

DATA MANAGEMENT & VISUALIZATION

Machine Learning - Yousef Elbaroudy

GUIDELINES

- Try to **SHUTDOWN YOUR PHONE** and focus on the important information mentioned through the session
- Apply what you take on the practical section
- Do not try to memorize everything you got, just learn
- Don't mind to ask about anything you want to know

Enjoy the Session 😊

DATA MANAGEMENT

- In order to manage data through python, there are many ways to do it !
- Also, you may need to do some mathematical operations to matrices, lists or any structure
- You can simply manage your data as the same as SQL

HOW THEN ?

NumPy



WHAT IS NUMPY ?

- NumPy is an open-source Python library that facilitates efficient numerical operations on large quantities of data.
- There are a few functions that exist in NumPy that we use on pandas DataFrames.
- For us, the most important part about NumPy is that pandas is built on top of it. So, NumPy is a dependency of Pandas.



NUMPY ARRAYS

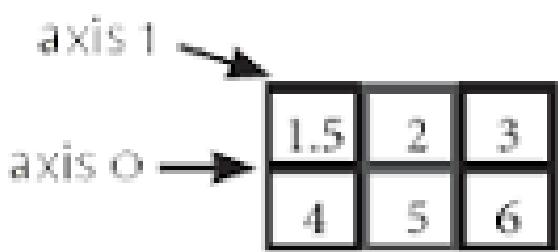
- NumPy arrays are unique in that they are more flexible than normal Python lists.
- They are called ndarrays since they can have any number (n) of dimensions (d).
- They hold a collection of items of any one data type and can be either a vector (one-dimensional) or a matrix (multi-dimensional).
- NumPy arrays allow for **fast element access** and efficient data manipulation.

NumPy Arrays

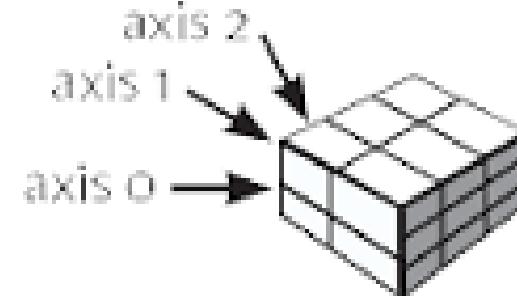
1D array



2D array



3D array





Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)],[(3,2,1), (4,5,6)]], dtype = float)
```

CREATING ARRAYS

Initial Placeholders

```
>>> np.zeros((3,4)) #Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16) #Create an array of ones
>>> d = np.arange(10,25,5) #Create an array of evenly spaced values (step value)
>>> np.linspace(0,2,9) #Create an array of evenly spaced values (number of samples)
>>> e = np.full((2,2),7) #Create a constant array
>>> f = np.eye(2) #Create a 2X2 identity matrix
>>> np.random.random((2,2)) #Create an array with random values
>>> np.empty((3,2)) #Create an empty array
```



Inspecting Your Array

```
>>> a.shape #Array dimensions  
>>> len(a) #Length of array  
>>> b.ndim #Number of array dimensions  
>>> e.size #Number of array elements  
>>> b.dtype #Data type of array elements  
>>> b.dtype.name #Name of data type  
>>> b.astype(int) #Convert an array to a different type
```



Data Types

```
>>> np.int64 #Signed 64-bit integer types  
>>> np.float32 #Standard double-precision floating point  
>>> np.complex #Complex numbers represented by 128 floats  
>>> np.bool #Boolean type storing TRUE and FALSE values  
>>> np.object #Python object type  
>>> np.string_ #Fixed-length string type  
>>> np.unicode_ #Fixed-length unicode type
```

DATATYPES

`np.inf`

`np.nan`

ALSO

> Array Mathematics

Arithmetic Operations

```
>>> g = a - b #Subtraction
array([[-0.5, 0. , 0. ],
       [-3. , -3. , -3. ]])
>>> np.subtract(a,b) #Subtraction
>>> b + a #Addition
array([[ 2.5, 4. , 6. ],
       [ 5. , 7. , 9. ]])
>>> np.add(b,a) Addition
>>> a / b #Division
array([[ 0.66666667, 1. , 1. ],
       [ 0.25 , 0.4 , 0.5 ]])
>>> np.divide(a,b) #Division
>>> a * b #Multiplication
array([[ 1.5, 4. , 9. ],
       [ 4. , 10. , 18. ]])
>>> np.multiply(a,b) #Multiplication
>>> np.exp(b) #Exponentiation
>>> np.sqrt(b) #Square root
>>> np.sin(a) #Print sines of an array
>>> np.cos(b) #Element-wise cosine
>>> np.log(a) #Element-wise natural logarithm
>>> e.dot(f) #Dot product
array([[ 7., 7.],
       [ 7., 7.]])
```

ARITHMETIC OPERATIONS

Comparison

```
>>> a == b #Element-wise comparison
array([[False, True, True],
       [False, False, False]], dtype=bool)
>>> a < 2 #Element-wise comparison
array([True, False, False], dtype=bool)
>>> np.array_equal(a, b) #Array-wise comparison
```

Aggregate Functions

```
>>> a.sum() #Array-wise sum  
>>> a.min() #Array-wise minimum value  
>>> b.max(axis=0) #Maximum value of an array row  
>>> b.cumsum(axis=1) #Cumulative sum of the elements  
>>> a.mean() #Mean  
>>> np.median(b) #Median  
>>> np.corrcoef(a) #Correlation coefficient  
>>> np.std(b) #Standard deviation
```



Copying Arrays

```
>>> h = a.view() #Create a view of the array with the same data  
>>> np.copy(a) #Create a copy of the array  
>>> h = a.copy() #Create a deep copy of the array
```

COPYING ARRAY

>

Sorting Arrays

```
>>> a.sort() #Sort an array  
>>> c.sort(axis=0) #Sort the elements of an array's axis
```

SORTING

> Subsetting, Slicing, Indexing

Subsetting

```
>>> a[2] #Select the element at the 2nd index  
3  
>>> b[1,2] #Select the element at row 1 column 2 (equivalent to b[1][2])  
6.0
```

1	2	3
1.5	2	3
4	5	6

Slicing

```
>>> a[0:2] #Select items at index 0 and 1  
array([1, 2])  
>>> b[0:2,1] #Select items at rows 0 and 1 in column 1  
array([ 2., 5.])  
>>> b[:,1] #Select all items at row 0 (equivalent to b[0:1, :])  
array([[1.5, 2., 3.]])  
>>> c[1,...] #Same as [1,:,:]  
array([[[ 3., 2., 1.],  
       [ 4., 5., 6.]]])  
>>> a[ : :-1] #Reversed array a array([3, 2, 1])
```

1	2	3
1.5	2	3
4	5	6

1.5	2	3
4	5	6

SUBSETTING, SLICING

INDEXING

Boolean Indexing

```
>>> a[a<2] #Select elements from a less than 2  
array([1])
```

1	2	3
---	---	---

Fancy Indexing

```
>>> b[[1, 0, 1, 0], [0, 1, 2, 0]] #Select elements (1,0),(0,1),(1,2) and (0,0)  
array([ 4., 2., 6., 1.5])  
>>> b[[1, 0, 1, 0]][[:,[0,1,2,0]]] #Select a subset of the matrix's rows and columns  
array([[ 4., 5., 6., 4.],  
       [ 1.5, 2., 3., 1.5],  
       [ 4., 5., 6., 4.],  
       [ 1.5, 2., 3., 1.5]])
```



Array Manipulation

Transposing Array

```
>>> i = np.transpose(b) #Permute array dimensions  
>>> i.T #Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel() #Flatten the array  
>>> g.reshape(3,-2) #Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6)) #Return a new array with shape (2,6)  
>>> np.append(h,g) #Append items to an array  
>>> np.insert(a, 1, 5) #Insert items in an array  
>>> np.delete(a,[1]) #Delete items from an array
```

ARRAY MANIPULATION

Combining Arrays

```
>>> np.concatenate((a,d),axis=0) #Concatenate arrays
array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b)) #Stack arrays vertically (row-wise)
array([[ 1. ,  2. ,  3. ],
       [ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f] #Stack arrays vertically (row-wise)
>>> np.hstack((e,f)) #Stack arrays horizontally (column-wise)
array([[ 7.,  7.,  1.,  0.],
       [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d)) #Create stacked column-wise arrays
array([[ 1, 10],
       [ 2, 15],
       [ 3, 20]])
>>> np.c_[a,d] #Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3) #Split the array horizontally at the 3rd index
[array([1]),array([2]),array([3])]
>>> np.vsplit(c,2) #Split the array vertically at the 2nd index
[array([[ 1.5,  2. ,  1. ],
       [ 4. ,  5. ,  6. ]]),
 array([[ 3.,  2.,  3.],
       [ 4.,  5.,  6.]])]
```

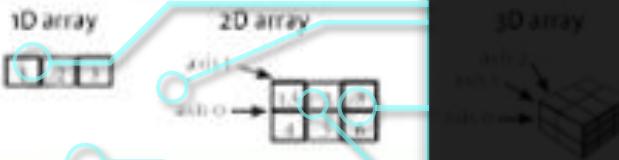
Numpy

The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



Creating Arrays

```
arr = np.array([1,2,3])
arr = np.arange(1,5,1)
arr = np.arange(1,5,1, dtype = float)
arr = np.arange(1,5,1, (1,2,3,4,5))
arr = np.zeros(3)
arr = np.ones((3,2))
arr = np.empty((3,2))
arr = np.full((3,2), 10)
arr = np.full((3,2), 10, dtype = complex)
arr = np.eye(3,3)
arr = np.identity(3)
arr = np.random(3,2)
arr = np.random((3,2))
arr = np.empty_like(arr)
```

Initial Placeholders

```
np.zeros(3,4) #Create an array of zeros
np.ones(3,4) #Create an array of ones
np.empty(3,4) #Create an array of empty spaced values (tiny zeros)
np.full(3,4) #Create an array of evenly spaced values (number of samples)
np.linspace(0,1,10) #Create a constant array
np.eye(3,3) #Create identity matrix
np.random.rand(3,2) #Create an array with random values
np.empty(3,2) #Create an empty array
```

I/O

Saving & Loading On Disk

```
np.savetxt('array1', a)
np.savetxt('array1.txt', a)
np.loadtxt('array1')
```

Inspecting Your Array

```
arr.shape #Array dimensions
arr.size #Length of array
arr.ndim #Number of array dimensions
arr.itemsize #Number of array elements
arr.dtype.name #Type of array elements
arr.dtype.type #Type of data type
arr.dtype.item #Element in array in a different type
```

Data Types

```
np.int_, np.int8, np.int16, np.int32, np.int64 #Integer types
np.float_, np.float16, np.float32, np.float64 #Floating point
np.bool_ #Boolean numbers represented as 1/0 floats
np.bool_ #Boolean type storing TRUE and FALSE values
np.str_ #String object type
np.string_, np.unicode_ #String string type
np.datetime_ #Aligned-timedelta timedelta type
```

Sorting Arrays

```
arr.argsort() #Sort on array
arr.sort() #Sort the elements of an array's data
```

Subsetting, Slicing, Indexing

Subsetting

```
arr[2] #Select the element at the 2nd index
arr[1,2] #Select the element at row 1 column 2 (equivalent to arr[1][2])
arr[1:2]
```



Slicing

```
arr[1:2] #Select items at index 1 and 2
arr[1:-1]
arr[1:2, 1:2] #Select items of row 1 and 2 in column 1
arr[1:2, 1:-1]
arr[1:-1, 1:-1] #Select all items at row 1 (equivalent to arr[1][1:-1])
arr[1:-1, 1:-1]
```



Advanced Indexing

```
arr[[1,2], [1,2]] #Select items from row 1 and 2 in column 1 and 2
arr[[1,2], 1:2]
```



Boolean Indexing

```
arr[(arr>1) & (arr<4)] #Select elements (1,2,3) in row 1 and (2,3,4) in row 2
arr[(arr>1) & (arr<4), 1:2] #Select a subset of the matrix's rows and columns
arr[(arr>1) & (arr<4), 1:-1]
```



Advanced Manipulation

Transposing Array

```
arr.T #Transpose the array
arr.T #Transpose array dimensions
```

Reshaping Array

```
arr.reshape(3,3) #Reshape the array
arr.reshape(3,-1) #Reshape, but don't change data
```

Adding/Removing Elements

```
arr.resize(3,3) #Return a new array with shape (3,3)
arr.append([1]) #Append item to an array
arr.insert(1, 0) #Insert item at an array
arr.delete(1) #Delete item from array
```

Combining Arrays

```
np.concatenate((a,b),axis=0) #Concatenate arrays
array([1, 2, 3, 10, 20])
np.vstack((a,b)) #Stack arrays vertically (row-wise)
array([[1, 2, 3],
       [10, 20, 30]])
np.hstack((a,b)) #Stack arrays horizontally (column-wise)
array([1, 2, 3,
       10, 20, 30])
np.column_stack((a,b)) #Creates stacked column-wise arrays
array([1, 2, 3,
       1, 2, 3,
       10, 20, 30])
np.dstack((a,b)) #Creates stacked volume-wise arrays
```

Defining Arrays

```
np.array([1,2]) #Create stacked column-wise arrays
```

DATA CAMP CHEAT SHEET (NUMPY)

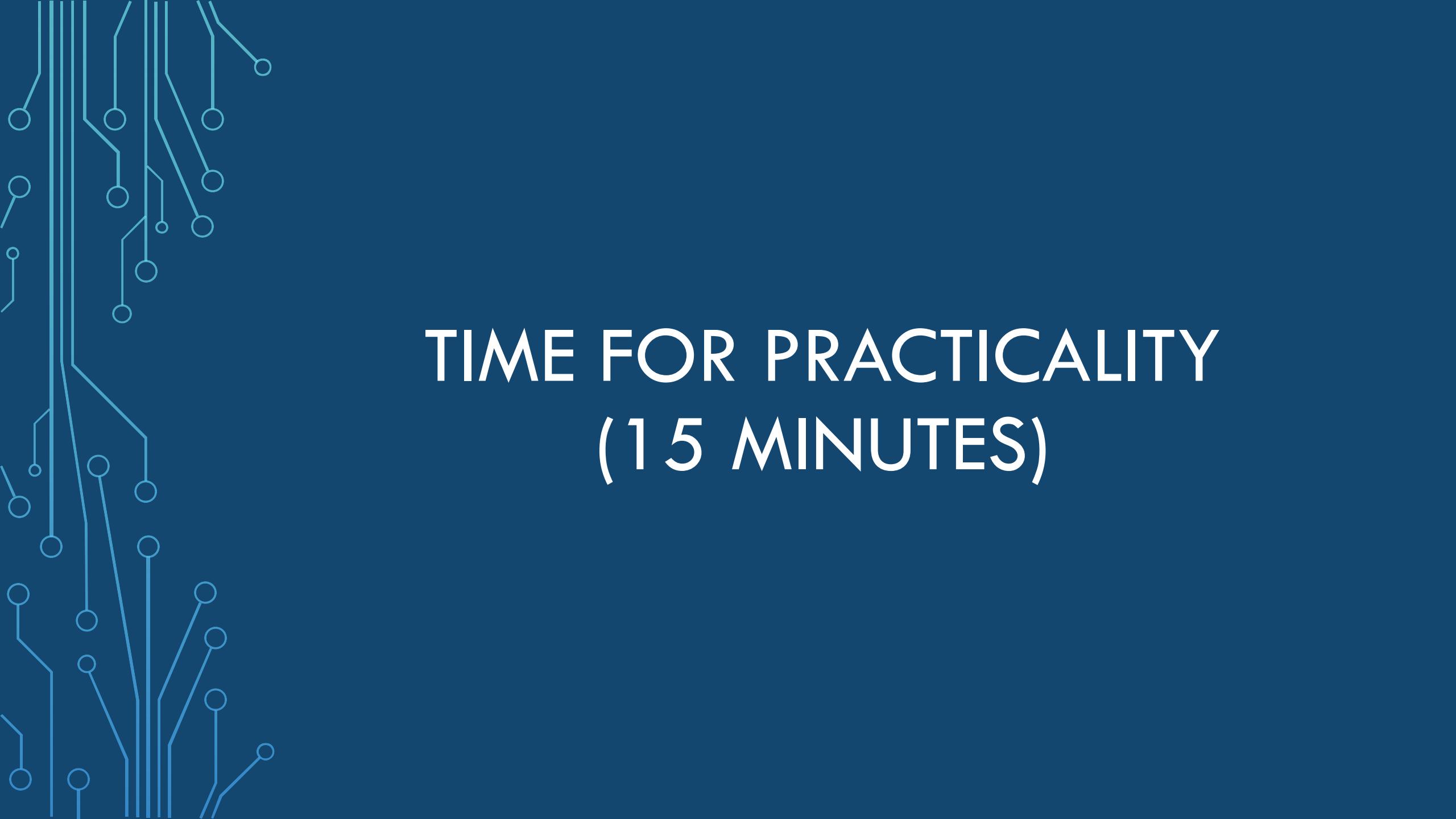
https://images.datacamp.com/image/upload/v1676302459/Marketing/Blog/Numpy_Cheat_Sheet.pdf

Comparison

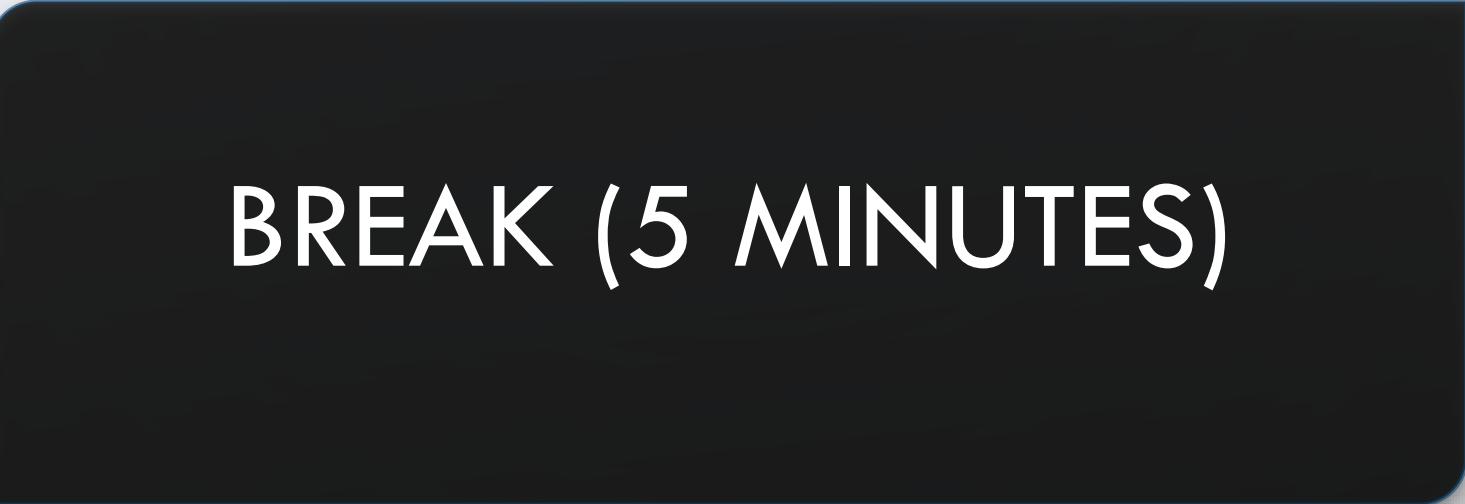
```
arr == arr #Element-wise comparison
arr != arr #Element-wise True/False
arr < arr #Element-wise comparison
arr <= arr #Element-wise comparison
arr == arr #Matrix-wise comparison
```

Aggregate Functions

```
arr.sum() #Array-wise sum
arr.min() #Array-wise minimum value
arr.max() #Maximum value of an array
arr.sum(axis=0) #Summation axis of the elements
arr.mean() #Mean
arr.median() #Median
arr.std() #Standard deviation
```



TIME FOR PRACTICALITY
(15 MINUTES)



BREAK (5 MINUTES)

Pandas



WHAT IS PANDAS ?

- *Pandas* is a very popular library for working with data (its goal is to be the most powerful and flexible open-source tool, and in our opinion, it has reached that goal).
- **DataFrames** are at the center of pandas. A DataFrame is structured like a table or spreadsheet. The rows and the columns both have indexes, and you can perform operations on rows or columns separately.

DATAFRAME

- A pandas **DataFrame** can be easily changed and manipulated. Pandas has helpful functions for handling missing data, performing operations on columns and rows, and transforming data.
- If that wasn't enough, a lot of SQL functions have counterparts in pandas, such as join, merge, filter by, and group by.
- With all of these powerful tools, it should come as no surprise that pandas is very popular among data scientists.

> Pandas Data Structures

Series

A one-dimensional labeled array capable of holding any data type

Index →

a	3
b	-5
c	7
d	4

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

Dataframe

A two-dimensional labeled data structure with columns of potentially different types

Columns →

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847628

Index →

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
   ...: 'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
   ...: 'Population': [11190846, 1303171035, 207847628]}
>>> df = pd.DataFrame(data,
   ...:                   columns=['Country', 'Capital', 'Population'])
```



Dropping

```
>>> s.drop(['a', 'c']) #Drop values from rows (axis=0)  
>>> df.drop('Country', axis=1) #Drop values from columns(axis=1)
```

DROPPING

> I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> df.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

I/O

> Selection

Also see NumPy Arrays

Getting

```
>>> s['b'] #Get one element  
-5  
>>> df[1:] #Get subset of a DataFrame  
Country Capital Population  
1 India New Delhi 1303171035  
2 Brazil Brasilia 207847528
```

SELECTION

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]] #Select single value by row & column  
'Belgium'  
>>> df.iat([0],[0])  
'Belgium'
```

By Label

```
>>> df.loc[[0], ['Country']] #Select single value by row & column labels  
'Belgium'  
>>> df.at([0], ['Country'])  
'Belgium'
```

By Label/Position

```
>>> df.ix[2] #Select single row of subset of rows
Country Brazil
Capital Brasilia
Population 207847528
>>> df.ix[:, 'Capital'] #Select a single column of subset of columns
0 Brussels
1 New Delhi
2 Brasilia
>>> df.ix[1, 'Capital'] #Select rows and columns
'New Delhi'
```

Boolean Indexing

```
>>> s[~(s > 1)] #Series s where value is not >1
>>> s[(s < -1) | (s > 2)] #s where value is <-1 or >2
>>> df[df['Population']>1200000000] #Use filter to adjust DataFrame
```

Setting

```
>>> s['a'] = 6 #Set index a of Series s to 6
```



Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape #(rows,columns)
>>> df.index #Describe index
>>> df.columns #Describe DataFrame columns
>>> df.info() #Info on DataFrame
>>> df.count() #Number of non-NA values
```

RETRIEVING INFORMATION

Summary

```
>>> df.sum() #Sum of values  
>>> df.cumsum() #Cummulative sum of values  
>>> df.min()/df.max() #Minimum/maximum values  
>>> df.idxmin()/df.idxmax() #Minimum/Maximum index value  
>>> df.describe() #Summary statistics  
>>> df.mean() #Mean of values  
>>> df.median() #Median of values
```

> Applying Functions

```
>>> f = lambda x: x*2  
>>> df.apply(f) #Apply function  
>>> df.applymap(f) #Apply function element-wise
```

APPLYING FUNCTION

> Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a 10.0
b NaN
c 5.0
d 7.0
```

DATA ALIGNMENT

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_values=0)
a 10.0
b -5.0
c 5.0
d 7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

Pandas

The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.

Use the following import convention:

```
>>> import pandas as pd
```

> Pandas Data Structures

Series

A one-dimensional labeled array capable of holding any data type

```
in> s = pd.Series([1, -1, 2, 4], index=[10, 15, 20, 25])
```

Dataframe

A two-dimensional labeled data structure with columns of potentially different types

```
columns => Country Capital Population  
index => 10 15 20 25  
  
in> data = {'Country': ['Brazil', 'India', 'China'],  
           'Capital': ['Brasília', 'New Delhi', 'Beijing'],  
           'Population': [2000000000, 1300000000, 1400000000]}  
in> df = pd.DataFrame(data,  
                     columns=['Country', 'Capital', 'Population'])
```

> Dropping

```
in> df.drop(['Country'], axis=1)  
in> df.drop('Country', axis=0)
```

> Asking For Help

```
in> help(pd.Series)
```

> Sort & Rank

```
in> df.sort_index() sort by labels along an axis  
in> df.sort_values('Country') sort by the values along an axis  
in> df.rank() Assign ranks to entries
```

Read and Write to Excel

```
in> pd.read_excel('file.xlsx')  
in> df.to_excel('file.xlsx', sheet_name='Sheet1')  
  
Read multiple sheets from the same file  
  
in> data = pd.read_excel('file.xlsx', sheet_name='Sheet1')  
in> df = pd.read_excel(data, 'Sheet2')
```

Read and Write to SQL Query or Database Table

```
in> from sqlalchemy import create_engine  
in> engine = create_engine('sqlite:///memory:')  
in> df.to_sql('SELECT * FROM countries', engine)  
in> df.read_sql_table('my_table', engine)  
in> df.read_sql_query('SELECT * FROM my_table', engine)  
read_sql() is a convenience wrapper around read_sql_table() and read_sql_query()  
in> df.to_sql('mydb', engine)
```

> Selection

Getting

The .at[] method allows us to access
Country Capital Population

1 Brazil Brasília 2000000000

2 India New Delhi 1300000000

3 China Beijing 1400000000

Selecting, Expanding, Shrinking

By Position

```
in> df.at[10] #Select single value by row & column index
```

```
in> df.iat[10, 1] #Select single value by row & column index
```

```
in> df.iat[10, 1] #Select single value by row & column index
```

```
in> df.iat[10, 1] #Select single value by row & column index
```

```
in> df.iat[10, 1] #Select single value by row & column index
```

```
in> df.iat[10, 1] #Select single value by row & column index
```

By Label/Position

```
in> df.iat[0] #Select single row or subset of rows
```

Country Brazil
Capital Brasília

Population 2000000000

```
in> df.iat[1, 'Capital'] #Select a single column of subset of rows
```

1 Brasília

2 New Delhi

3 Beijing

```
in> df.iat[1, 'Capital'] #Select rows and columns
```

1 New Delhi

Boolean Indexing

```
in> df[(x < 10)] #Selects a where value is not 10
```

```
in> df[(x < 10) | (x > 20)] #Selects value is not 10 or 20
```

```
in> df[df['Population']>1000000000] #Use filter to select DataFrame
```

1 New Delhi

Setting

```
in> df['x'] = 6 #Set index or of Series or to a
```

```
in> df.sum() Summarize numerical columns  
in> df.info() Info on DataFrame  
in> df.count() Number of non-null values
```

Summary

```
in> df.agg() #List of values  
in> df.agg().sum() #Summation of values  
in> df.agg('mean').min() #Minimum/maximum values  
in> df.agg('count').max() #Maximum/minimum index value  
in> df.describe() #Summary statistics  
in> df.mean() #Mean of values  
in> df.median() #Median of values
```

> Applying Functions

```
in> f = lambda x: x**2  
in> df.apply(f) #Apply function  
in> df.applymap(f) #Apply function element-wise
```

> Data Alignment

Data Alignment

The align() method is used to align the index of the left frame with the index of the right frame. This can be done in the index that don't overlap.

```
in> df1 = pd.DataFrame([[1, 2, 3], [4, 5, 6]], index=[10, 15], columns=[1, 2, 3])
```

1 2 3

4 5 6

10 15

1 2 3

The you also do the inverse operation yourself with the help of the fill methods.

1 2 3

4 5 6

10 15

1 2 3

4 5 6

10 15

1 2 3

4 5 6

10 15

1 2 3

4 5 6

10 15

1 2 3

4 5 6

10 15

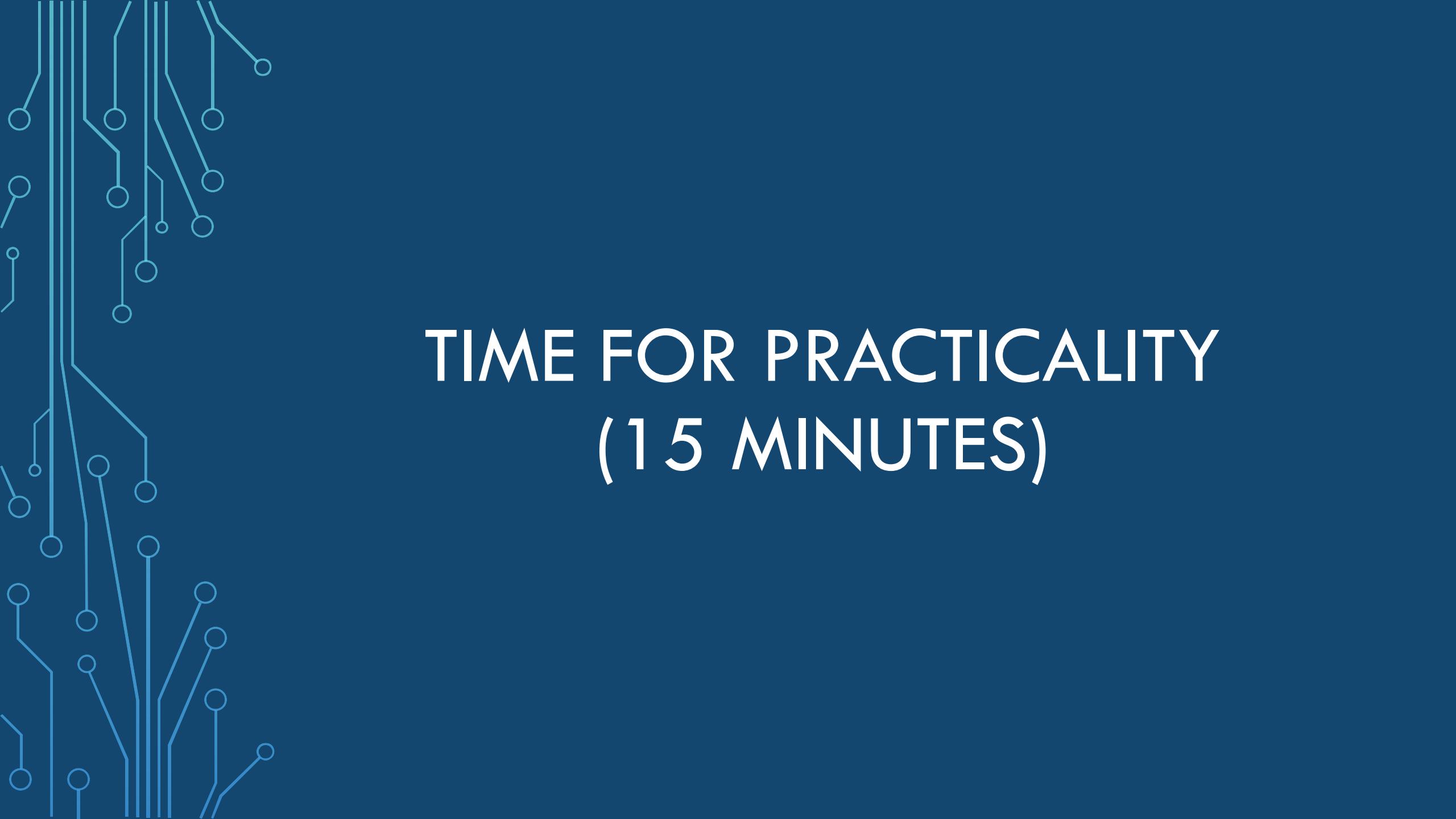
1 2 3

4 5 6

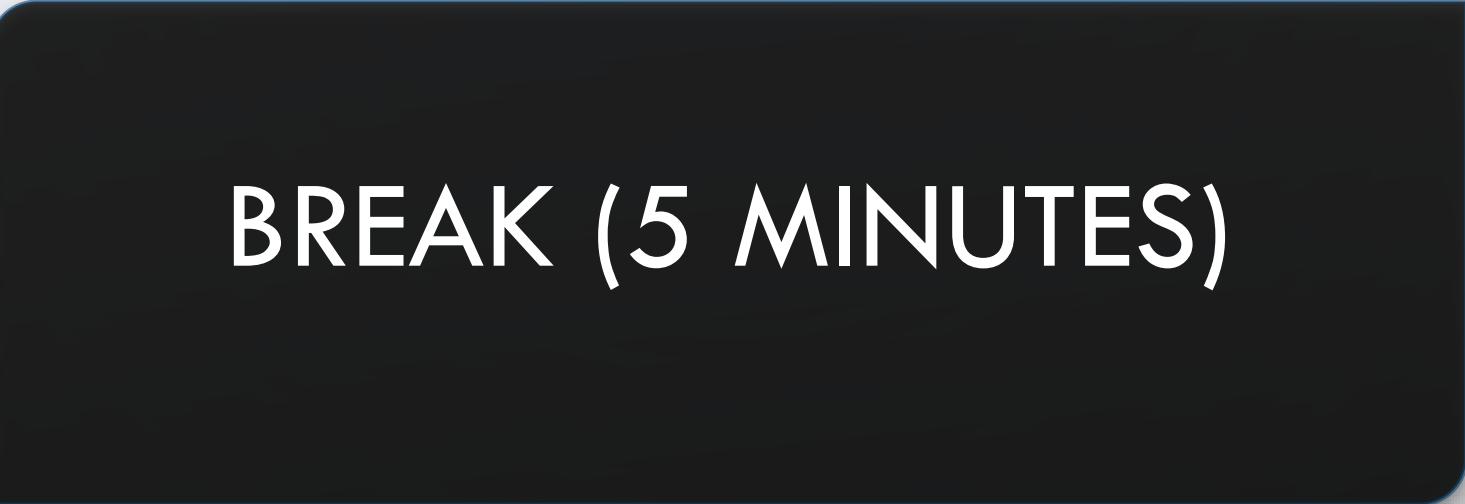
10 15



https://images.datacamp.com/image/upload/v1676302204/Marketing/Blog/Pandas_Cheat_Sheet.pdf



TIME FOR PRACTICALITY
(15 MINUTES)



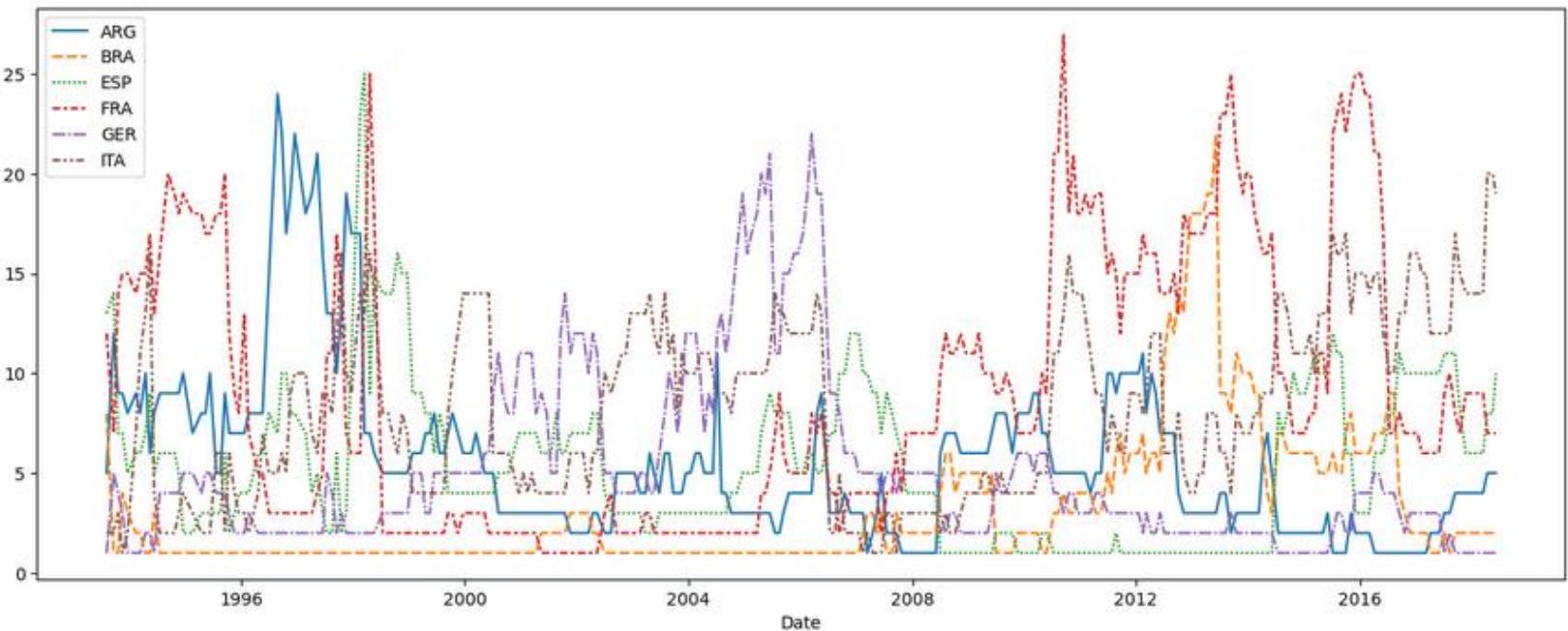
BREAK (5 MINUTES)

- A picture is worth a thousand words
- For exploratory data analysis
- Communicate data clearly
- Share unbiased representation of data
- Support recommendations to different stakeholders

WHY VISUALIZATION ?

THE IMPORTANT OF VISUALIZATION

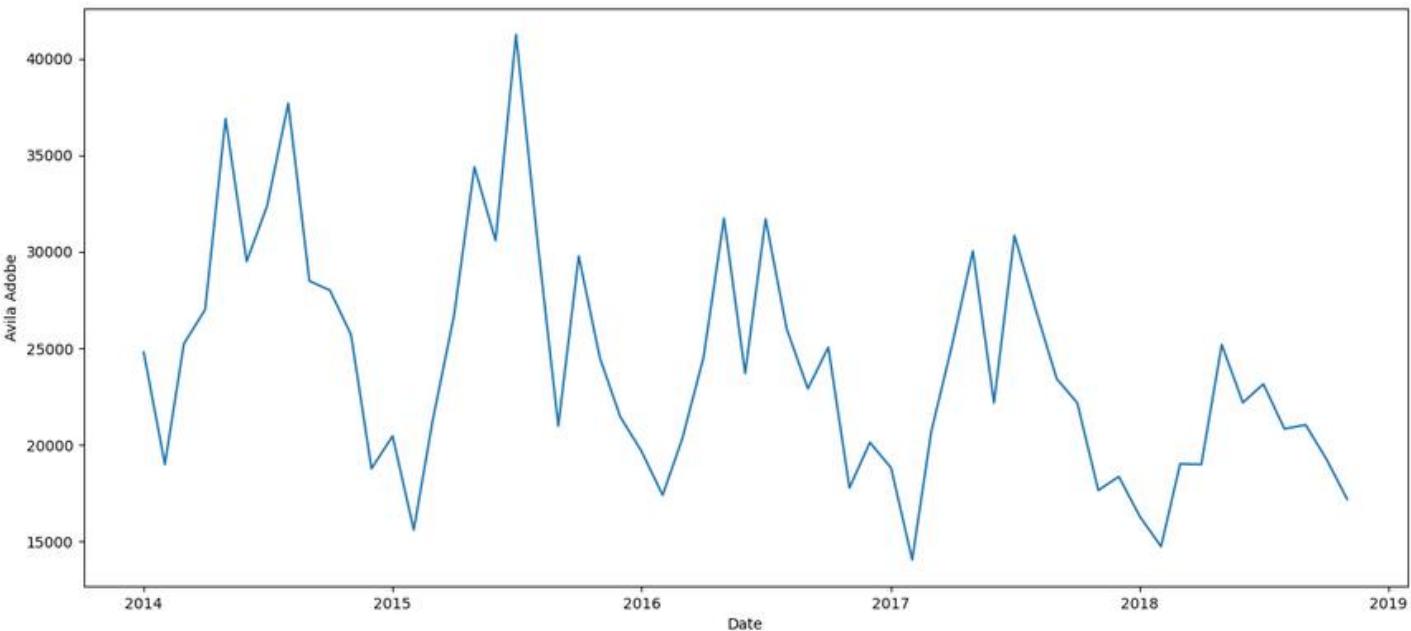
- Considering only the years represented in the figure, which countries spent at least 5 consecutive years in the #1 ranked spot?



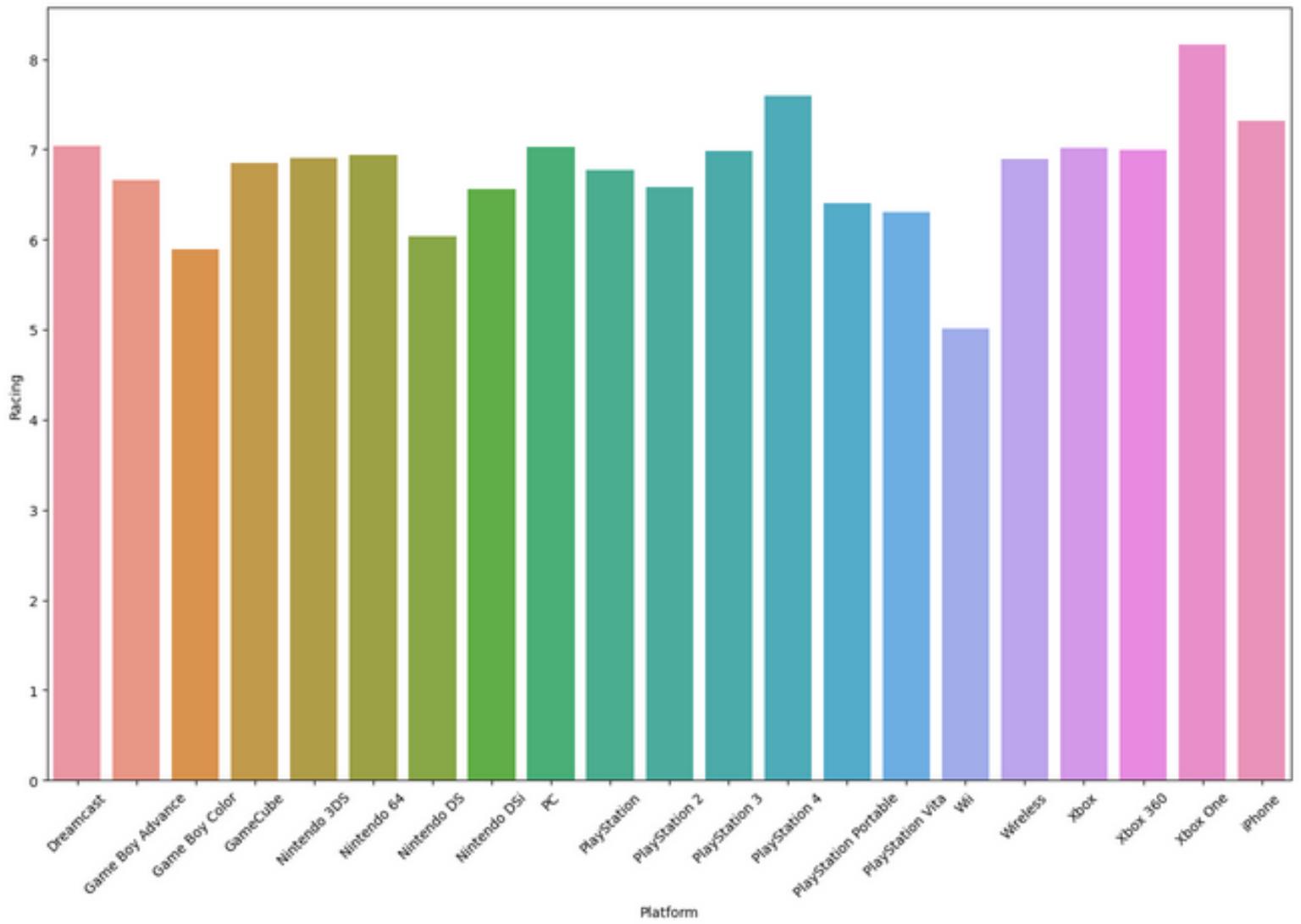
Does Avila Adobe get more visitors:

- **in September-February (in LA, the fall and winter months), or**
- **in March-August (in LA, the spring and summer)?**

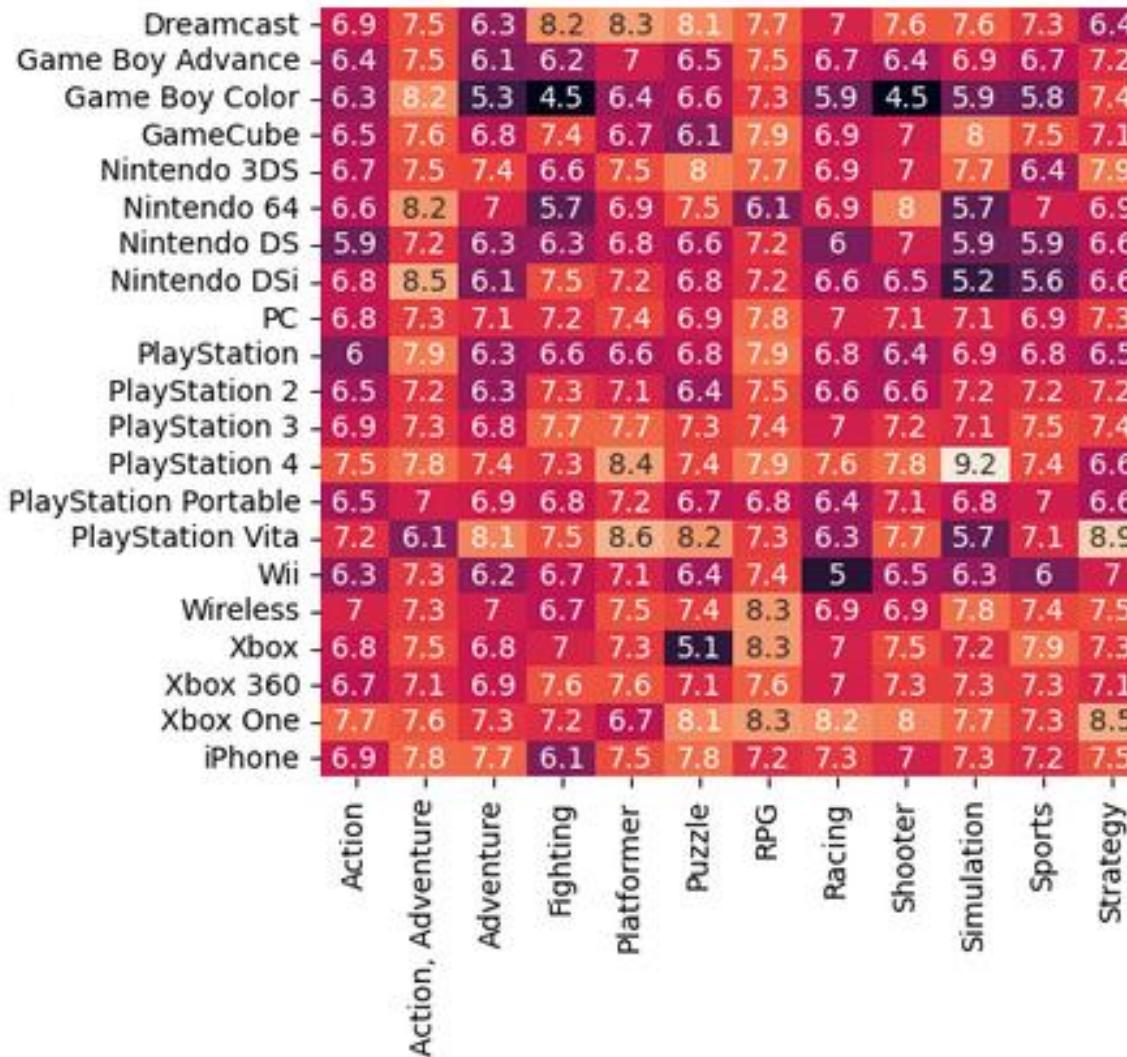
Using this information, when should the museum staff additional seasonal employees?



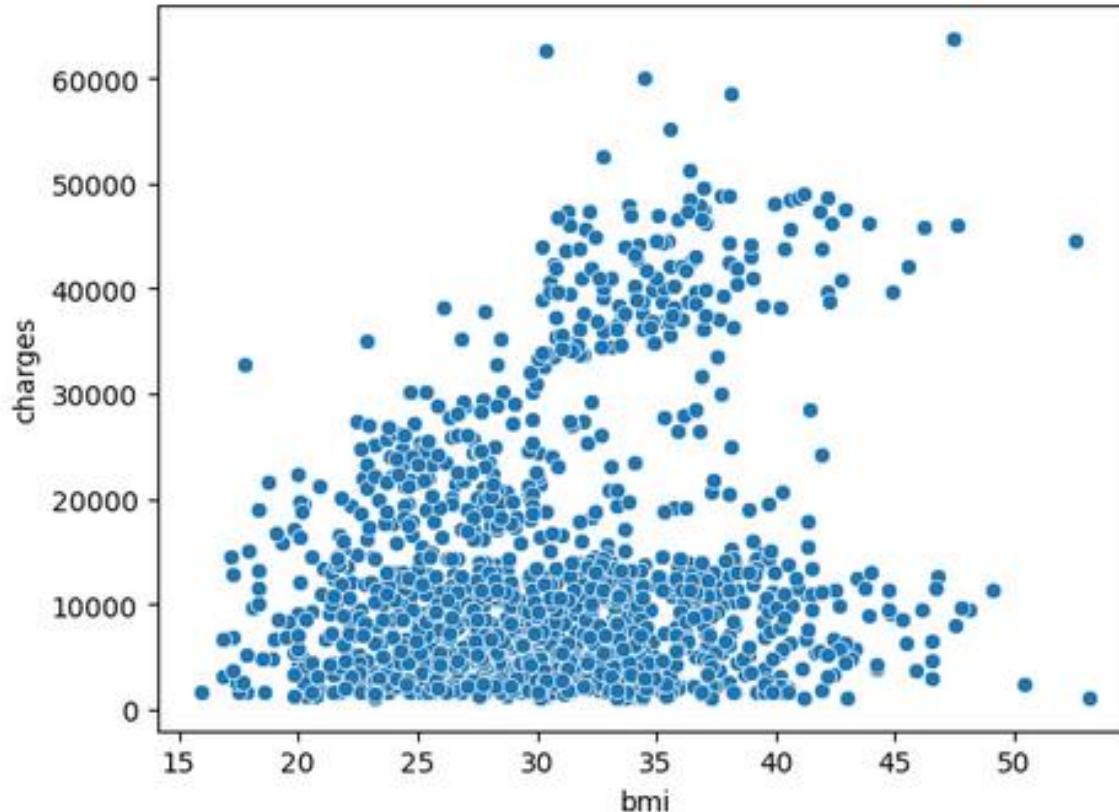
- Based on the bar chart, do you expect a racing game for the Wii platform to receive a high rating? If not, what gaming platform seems to be the best alternative?



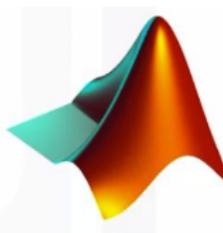
- Which combination of genre and platform receives the highest average ratings? Which combination receives the lowest average rankings?



**IS THERE ANY
RELATION
BETWEEN BODY
MASS INDEX AND
INSURANCE
CHARGES ?**



- Neurobiologist
- Part of a team analyzing **Electrocorticography Signals (ECOG)**
 - **Electrocorticography** is the process of recording electrical activity in the brain
- The team
 - used a proprietary software (**MATLAB** based version) for analysis
 - had only one license and were taking turns in using it
- John replace the proprietary software with **Matplotlib**



MATPLOTLIB

PYTHON MATPLOTLIB

- MatLab-style Plotting Library
- Created in 2002
- Most popular data visualization library in Python
- Well supported in different environments
 - Python scripts & iPython shell & web app servers & **Jupyter Notebook**
- Originally developed as an **ECoG** visualization tool

matplotlib

> Prepare The Data

1D Data

```
>>> import numpy as np  
>>> x = np.linspace(0, 10, 100)  
>>> y = np.cos(x)  
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))  
>>> data2 = 3 * np.random.random((10, 10))  
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]  
>>> U = -1 - X**2 + Y  
>>> V = 1 + X - Y**2  
>>> from matplotlib.cbook import get_sample_data  
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

PREPARE THE DATA

> Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()  
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

Axes

All plotting is done with respect to an *Axes*. In most cases, a *subplot* will fit your needs.
A subplot is an axes on a grid system.

```
>>> fig.add_axes()  
>>> ax1 = fig.add_subplot(221) #row-col-num  
>>> ax3 = fig.add_subplot(212)  
>>> fig3, axes = plt.subplots(nrows=2, ncols=2)  
>>> fig4, axes2 = plt.subplots(ncols=3)
```

CREATE PLOT

> Show Plot

```
>>> plt.show()
```

SHOW PLOT

PLOTTING ROUTINES

> Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()  
>>> lines = ax.plot(x,y) #Draw points with lines or markers connecting them  
>>> ax.scatter(x,y) #Draw unconnected points, scaled or colored  
>>> axes[0,0].bar([1,2,3],[3,4,5]) #Plot vertical rectangles (constant width)  
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2]) #Plot horizontal rectangles (constant height)  
>>> axes[1,1].axhline(0.45) #Draw a horizontal line across axes  
>>> axes[0,1].axvline(0.65) #Draw a vertical line across axes  
>>> ax.fill(x,y,color='blue') #Draw filled polygons  
>>> ax.fill_between(x,y,color='yellow') #Fill between y-values and g
```

Data Distributions

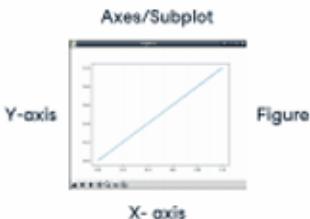
```
>>> ax1.hist(y) #Plot a histogram  
>>> ax3.boxplot(y) #Make a box and whisker plot  
>>> ax3.violinplot(z) #Make a violin plot
```

DISTRIBUTIONS

WORKFLOW (PIPELINE OF PLOTTING)

> Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

- 1 Prepare Data
 - 2 Create Plot
 - 3 Plot
 - 4 Customized Plot
 - 5 Save Plot
 - 6 Show Plot
- ```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4] #Step 1
>>> y = [18,20,25,38]
>>> fig = plt.figure() #Step 2
>>> ax = fig.add_subplot(111) #Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3) #Step 3, 4
>>> ax.scatter([2,4,6],
 [5,15,25],
 color='darkgreen',
 marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png') #Step 5
>>> plt.show() #Step 6
```

## > Plotting Cutomize Plot

### Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
 cmap='seismic')
```

# CUSTOMIZATION

## Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

# MARKERS

## Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

# LINESTYLES

## MathText

```
>>> plt.title(r'$\sigma_i=15$', fontsize=20)
```

# MATHTEXT

# LIMITS, LEGENDS AND LAYOUTS

## Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1) #Add padding to a plot
>>> ax.axis('equal') #Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5]) #Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5) #Set limits for x-axis
```

## Legends

```
>>> ax.set(title='An Example Axes', #Set a title and x-and y-axis labels
 ylabel='Y-Axis',
 xlabel='X-Axis')
>>> ax.legend(loc='best') #No overlapping plot elements
```

## Ticks

```
>>> ax.xaxis.set(ticks=range(1,5), #Manually set x-ticks
 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y', #Make y-ticks longer and go in and out
 direction='inout',
 length=10)
```

## Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5, #Adjust the spacing between subplots
 hspace=0.3,
 left=0.125,
 right=0.9,
 top=0.9,
 bottom=0.1)
>>> fig.tight_layout() #Fit subplot(s) in to the figure area
```

## Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

### Prepare The Data

#### 1D Data

```
arr = np.array([1, 2, 3])
x = np.linspace(0, 10, 100)
y = np.sin(x)
z = np.zeros(10)
```

#### 2D Data or Images

```
img = np.random.rand(10, 10)
img[0, 0] = 0.1
img[0, 1] = 0.2
img[0, 2] = 0.3
img[0, 3] = 0.4
img[0, 4] = 0.5
img[0, 5] = 0.6
img[0, 6] = 0.7
img[0, 7] = 0.8
img[0, 8] = 0.9
img[0, 9] = 1.0
```

### Create Plot

#### Figure

```
fig = plt.Figure()
fig = plt.Figure(figsize=(10, 10))
```

#### Axes

An axes is a rectangular region in an area in your figure in which all the data is plotted on both axes.

```
ax = fig.add_subplot()
ax = fig.add_subplot(111, frame_on=False)
ax = fig.add_subplot(111, frame_on=False)
ax = fig.add_subplot(111, frame_on=False)
ax = fig.add_subplot(111, frame_on=False)
```

### Save Plot

```
fig.savefig('my_figure.png')
```

### Show Plot

```
plt.show()
```

## 2D Data

```
df = pd.read_csv('iris.csv')
df = df.sample(100)
df['species'].unique()
df['species'].value_counts()
df['species'].value_counts().plot(kind='bar')
df['species'].value_counts().plot(kind='bar', color='red')
df['species'].value_counts().plot(kind='bar', color='red', title='Iris Species Distribution')
df['species'].value_counts().plot(kind='bar', color='red', title='Iris Species Distribution', legend=False)
```

## Vector Fields

```
df = pd.read_csv('iris.csv')
df = df.sample(100)
df['species'].unique()
df['species'].value_counts()
df['species'].value_counts().plot(kind='bar')
df['species'].value_counts().plot(kind='bar', color='red')
df['species'].value_counts().plot(kind='bar', color='red', title='Iris Species Distribution')
df['species'].value_counts().plot(kind='bar', color='red', title='Iris Species Distribution', legend=False)
```

# DATA CAMP CHEAT SHEET (MATPLOTLIB)

## Plot Anatomy

Figure

## Workflow

The steps involved in creating plots with Matplotlib are:

- Import Data
- Create Plot
- Fix
- Communicate Plot
- Save Plot
- Close Plot

```
import matplotlib.pyplot as plt
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y)
plt.title('Sine Wave')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
plt.close()
```

### Close and Clear

## Markers

```
marker='o', ms=10, mew=1, mfc='r'
marker='o', ms=10, mew=1, mfc='r', mfc='r'
marker='o', ms=10, mew=1, mfc='r', mfc='r', mfc='r'
```

## LineStyles

```
linestyle='solid', linewidth=2
linestyle='solid', linewidth=2
linestyle='solid', linewidth=2
linestyle='solid', linewidth=2
linestyle='solid', linewidth=2, label='solid'
linestyle='solid', linewidth=2, label='solid'
```

## Text & Annotations

```
text('Hello', 10, 10)
```

## Margins

```
margin=0.1
```

## Limits, Legends and Layouts

### Direct Accessing

```
ax.set_xlim(0, 10) # Set the x-axis limit to 10
ax.set_ylim(0, 10) # Set the y-axis limit to 10
ax.set_xscale('log') # Set the x-axis scale to logarithmic
ax.set_yscale('log') # Set the y-axis scale to logarithmic
```

### Layout

```
fig = plt.figure()
fig = plt.figure(figsize=(10, 10))
fig = plt.figure(figsize=(10, 10), dpi=100)
fig = plt.figure(figsize=(10, 10), dpi=100, facecolor='white')
fig = plt.figure(figsize=(10, 10), dpi=100, facecolor='white', edgecolor='black')
fig = plt.figure(figsize=(10, 10), dpi=100, facecolor='white', edgecolor='black', frameon=True)
```

### Annotations

```
ax.annotate('Hello', (10, 10), color='red', fontweight='bold')
```

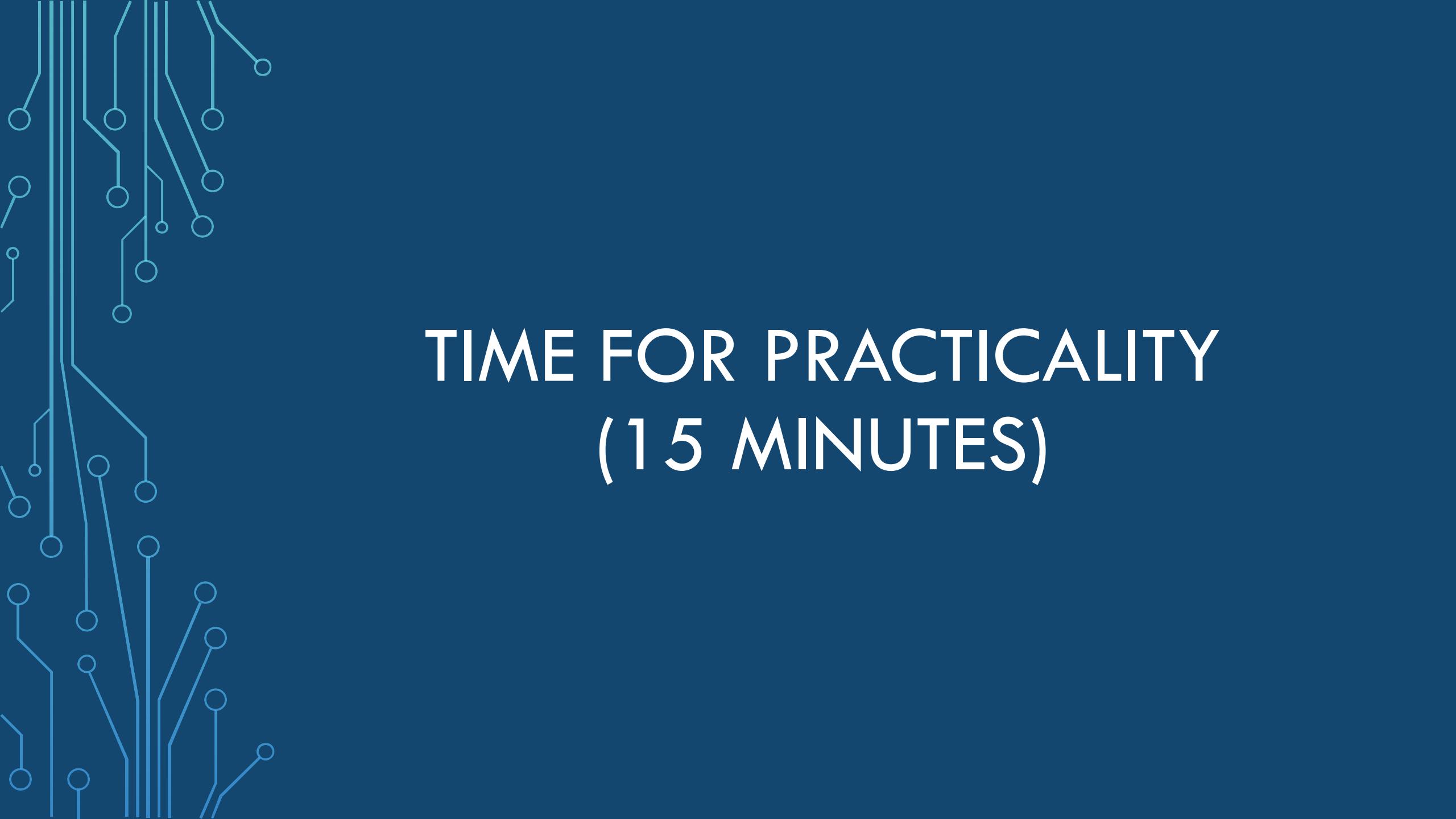
```
ax.annotate('Hello', (10, 10), color='red', fontweight='bold', rotation=45, xytext=(15, 15))
ax.annotate('Hello', (10, 10), color='red', fontweight='bold', rotation=45, xytext=(15, 15), arrowprops=dict(facecolor='black'))
```

### Script Syntax

```
import matplotlib, numpy as np
matplotlib.rcParams['font.size'] = 10
matplotlib.rcParams['font.family'] = 'serif'
matplotlib.rcParams['font.serif'] = 'Times New Roman'
matplotlib.rcParams['font.sans-serif'] = 'Arial'
matplotlib.rcParams['font.monospace'] = 'Ubuntu Mono'
matplotlib.rcParams['font.bold'] = 'bold'
matplotlib.rcParams['font.italic'] = 'italic'
matplotlib.rcParams['font.variant'] = 'normal'
matplotlib.rcParams['font.weight'] = 'normal'
matplotlib.rcParams['font.stretch'] = 'normal'
matplotlib.rcParams['font.size'] = 10
matplotlib.rcParams['font.family'] = 'serif'
matplotlib.rcParams['font.serif'] = 'Times New Roman'
matplotlib.rcParams['font.sans-serif'] = 'Arial'
matplotlib.rcParams['font.monospace'] = 'Ubuntu Mono'
matplotlib.rcParams['font.bold'] = 'bold'
matplotlib.rcParams['font.italic'] = 'italic'
matplotlib.rcParams['font.variant'] = 'normal'
matplotlib.rcParams['font.weight'] = 'normal'
matplotlib.rcParams['font.stretch'] = 'normal'
```

### Axis Spines

```
ax.spines['left'].set_color('red')
ax.spines['left'].set_color('red')
ax.spines['left'].set_color('red')
ax.spines['left'].set_color('red')
```



TIME FOR PRACTICALITY  
(15 MINUTES)

# SUMMER TRAINING PROJECT



Choose whether to be in groups  
(Teams) or individual



Try to implement the project in  
**INCREMENTAL WAY**



Starting with the next session  
(Exploratory Data Analysis, Data  
Cleaning and Preprocessing)



Whenever the Machine Learning  
Algorithms will be discussed;  
implement your required algorithm  
on your workflow and so on.



The criteria for the project  
evaluation will be explained next  
session



Prepare your ideas ...



# QUESTIONS



THANK YOU