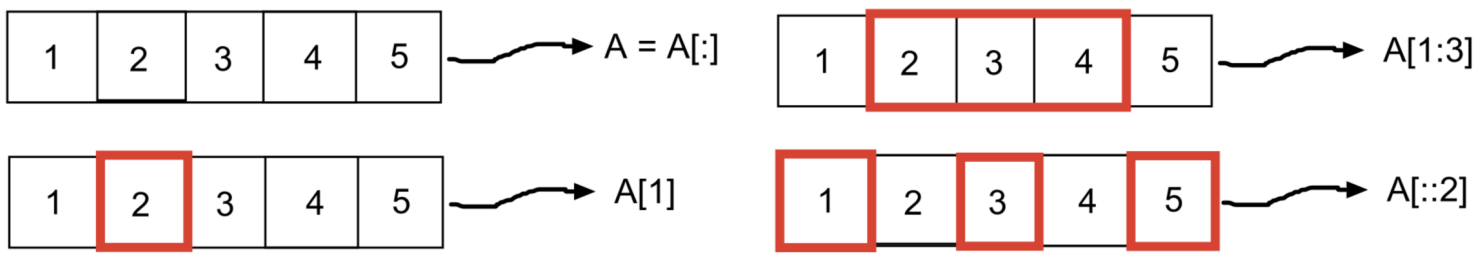
Input

```
1 ################### ARANGE ############################
2 #create a range
3 x1 = np.arange(0,10,1) #arguments:start,stop,step
4 x2 = np.arange(-1,1,0.1)
5 print(x1)
6 print("----------------------")
7 print(x2)
```

Output

```
[0 1 2 3 4 5 6 7 8 9]
----------------------
[-1.00000000e+00 -9.00000000e-01 -8.00000000e-01 -7.00000000e-01
 -6.00000000e-01 -5.00000000e-01 -4.00000000e-01 -3.00000000e-01
 -2.00000000e-01 -1.00000000e-01 -2.22044605e-16  1.00000000e-01
  2.00000000e-01  3.00000000e-01  4.00000000e-01  5.00000000e-01
  6.00000000e-01  7.00000000e-01  8.00000000e-01  9.00000000e-01]
```

NumPy - Vector



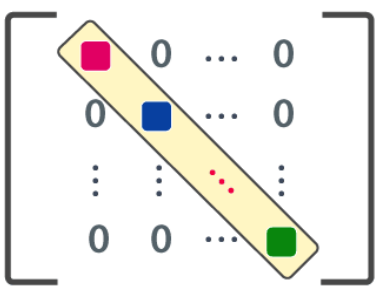
NumPy - Array

```
1 d1 = np.diag([1,2,3])  # a diagonal matrix
2 d2 = np.diag([1,2,3], k=1) #diagonal with offset from the main diagonal
3
4 m1 = np.zeros((5,4))
5 m2 = np.ones((2,3))
6
7 print("m1 : \n", m1)
8 print("----------------------")
9 print("m2 : \n", m2)
```

https://byjus.com/maths/diagonal-matrix/

NumPy - Array

```
d1 = np.diag([1,2,3])  # a diagonal matrix
d2 = np.diag([1,2,3], k=1) #diagonal with offset from the main diagonal

m1 = np.zeros((5,4))
m2 = np.ones((2,3))

print("m1 : \n", m1)
print("----------------------")
print("m2 : \n", m2)
```



https://byjus.com/maths/diagonal-matrix/

NumPy - Array

```
1 d1 = np.diag([1,2,3])  # a diagonal matrix
2 d2 = np.diag([1,2,3], k=1) #diagonal with offset from the main diagonal
3
4 m1 = np.zeros((5,4))
5 m2 = np.ones((2,3))
6
7 print("m1 : \n", m1)
8 print("----------------------")
9 print("m2 : \n", m2)
```

```
d1 :
 [[1 0 0]
 [0 2 0]
 [0 0 3]]
d2 :
 [[0 1 0 0]
 [0 0 2 0]
 [0 0 0 3]
 [0 0 0 0]]
```

https://byjus.com/maths/diagonal-matrix/

NumPy - Array

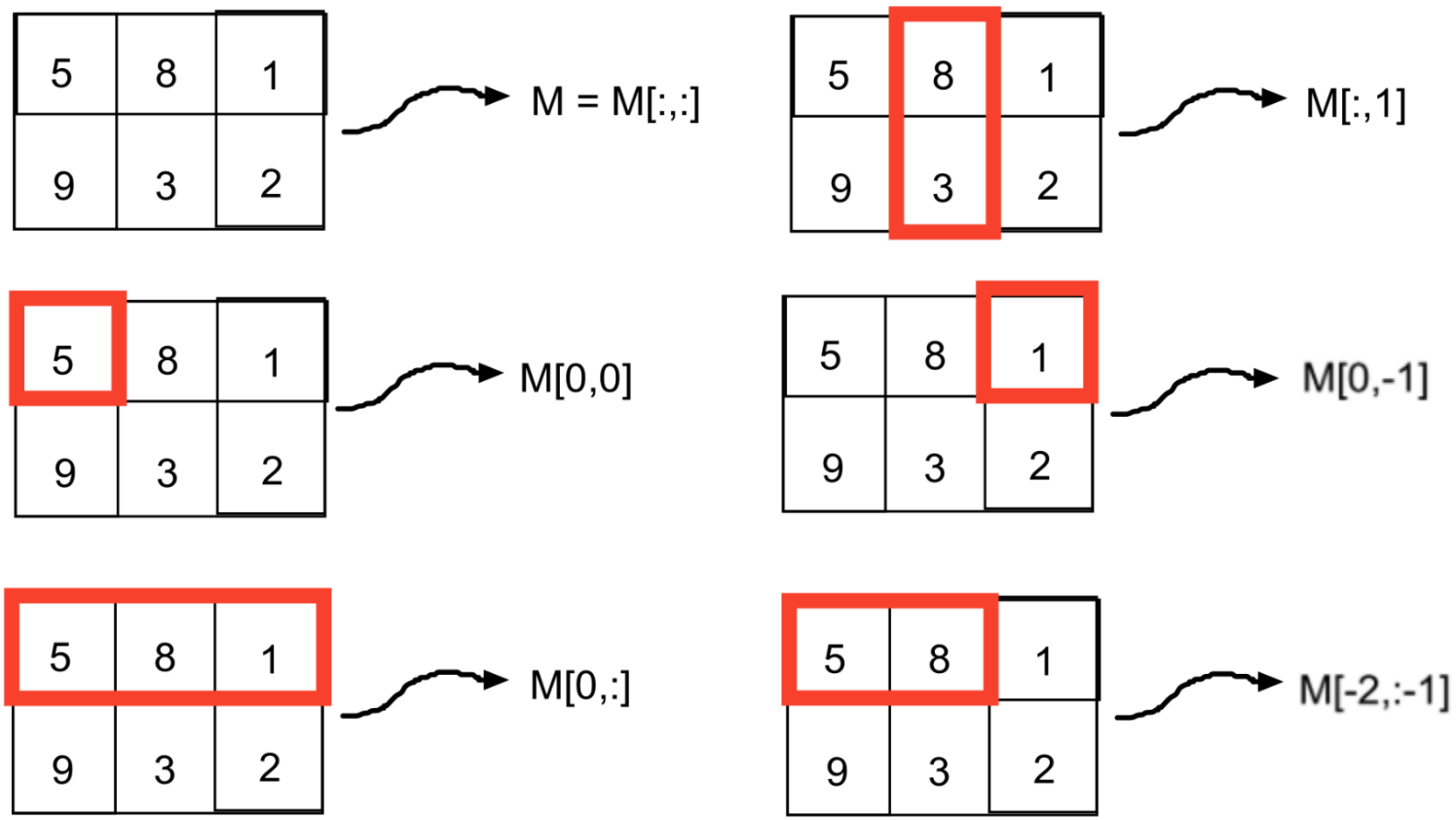
```
1 d1 = np.diag([1,2,3])  # a diagonal matrix
2 d2 = np.diag([1,2,3], k=1) #diagonal with offset from the main diagonal
3
4 m1 = np.zeros((5,4))
5 m2 = np.ones((2,3))
6
7 print("m1 : \n", m1)
8 print("----------------------")
9 print("m2 : \n", m2)
```

```
d1 :
 [[1 0 0]
 [0 2 0]
 [0 0 3]]
d2 :
 [[0 1 0 0]
 [0 0 2 0]
 [0 0 0 3]
 [0 0 0 0]]
```

```
m1 :
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
----------------------
m2 :
 [[1. 1. 1.]
 [1. 1. 1.]]
```

https://byjus.com/maths/diagonal-matrix/

NumPy - Array


5 8 1
9 3 2
M = M[:,:]
5 8 1
9 3 2
M[:,1]
5 8 1
9 3 2
M[0,0]
5 8 1
9 3 2
M[0,-1]
5 8 1
9 3 2
M[0,:]
5 8 1
9 3 2
M[-2,:-1]

NumPy

```
1  ############### FANCY INDEXING ####################
2
3  A = np.array([[n+m*2 for n in range(5)] for m in range(5)])
4  print(A)
5  row_indices = [1, 2, 3]
6  print(A[row_indices])
7  col_indices = [1, 2, -1]
8  print(A[row_indices,col_indices])
9  #
10 print("-"*20)
11 which = [1, 0, 1, 0,2]
12 choices = [[-2,-2,-2,3,8], [5,5,4,5,8],[9,9,9,9,9]]
13 # # #Constructs an array by picking elements from several arrays
14 print(np.choose(which, choices))
```

```
The Matrix A:
 [[ 0  1  2  3  4]
 [ 2  3  4  5  6]
 [ 4  5  6  7  8]
 [ 6  7  8  9 10]
 [ 8  9 10 11 12]]
Selected rows of A:
 [[ 2  3  4  5  6]
 [ 4  5  6  7  8]
 [ 6  7  8  9 10]]
Selected indices of A:  [ 3  6 10]
numpy choose():  [ 5 -2  4  3  9]
```

NumPy

```
1  ################ FANCY INDEXING ######################
2
3  A = np.array([[n+m*2 for n in range(5)] for m in range(5)])
4  print(A)
5  row_indices = [1, 2, 3]
6  print(A[row_indices])
7  col_indices = [1, 2, -1]
8  print(A[row_indices,col_indices])
9  #
10 print("-"*20)
11 which = [1, 0, 1, 0,2]
12 choices = [[-2,-2,-2,3,8], [5,5,4,5,8],[9,9,9,9,9]]
13 # # #Constructs an array by picking elements from several arrays
14 print(np.choose(which, choices))
```

```
The Matrix A:
 [[ 0  1  2  3  4]
 [ 2  3  4  5  6]
 [ 4  5  6  7  8]
 [ 6  7  8  9 10]
 [ 8  9 10 11 12]]
Selected rows of A:
 [[ 2  3  4  5  6]
 [ 4  5  6  7  8]
 [ 6  7  8  9 10]]
Selected indices of A:  [ 3  6 10]
numpy choose():  [ 5 -2  4  3  9]
```

NumPy

```
A = np.array([[n+m*10 for n in range(3)] for m in range(3)])
print("2-d Array\n A:\n",A)

v1 = np.arange(0, 3)
print("Vector:\n", v1)
print("-"*25)
print("Matrix Multiplication:")
print("A*A: \n",np.dot(A, A)) #matrix multiplication using array
print("A*V:\n",np.dot(A, v1))  #matrix multiplication using array and vector
```

```
2-d Array
 A:
 [[ 0  1  2]
 [10 11 12]
 [20 21 22]]
Vector:
 [0 1 2]
-------------------------
Matrix Multiplication:
A*A:
 [[ 50  53  56]
 [350 383 416]
 [650 713 776]]
A*V:
 [ 5 35 65]
```

https://numpy.org/doc/stable/reference/generated/numpy.matrix.html

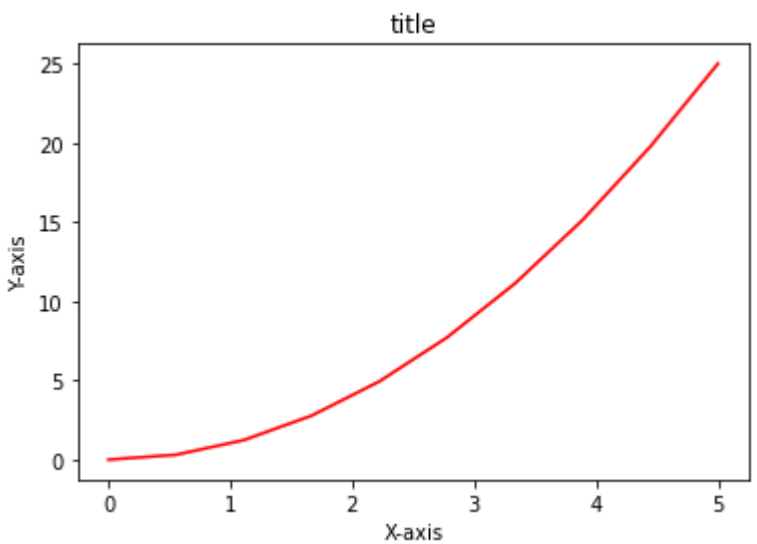
NumPy - Matrix Manipulation

```
1 A = np.array([[n+m*10 for n in range(5)] for m in range(5)])
2 # #
3 n, m = A.shape
4 print("Shape of the Array:\n{0}x{1}".format(n,m))
5 #
6 print("N-dimensional Array A:\n",A)
7 #flattenning
8 C = A.flatten()
9 print("Flatten to a Vector: \n", C)
```

```
Shape of the Array:
5x5
N-dimensional Array A:
 [[ 0  1  2  3  4]
 [10 11 12 13 14]
 [20 21 22 23 24]
 [30 31 32 33 34]
 [40 41 42 43 44]]
Flatten to a Vector:
 [ 0  1  2  3  4 10 11 12 13 14 20 21 22 23 24 30 31 32 33 34 40 41 42 43
 44]
```

Matplotlib

```
1  import matplotlib.pyplot as plt
2
3  x = np.linspace(0, 5, 10) #Returns num evenly spaced samples
4  y = x ** 2
5
6  plt.figure()
7  plt.plot(x, y, 'r') #plot(coord_x, coord_y, color)
8  plt.xlabel('X-axis')
9  plt.ylabel('Y-axis')
10 plt.title('title')
11 plt.show()
```



Seaborn

```
1 import seaborn as sns
2 sns.set()
3
4 iris = sns.load_dataset("iris")
5
6 sns.relplot(x="species", y="sepal_width", data=iris);
```

Seaborn

```python
import seaborn as sns
sns.set()

iris = sns.load_dataset("iris")

sns.relplot(x="species", y="sepal_width", data=iris);
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
..	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

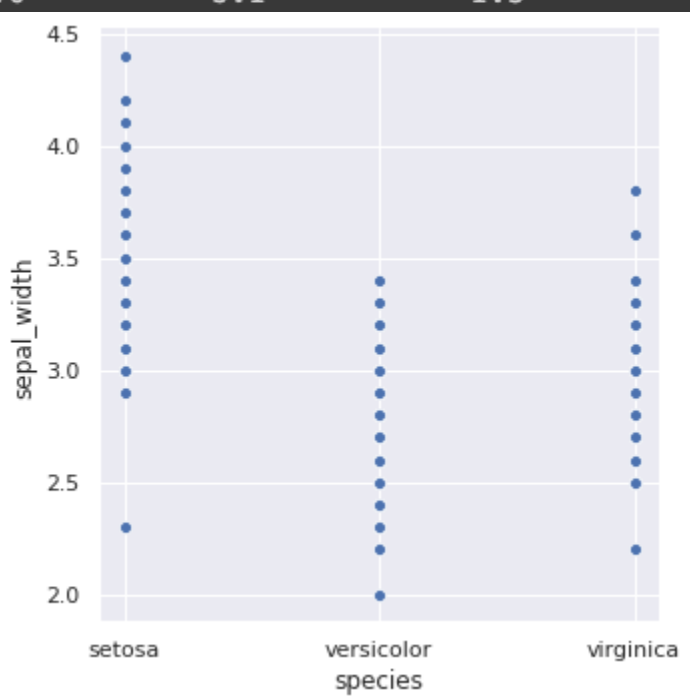
[150 rows x 5 columns]

Seaborn

```python
import seaborn as sns
sns.set()

iris = sns.load_dataset("iris")

sns.relplot(x="species", y="sepal_width", data=iris);
```

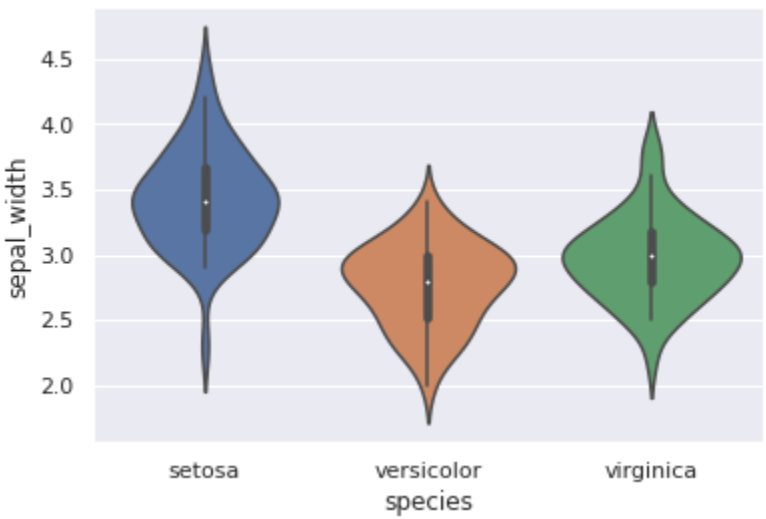
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.			0.2	setosa
..	..			...	...
145	6.			2.3	virginica
146	6.			1.9	virginica
147	6.			2.0	virginica
148	6.			2.3	virginica
149	5.			1.8	virginica

[150 rows x 5 co



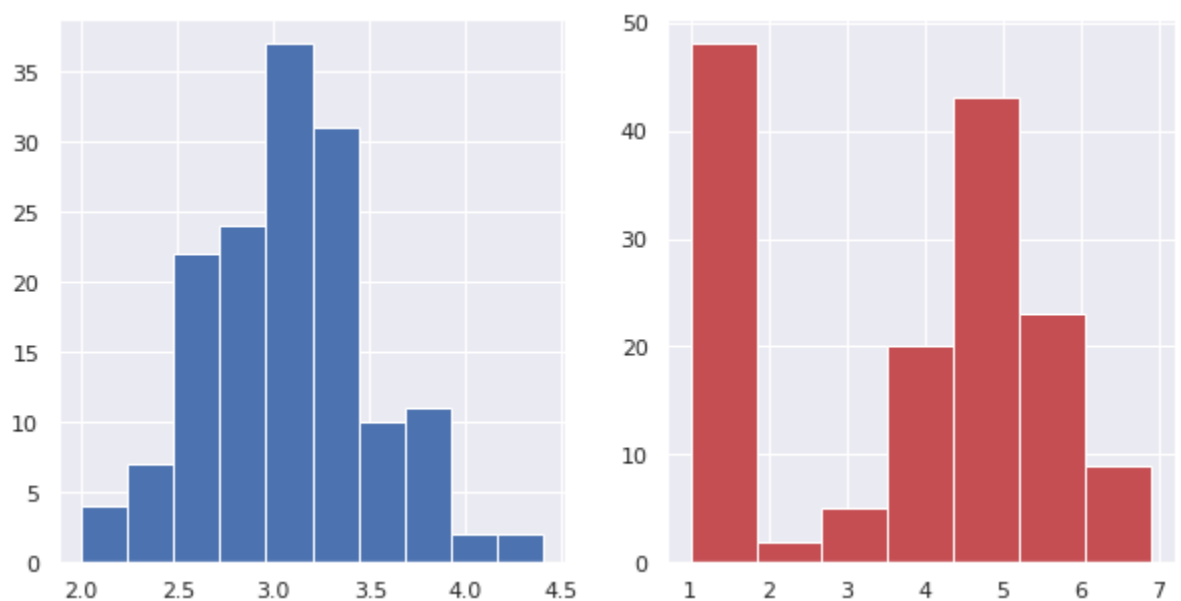
Seaborn

```
1 sns.set()
2
3 iris = sns.load_dataset("iris")
4
5 sns.violinplot(x="species", y="sepal_width", data=iris);
```



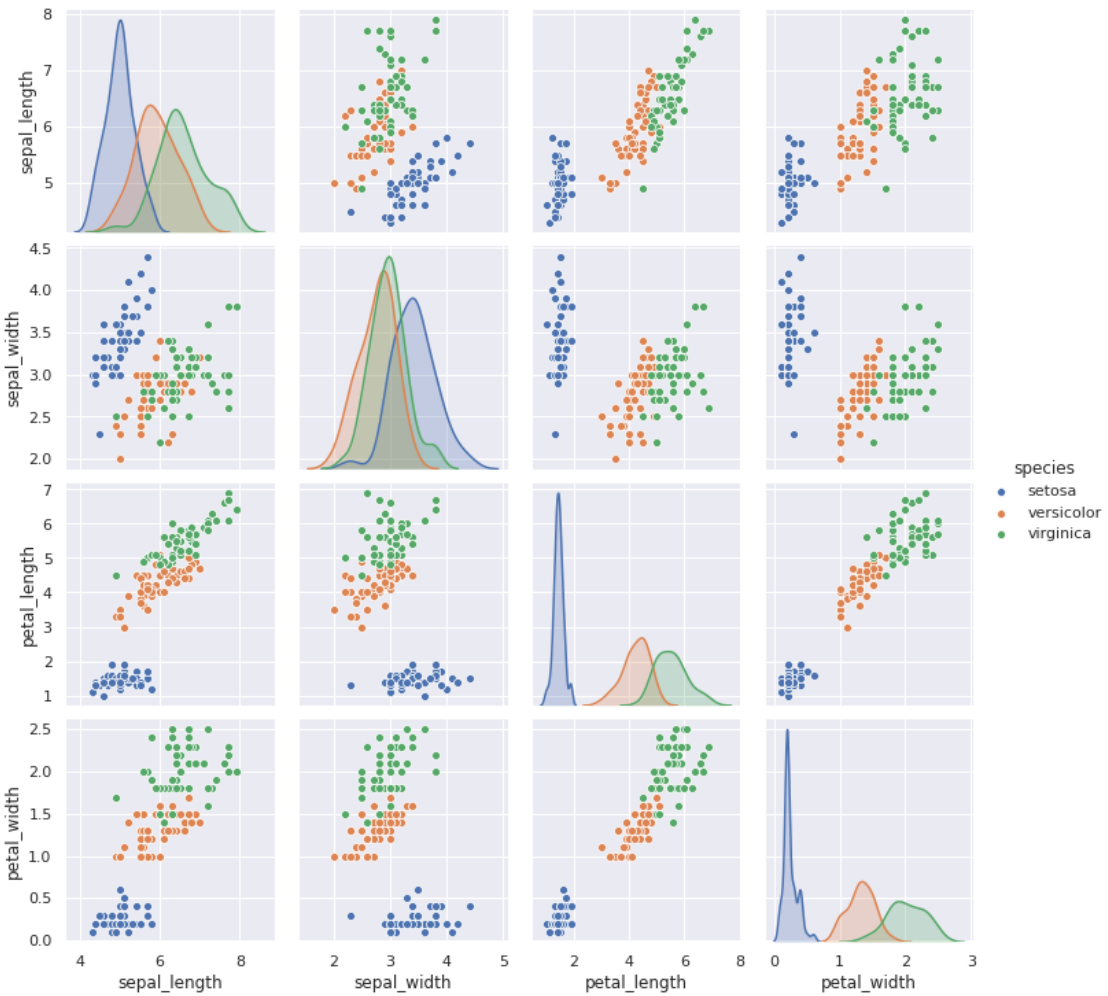
Seaborn

```
#Iris Dataset
data = sns.load_dataset("iris")
f, (ax1, ax2) = plt.subplots (1, 2, figsize=(10,5))
#histogram for sepal_widht colored as blue
ax1.hist(data["sepal_width"], label="sepal_width",bins=10, color='b')
#histogram for sepal_widht colored as red
ax2.hist(data["petal_length"], label="petal_length",bins=7, color='r')
plt.show()
```

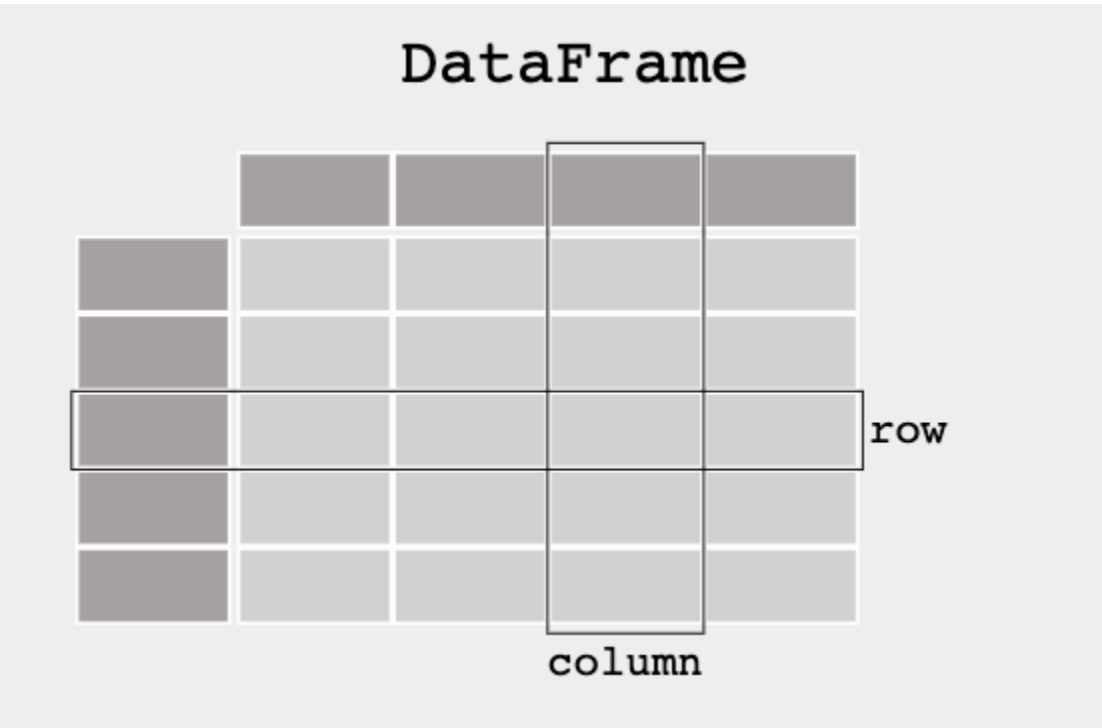


Seaborn

```
1 sns.pairplot(data, hue='species')
```



Pandas



https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

Pandas

Kaggle - Gotta train'em all

```
1 import pandas as pd
2
3 df = pd.read_csv("pokemon.csv", encoding = "latin-1")
4 df.head(10)
```

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

Pandas

Kaggle - Gotta train'em all

```python
import pandas as pd

df = pd.read_csv("pokemon.csv", encoding = "latin-1")
df.head(10)
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False
5	5	Charmeleon	Fire	NaN	405	58	64	58	80	65	80	1	False
6	6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	False
7	6	CharizardMega Charizard X	Fire	Dragon	634	78	130	111	130	85	100	1	False
8	6	CharizardMega Charizard Y	Fire	Flying	634	78	104	78	159	115	100	1	False
9	7	Squirtle	Water	NaN	314	44	48	65	50	64	43	1	False

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

Pandas

```
1 import pandas as pd
2
3 df.tail()
4 #transposed version of dataframe
5 df.tail().T
```

https://nbviewer.jupyter.org/github/schmit/cme193-ipython-notebooks-lecture/blob/master/2.%20Pandas.ipynb

Pandas

```
1 import pandas as pd
2
3 df.tail()
4 #transposed version of dataframe
5 df.tail().T
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
794	718	Zygarde50% Forme	Dragon	Ground	600	108	100	121	81	95	95	6	True
795	719	Diancie	Rock	Fairy	600	50	100	150	100	150	50	6	True
796	719	DiancieMega Diancie	Rock	Fairy	700	50	160	110	160	110	110	6	True
797	720	HoopaHoopa Confined	Psychic	Ghost	600	80	110	60	150	130	70	6	True
798	720	HoopaHoopa Unbound	Psychic	Dark	680	80	160	60	170	130	80	6	True
799	721	Volcanion	Fire	Water	600	80	110	120	130	90	70	6	True

https://nbviewer.jupyter.org/github/schmit/cme193-ipython-notebooks-lecture/blob/master/2.%20Pandas.ipynb

Pandas

```
1 import pandas as pd
2
3 df.tail()
4 #transposed version of dataframe
5 df.tail().T
```

	794	795	796	797	798	799
#	718	719	719	720	720	721
Name	Zygarde50% Forme	Diancie	DiancieMega Diancie	HoopaHoopa Confined	HoopaHoopa Unbound	Volcanion
Type 1	Dragon	Rock	Rock	Psychic	Psychic	Fire
Type 2	Ground	Fairy	Fairy	Ghost	Dark	Water
Total	600	600	700	600	680	600
HP	108	50	50	80	80	80
Attack	100	100	160	110	160	110
Defense	121	150	110	60	60	120
Sp. Atk	81	100	160	150	170	130
Sp. Def	95	150	110	130	130	90
Speed	95	50	110	70	80	70
Generation	6	6	6	6	6	6
Legendary	True	True	True	True	True	True

https://nbviewer.jupyter.org/github/schmit/cme193-ipython-notebooks-lecture/blob/master/2.%20Pandas.ipynb

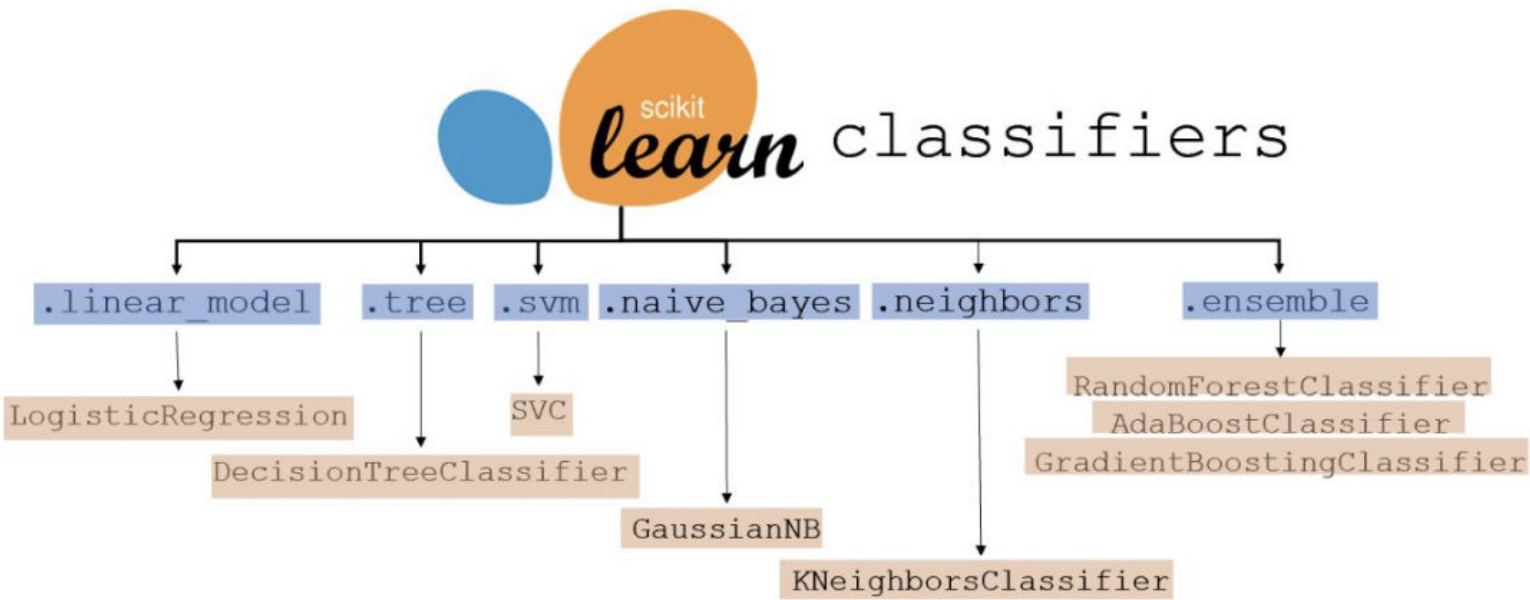
Pandas

```
1 import pandas as pd
2
3 #the information located in 117
4 df.loc[117]
```

```
#                109
Name          Koffing
Type 1         Poison
Type 2            NaN
Total             340
HP                 40
Attack             65
Defense            95
Sp. Atk            60
Sp. Def            45
Speed              35
Generation          1
Legendary       False
Name: 117, dtype: object
```

https://nbviewer.jupyter.org/github/schmit/cme193-ipython-notebooks-lecture/blob/master/2.%20Pandas.ipynb

Scikit-Learn



Demo

Gotta Train'em All!

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 %matplotlib inline #to render the figure in notebook
6
7 pokemon = pd.read_csv("pokemon.csv", index_col=0)
8 pokemon.head()
```

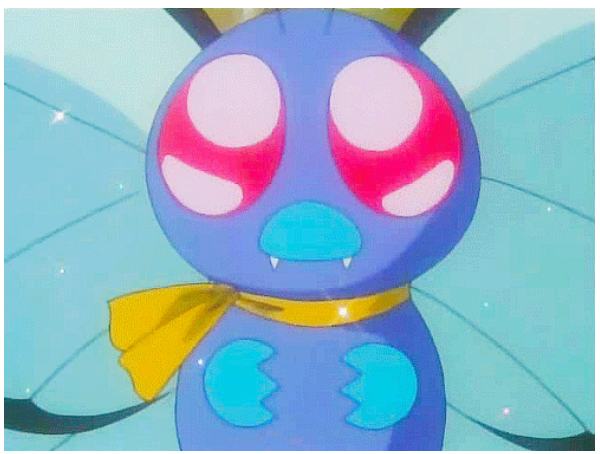
#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

Gotta Train'em All!

```
1 pokemon.groupby(['Type 1', 'Type 2'])[['Speed']].mean()
```

Type 1	Type 2	Speed
Bug	Electric	86.500000
	Fighting	80.000000
	Fire	80.000000
	Flying	82.857143
	Ghost	40.000000
...	...	...
Water	Ice	66.666667
	Poison	85.000000
	Psychic	44.000000
	Rock	36.000000
	Steel	60.000000

136 rows × 1 columns



https://nbviewer.jupyter.org/github/schmit/cme193-ipython-notebooks-lecture/blob/master/2.%20Pandas.ipynb

Gotta Train'em All!

```
1 #Number of pokemon has Null value in feature value
2 pokemon.isnull().sum()
```

```
Name            0
Type 1          0
Type 2        386
Total           0
HP              0
Attack          0
Defense         0
Sp. Atk         0
Sp. Def         0
Speed           0
Generation      0
Legendary       0
dtype: int64
```

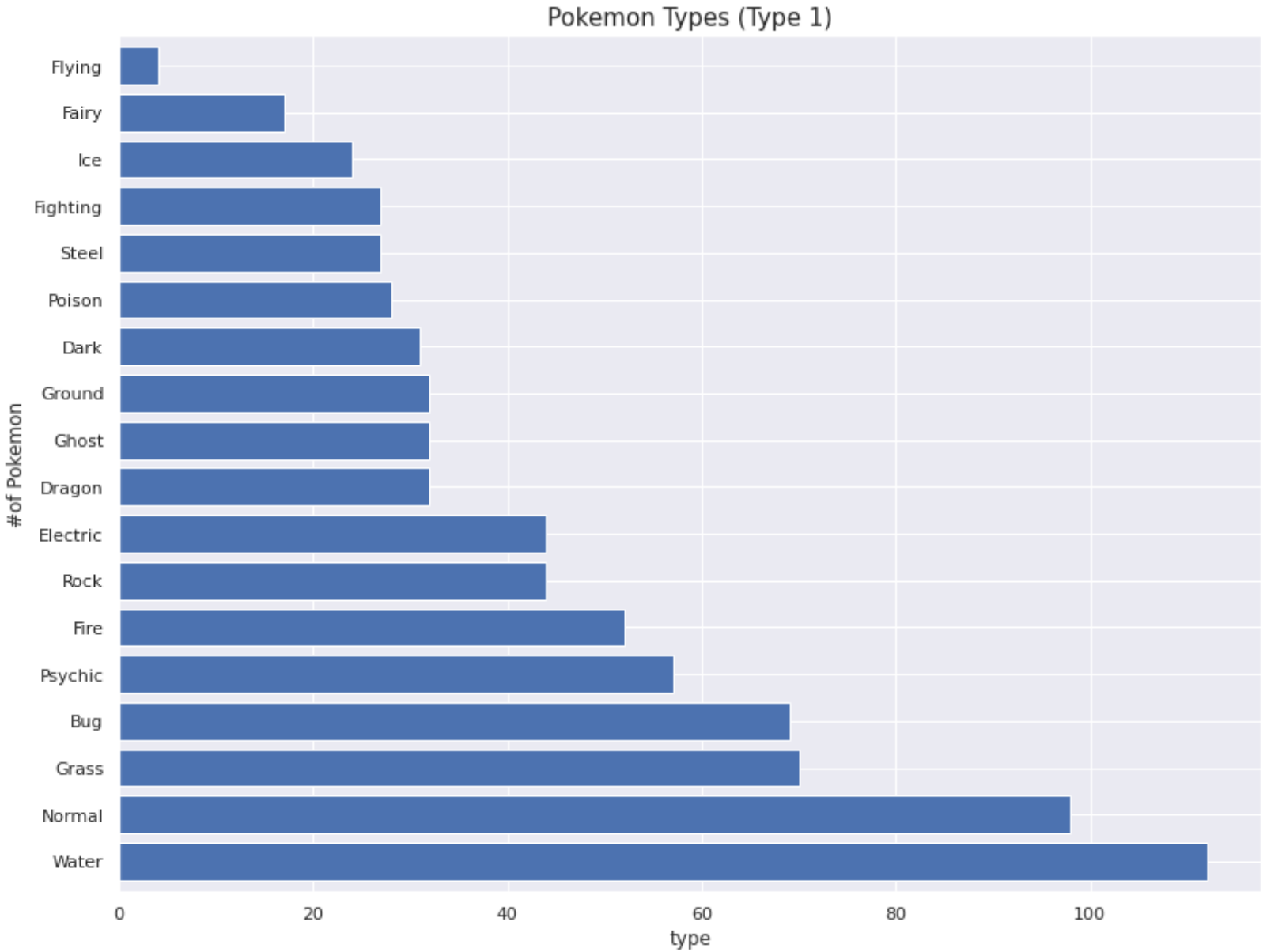
Gotta Train'em All!

```
sns.set_style('darkgrid')
sns.set_color_codes("pastel")
plt.figure(figsize=(13,10))

#Number of pokemon according to their Type 1
pokemon['Type 1'].value_counts().plot.barh(width=.8).set_title('Pokemon Types (Type 1)', fontsize

plt.xlabel('type')
plt.ylabel('#of Pokemon');
```

Gotta Train'em All!

```
1 sns.set
2 sns.set
3 plt.fig
4
5 #Number
6 pokemon
7
8 plt.xla
9 plt.yla
```

```
ontsize=
```



Gotta Train'em All!

```
1 #How many legendary pokemon exists in this list?
2 pokemon['Legendary'].value_counts()
```

Gotta Train'em All!

```
1 #How many legendary pokemon exists in this list?
2 pokemon['Legendary'].value_counts()
```

```
False    735
True      65
Name: Legendary, dtype: int64
```

Gotta Train'em All!

```
1 #How many legendary pokemon exists in this list?
2 pokemon['Legendary'].value_counts()
```

```
False    735
True      65
Name: Legendary, dtype: int64
```

```
1 #What is the type of Legendary Pokemons according to their Type 1 value?
2 pokemon[pokemon['Legendary']==True]['Type 1'].value_counts()
```

Gotta Train'em All!

```
#How many legendary pokemon exists in this list?
pokemon['Legendary'].value_counts()
```

```
False    735
True      65
Name: Legendary, dtype: int64
```

```
#What is the type of Legendary Pokemons according to their Type 1 value?
pokemon[pokemon['Legendary']==True]['Type 1'].value_counts()
```

```
Psychic     14
Dragon      12
Fire         5
Electric     4
Ground       4
Rock         4
Water        4
Steel        4
Grass        3
Flying       2
Ghost        2
Normal       2
Ice          2
Dark         2
Fairy        1
Name: Type 1, dtype: int64
```

Gotta Train'em All!

```
#Clean the data
#Since we don't need Type 2, Name and Total we'll drop them
pokemon_cln = pokemon.copy()
pokemon_cln.drop(['Name', 'Type 2', 'Total'], axis=1, inplace=True)
pokemon_cln.head(10)
```

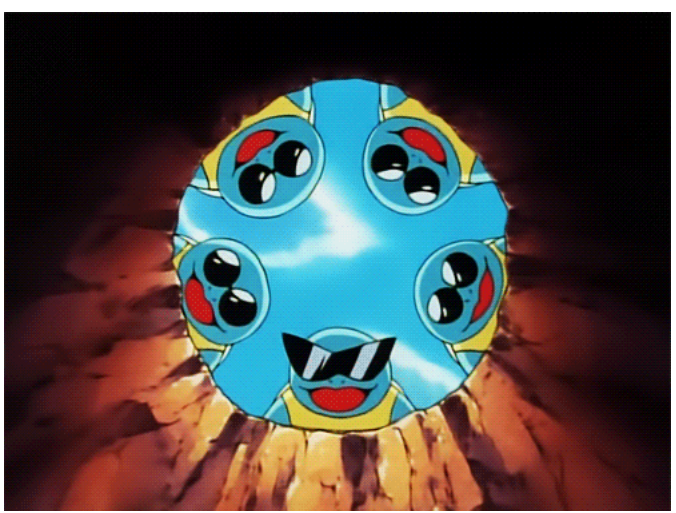
Gotta Train'em All!

```
#Clean the data
#Since we don't need Type 2, Name and Total we'll drop them
pokemon_cln = pokemon.copy()
pokemon_cln.drop(['Name', 'Type 2', 'Total'], axis=1, inplace=True)
pokemon_cln.head(10)
```

#	Type 1	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
1	Grass	45	49	49	65	65	45	1	False
2	Grass	60	62	63	80	80	60	1	False
3	Grass	80	82	83	100	100	80	1	False
3	Grass	80	100	123	122	120	80	1	False
4	Fire	39	52	43	60	50	65	1	False
5	Fire	58	64	58	80	65	80	1	False
6	Fire	78	84	78	109	85	100	1	False
6	Fire	78	130	111	130	85	100	1	False
6	Fire	78	104	78	159	115	100	1	False
7	Water	44	48	65	50	64	43	1	False

Gotta Train'em All!

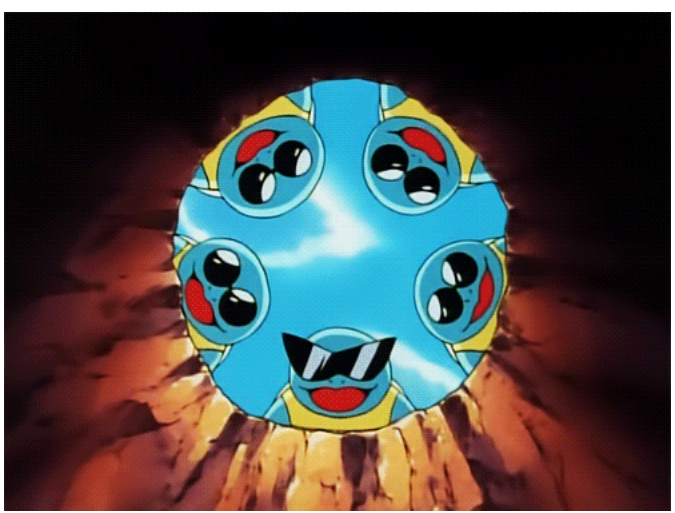
```
1 pokemon_cln.dtypes
2 # Type 1 is object
3 # Legendary is bool
```



Gotta Train'em All!

```
1 pokemon_cln.dtypes
2 # Type 1 is object
3 # Legendary is bool
```

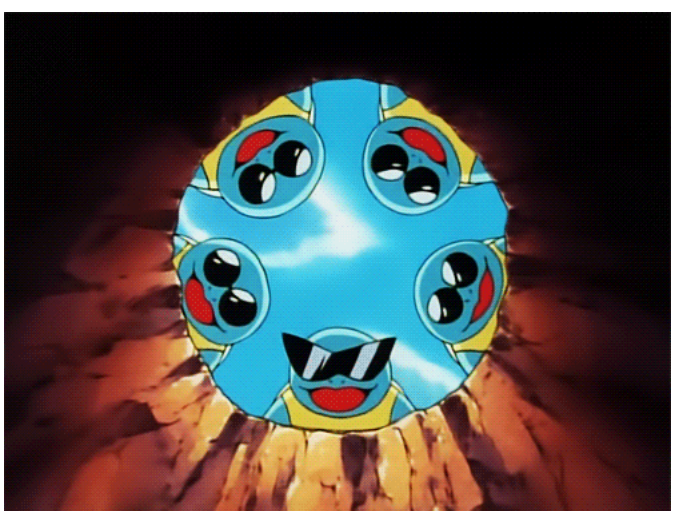
```
Type 1          object
HP               int64
Attack           int64
Defense          int64
Sp. Atk          int64
Sp. Def          int64
Speed            int64
Generation       int64
Legendary         bool
dtype: object
```



Gotta Train'em All!

```
1 #Standardize data for Legendary
2 pokemon_cln['Legendary'] = pokemon_cln['Legendary'].astype(int)
3 #check the data types again
4 pokemon_cln.dtypes
```

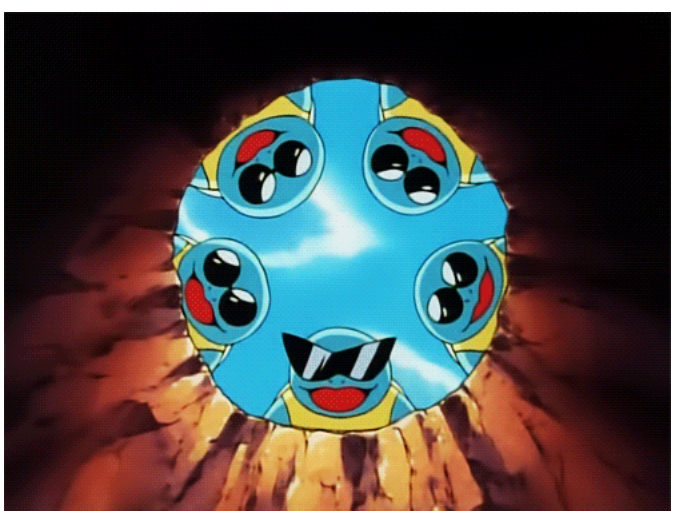
```
Type 1          object
HP               int64
Attack           int64
Defense          int64
Sp. Atk          int64
Sp. Def          int64
Speed            int64
Generation       int64
Legendary         bool
dtype: object
```



Gotta Train'em All!

```
1 #Standardize data for Legendary
2 pokemon_cln['Legendary'] = pokemon_cln['Legendary'].astype(int)
3 #check the data types again
4 pokemon_cln.dtypes
```

```
Type 1          object
HP               int64
Attack           int64
Defense          int64
Sp. Atk          int64
Sp. Def          int64
Speed            int64
Generation       int64
Legendary        int64
dtype: object
```



Gotta Train'em All!

```
# Standardize data for Type 1
from sklearn.preprocessing import LabelEncoder
lb_make = LabelEncoder()
pokemon_cln['Type 1'] = lb_make.fit_transform(pokemon_cln['Type 1'])
```

Gotta Train'em All!

```
# Standardize data for Type 1
from sklearn.preprocessing import LabelEncoder
lb_make = LabelEncoder()
pokemon_cln['Type 1'] = lb_make.fit_transform(pokemon_cln['Type 1'])
```

```
Type 1          int64
HP              int64
Attack          int64
Defense         int64
Sp. Atk         int64
Sp. Def         int64
Speed           int64
Generation      int64
Legendary       int64
dtype: object
```

Gotta Train'em All!

```
# Standardize data for Type 1
from sklearn.preprocessing import LabelEncoder
lb_make = LabelEncoder()
pokemon_cln['Type 1'] = lb_make.fit_transform(pokemon_cln['Type 1'])
```

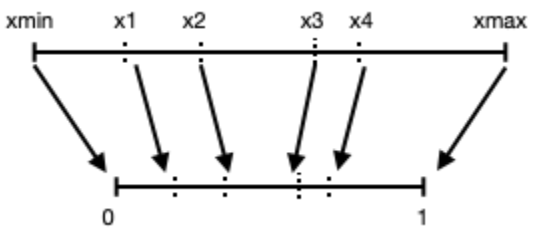
```
Type 1          int64
HP              int64
Attack          int64
Defense         int64
Sp. Atk         int64
Sp. Def         int64
Speed           int64
Generation      int64
Legendary       int64
dtype: object
```

Gotta Train'em All!

```
#Min-Max Normalization is applied
#but we'll exclude Legendary since it'll be our class value
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
X_minmax = min_max_scaler.fit_transform(pokemon_cln.drop('Legendary', axis=1))
X = pd.DataFrame(X_minmax, columns=pokemon_cln.columns[:-1])
y = pokemon_cln['Legendary']
print(X,y)
```

Gotta Train'em All!

```
#Min-Max Normalization is applied
#but we'll exclude Legendary since it'll be our class value
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
X_minmax = min_max_scaler.fit_transform(pokemon_cln.drop('Legendary', axis=1))
X = pd.DataFrame(X_minmax, columns=pokemon_cln.columns[:-1])
y = pokemon_cln['Legendary']
print(X,y)
```



Gotta Train'em All!

```
#Min-Max Normalization is applied
#but we'll exclude Legendary since it'll be our class value
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
X_minmax = min_max_scaler.fit_transform(pokemon_gln.drop('Legendary', axis=1))
X =
y =
print
```

	Type 1	HP	Attack	...	Sp. Def	Speed	Generation
0	0.529412	0.173228	0.237838	...	0.214286	0.228571	0.0
1	0.529412	0.232283	0.308108	...	0.285714	0.314286	0.0
2	0.529412	0.311024	0.416216	...	0.380952	0.428571	0.0
3	0.529412	0.311024	0.513514	...	0.476190	0.428571	0.0
4	0.352941	0.149606	0.254054	...	0.142857	0.342857	0.0
..	...	...	...	...	...	...	...
795	0.882353	0.192913	0.513514	...	0.619048	0.257143	1.0
796	0.882353	0.192913	0.837838	...	0.428571	0.600000	1.0
797	0.823529	0.311024	0.567568	...	0.523810	0.371429	1.0
798	0.823529	0.311024	0.837838	...	0.523810	0.428571	1.0
799	0.352941	0.311024	0.567568	...	0.333333	0.371429	1.0

[800 rows x 8 columns] #

```
1      0
2      0
3      0
3      0
4      0
      ..
719    1
719    1
720    1
720    1
721    1
Name: Legendary, Length: 800, dtype: int64
```

Gotta Train'em All!

```
1 from sklearn.model_selection import train_test_split
2 #Divide the dataset into train and test
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=90)
4 #print out the length of arrays in order
5 print("X_train: {0}, y_train: {1}, X_test: {2}, y_test: {3}".format(len(X_train), len(y_train), l
```

```
X_train: 560, y_train: 560, X_test: 240, y_test: 240
```

@abilgegunduz

@aysebilgegunduz

womentech network

Thanks for listening...