Lecture Notes/Tutorials by Dr. Bilgic @RIT

Intro to Python and JupyterLab

Topics: Simply What Numpy, Pandas, and Matplotlib Packs are!

Your Task:

1) Review the code, comments

2) Practice in a new notebook

Table of Contents

What is Jupyter-Lab?

- Visit main resource about Jupyter-Lab
- Browse the lab features: buttons, left area, contextual help, pip, create new
- Click on Help >> JupyterLab Reference >> Starting
- Shortcuts

What is Numpy?

What is:

- NumPy is a Python library used for working with arrays, but they are slow to process.

- It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
- NumPy stands for Numerical Python.
- NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.
- The source code for NumPy is located at this github repository https://github.com/numpy/numpy.

Import the numpy with notebook and terminal:

Notebook:

> import numpy as np

Terminal:

> conda install numpy
>
> !pip install numpy

Practices

In [ ]:
```
# Python evaluates the code
print("Hello Universe")

i=0
while i<5:
    print(i)
    i+=1
```

In [ ]:
```
# Create an alias (np) with the as keyword while importing
import numpy as np

# Which version
print(np.__version__)
```

In [ ]:
```
# Create an array
import numpy as np #redandant to run every time. one time is enough (see the ab

# Assign the array to arr 1-D array (vector)
arr = np.array( [1, 2, 3, 4, 5] )

# Print the object
print(arr)
```

```
# Type of the object
print(type(arr))
```

In [ ]:
```
# Create a 0-D array with value 42

arr = np.array(42)

print(arr)
```

In [ ]:
```
# Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6 (mat

arr = np.array( [ [1, 2, 3], [4, 5, 6] ] )

print(arr)
```

In [ ]:
```
# Create a 3-D array with two 2-D arrays, both containing two arrays with the v

arr = np.array( [ [[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]] ] )

print(arr)
```

In [ ]:
```
# Check how many dimensions the arrays have:

a = np.array(42)
b = np.array([1, 2, 3, 4, 5]) #1-D,vector
c = np.array([[1, 2, 3], [4, 5, 6]]) #2-D,matrix
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]) #3-D

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

In [ ]:
```
# Get the first element from the following array:

arr = np.array([1, 2, 3, 4])

print(arr[0]) #try: 0, 2, -1
```

In [ ]:
```
# Access the 2nd element on 1st dim:

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st dim: ', arr[0, 1])
```

In [ ]:
```
print(arr)
```

In [ ]:
```
# Question

#print('4nd element on 2st dim: ', arr[?, ??]) #4th column, 2nd row
#print('4nd element on 2st dim: ', arr[1, 3]) #4th column, 2nd row
#print('4nd element on 2st dim: ', arr[1, -2]) #4th column, 2nd row
print('4th element on 2nd dim: ', arr[-1, -2]) #4th column, 2nd row
```

In [ ]:
```
# Slice elements from index 1 to index 5 from the following array:

import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7]) #1 is index 0.

print(arr[1:5]) #see 6 is not included
```

In [ ]:
```
# Slice elements from index 4 to the end of the array:

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[4:])
```

In [ ]:
```
# Slice elements from the beginning to index 4 (not included):

import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[:4])
```

> Keep practicing on it from W3Schools. We will revisit Numpy.
>
> A Task: Visit Source for more info and practice.

What is Pandas?

What is:

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.
- The source code for Pandas is located at this github repository https://github.com/pandas-dev/pandas.

- Pandas are fast, also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Import the pandas with notebook and terminal:

Notebook:

> import pandas as pd

Terminal:

> conda install pandas
>
> pip install pandas

Practices

In [ ]:
```
# Create an alias (np) with the as keyword while importing
import pandas as pd

# Which version
print(pd.__version__)
```

In [ ]:
```
# Make dictionary data a Data Frame
# The keys of the dictionary become the labels
import pandas as pd

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pd.DataFrame(mydataset)

print(myvar)
```

In [ ]:
```
# Create a simple Pandas Series from a list:

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

In [ ]:
```
# Create labels

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)

#Return the value of "y":
print(myvar["y"])
```

In [ ]:
```
print(myvar[2]) #index starts from 0
```

In [ ]:
```
# Create a simple Pandas DataFrame:

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

In [ ]:
```
# Pandas use the loc attribute to return one or more specified row(s)
#refer to the row index:
print(df.loc[0]) #what is .iloc?
```

In [ ]:
```
# Return row 0 and 1:
#use a list of indexes:
print(df.loc[[0, 1]])
```

In [ ]:
```
# A useful command
# Check the current directory
!pwd
```

In [ ]:
```
# Load a comma separated file (CSV file) into a DataFrame:
# Post train-titanic.csv under this notebook's folder.
# Check the path of the data set: Mine data set is

import pandas as pd

#df_csv = pd.read_csv('./1.DataAnalytics/2. Data Analysis and Visualization wit
df_csv = pd.read_csv('train-titanic.csv',
                     index_col="PassengerId") #from Kaggle. index is Passengeri

print(df_csv.tail()) #try: head(), tail()
```

α

Six codes that we should use after data set is imported

In [ ]:
```
#df_csv.shape #dim
#df_csv.info() #types and all missing,
#df_csv.head(10) #first 10 observation,
#df_csv.tail(10) #last 10 observation,
#df_csv.describe() #show only numerical summaries,
df_csv.describe(include='all') #include all variables including numerical and c
```

In [ ]:
```
# Info about Data Set
print(df_csv.info()) #try: head(), tail()
```

In [ ]:
```
# Load the CSV into a DataFrame:
import pandas as pd
```

```
#df_csv.to_string()
```

In [ ]:
```
# Return a new Data Frame with no empty cells:
# This is not a good practice. Run, but don't do!
# Remove all rows with NULL values: If you want to change the original DataFram

df_csv.dropna(inplace = True)

#print(df_csv.to_string())
```

In [ ]:
```
# We lost the rows that have at least one NA
# This is not a good practice. Don't do!
print(df_csv.info()) #try: head(), tail()
```

In [ ]:
```
# Print the replaced data set. lost rows. not good practice
print(df_csv.info()) #try: head(), tail()
```

In [ ]:
```
# Recover the data set
df_csv = pd.read_csv('train-titanic.csv')

print(df_csv.info()) #try: head(), tail()
```

Let's check the dim and first ten observations

In [ ]:
```
print(df_csv.head(10))
```

In [ ]:
```
# Dim of the data
df_csv.shape
```

In [ ]:
```
df_csv.head(10)
```

In [ ]:
```
# Copy the data set, and play with missing imputation
# Replace NULL values with the number 130: just demo, not useful

df_play = df_csv.copy()

df_play.fillna(130, inplace = True)

print(df_play.info())

# Print first 10
print(df_play.head(10))
```

In [ ]:
```
# Again copy
# Replace NULL values in the "Age" columns with the number 30:
df_play = df_csv.copy()

df_play["Age"].fillna(30, inplace = True)

# Print first 10
print(df_play.head(10))
```

In [ ]:
```
# Again copy
# Calculate the MEAN, and replace any empty values with it:
```

```
df_play = df_csv.copy()

x = df_play["Age"].mean()

df_play["Age"].fillna(x, inplace = True)

print(df_play.head(10))

#try .mean() .median() for Quantitative data; .mode() for Qualitative data
```

In [ ]:
```
# Calculate the MODE, and replace any empty values with it:
df_play = df_csv.copy()

y = df_play["Cabin"].mode()[0]

df_play["Cabin"].fillna(y, inplace = True)

print(df_play.head(10))
```

Data Reorganization, Missingness and Manipulation with Pandas

In [ ]:
```
# How to rename column names
# Visit https://towardsdatascience.com/21-pandas-operations-for-absolute-beginn

df = pd.DataFrame(data={'a':[1,1,3,4,5],
                        'b':[0,0,5,10,15]})

new_df = df.rename({'a':'new_a','b':'new_b'})
```

In [ ]:
```
print(new_df)
```

In [ ]:
```
# How to get column names in a list?

col = df.columns.tolist()

print(col)
```

In [ ]:
```
# How to get the frequency of values in a series?
# Just syntax is below
# Need to adjust the code for the data

df[col].value_counts() #returns a mapper of key,frequency pair

#df[col].value_counts()[key] #to get frequency of a key value
```

In [ ]:
```
# How to reset an index to an existing column or another list or array?

new_df = df.reset_index(drop=True,inplace=False)
```

In [ ]:
```
# Your turn
# Define list_of_cols, col_name, list_of_cols_to_drop

list_of_cols = ['a','b', 'c']

col_name = 'a'
```

```
list_of_cols_to_drop =['b','c']

# then run the following codes
```

In [ ]:
```
# How to remove a column?
# define list_of_cols_to_drop
list_of_cols_to_drop = 'a'

df.drop(columns = list_of_cols_to_drop)
```

In [ ]:
```
# How to change the index in a data frame?

df.set_index('b',inplace=True)

print(df)
```

In [ ]:
```
# How to remove rows or columns if they have nan values?

df.dropna(axis=0, inplace=True)
```

Let's go back to the titanic data set.

In [ ]:
```
# How to slice a data frame given a condition?
# Just illustration, don't run

df = pd.read_csv('train-titanic.csv')

mask = df['age'] == age_value

#or
mask = df['age'].isin(list_of_age_values)

result = df[mask]
```

In [ ]:
```
# With multiple conditions, choose rows

mask = (df['age']==age_value) & (df['height'] == height_value)

result = df[mask]
```

- How to slice a data frame given names of columns or index values of rows? Four ways:

a) df.iat[1,2] provides the element at 1th row and 2nd column. Here it's important to note that number 1 doesn't correspond to 1 in index column of dataframe. It's totally possible that index in df does not have 1 at all. It's like python array indexing.

b) df.at[first,col_name] provides the value in the row where index value is first and column name is col_name

c) df.loc[list_of_indices,list_of_cols] eg df.loc[[4,5],['age','height']] slices dataframe for matching indices and column names

d) df.iloc[[0,1],[5,6]] used for interger based indexing will return 0 and 1st row for 5th and 6th column.

In [ ]:
```
# How to sort by a column?

df.sort_values(by = list_of_cols, ascending=True)
```

- How to apply a function to each element to a series?

df['series_name'].apply(f)

where f is the function you want to apply to each element of the series. If you also want to pass arguments to the custom function, you could modify it like this.

def f(x,**kwargs):

```
   #do_somthing
   return value_to_store
```

df['series_name'].apply(f, a= 1, b=2,c =3)

If you want to apply a function to more than a series, then:

def f(row): age = row['age'] height = row['height']

df[['age','height']].apply(f,axis=1)

If you don't use axis=1, f will be applied to each element of both the series. axis=1 helps to pass age and height of each row for any manipulation you want.

In [ ]:
```
# How to apply a function to all elements in a data frame?

new_df = df.applymap(f)
```

In [ ]:
```
# How to slice a data frame if values of a series lie in a list?

df[df['age'].isin(age_list)]

# the opposite: data samples where age does not lie in the list use

df[~df['age'].isin(age_list)]
```

In [ ]:
```
# How to group-by column values and aggregate over another column or apply a fu

df.groupby(['age']).agg({'height':'mean'})

# want to group-by a certain column and convert all the corresponding grouped e
df.groupby(['age']).agg(list)
```

In [ ]:
```
# How to concatenate two data frames?
#df1 --> name,age,height
```

```
#df2---> name,age,height

#the result data-frame will have the columns appended from the data-frames
result = pd.concat([df1,df2],axis=0)

#For horizontal concatenation
#df1--> name,age
#df2--->height,salary
result = pd.concat([df1,df2], axis=1)
```

- How to merge two data frames?

For the previous example, assume you have an employee database forming two dataframes like

df1--> name, age, height df2---> name, salary, pincode, sick_leaves_taken

You may want to combine these two dataframe such that each row has all details of an employee. In order to acheive this, you would have to perform a merge operation.

df1.merge(df2, on=['name'], how='inner')

This operation will provide a dataframe where each row will comprise of name, age, height, salary, pincode, sick_leaves_taken. how = 'inner' means include the row in result if there is a matching name in both the data frames.

- For more info on merge, read.

More Data Manipulation with Pandas

Change data types

In [ ]:
```
# creating a DataFrame
df = pd.DataFrame({'srNo': [1, 2, 3],
                   'Name': ['Geeks', 'for',
                            'Geeks'],
                   'id': [111, 222,
                          333]})
# show the dataframe
print(df)

# show the datatypes
print(df.dtypes)
```

In [ ]:
```
# changing the dataframe
# data types to string
df = df.astype(str)

# show the data types
# of dataframe
df.dtypes
```

In [ ]:
```
# Convert the data type of “grade” column from “float” to “int”.
# dictionary
result_data = {'name': ['Alia', 'Rima', 'Kate',
                        'John', 'Emma', 'Misa',
                        'Matt'],
        'grade': [13.5, 7.1, 11.5,
                  3.77, 8.21, 21.22,
                  17.5],
        'qualify': ['yes', 'no', 'yes',
                    'no', 'no', 'yes',
                    'yes']}

# create a dataframe
df = pd.DataFrame(result_data)

# show the datatypes
print(df.dtypes)

# convert data type of grade column
# into integer
df.grade = df.grade.astype(int)

# show the dataframe
print(df)

# show the datatypes
print(df.dtypes)
```

In [ ]:
```
# We can pass pandas.to_numeric, pandas.to_datetime and pandas.to_timedelta
# as argument to apply() function to change the datatype of one or more columns
# to numeric, datetime and timedelta respectively.

# sample dataframe
df = pd.DataFrame({
    'A': ['a', 'b', 'c',
          'd', 'e'],
    'B': [12, 22, 35,
          '47', '55'],
    'C': [1.1, '2.1', 3.0,
          '4.1', '5.1'] })

# show the datatypes
print(df.dtypes)

# show the dataframe
print(df)

# using apply method
df[['B', 'C']] = df[['B', 'C']].apply(pd.to_numeric)

# show the data types
# of all columns
df.dtypes
```

In [ ]:
```
# using numpy
x = np.array([1, 2, 2.5])

x.dtype
```

In [ ]:
```
x.astype(int) #int, object, float64
```

In [ ]:
```
# from num to booleen
#run ? to read manual
#?
```

In [ ]:
```
# replace
df = pd.DataFrame({'A': [0, 1, 2, 3, 4],
                   'B': [5, 6, 7, 8, 9],
                   'C': ['a', 'b', 'c', 'd', 'e']})
print(df)

df.replace(0, 5)

print(df)
```

In [ ]:
```
# from num to string
d = {'col1': [1, 2, 3], 'col2': [4, 5, 6]}

print(d)

df = pd.DataFrame(d)

print(df.dtypes)

print(df)

df2 = df.to_string()

print(df2)
```

Obtain only Categorical columns

In [ ]:
```
df.head()
```

In [ ]:
```
# Need adjustment
categorical = df.dtypes[df.dtypes == 'object'].index
print(categorical)
df[categorical].describe()
```

Change labels

> ### Keep practicing on it from W3Schools.
>
> A Task: Visit Source for more info and practice. Complete.
>
> For more applications, visit 21 Pandas Operations

What is Matplotlib?

What is:

- Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
- Visit the main resource
- Matplotlib is open source and we can use it freely.
- Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
- Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely.
- Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.
- The source code for Matplotlib is located at this github repository https://github.com/matplotlib/matplotlib.

pyplot

- Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:
- Now the Pyplot package can be referred to as plt.

The link for the practices below- Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

- Now the Pyplot package can be referred to as plt.

The link for the practices below

Import the matlibplot with notebook and terminal:

Notebook:

```
import matplotlib
```

Terminal:

```
conda install matplotlib
```

```
!pip install matplotlib
```

Practices

In [ ]:
```
import matplotlib

print(matplotlib.__version__)
```

In [ ]:
```
# An idea how a plot can be obtained

import matplotlib.pyplot as plt

plt.plot([10, 20, 30, 40], 'b*') #try: bs, b*, b-, b--, ..

plt.ylabel('Some Numbers')

plt.show()
```

In [ ]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'bo')
```

In [ ]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'b*')
plt.axis([0, 6, 0, 20])
plt.show()
```

In [ ]:
```
t = np.arange(0., 5., 0.2)
plt.plot(t, t**3, 'r--') #add , t, t*2, 'b^' inside
```

In [ ]:
```
# np.arange command
print(np.arange(0., 5., 0.2)) # from 0 to 5 (not included 5), by .2
```

In [ ]:
```
import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
# plot(x,y)
plt.plot(t, 10-t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

In [ ]:
```
# Plotting categorical variables
names = ['group_a', 'group_b', 'group_c'] #x
values = [10, 20, 50] #y

plt.figure(figsize=(9, 3))

plt.subplot(131)
plt.bar(names, values)

plt.subplot(132)
plt.scatter(names, values)

plt.subplot(133)
```

```
plt.plot(names, values)

plt.suptitle('Categorical Plotting')
plt.show()
```

In [ ]:
```
# Skip
# Plot a function

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

In [ ]:
```
# Skip
# Working with text
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=1, facecolor='g', alpha=0.75)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')

plt.text(50, .01, r'$\mu=100,\ \sigma=15$') #math expresssion

plt.axis([40, 160, 0, 0.03])

plt.grid(True)
plt.show()
```

In [ ]:
```
# Annotating the text

import matplotlib.pyplot as plt

ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black', shrink=0.1) )

plt.ylim(-2, 2) #plt.xlim(-0,5)
plt.suptitle('More prof look')

plt.show()
```

%matplotlib inline

- When using the 'inline' backend, your matplotlib graphs will be included in your notebook, next to the code.
- %matplotlib inline will lead to interactive plots embedded within the notebook AND lead to static images of your plot embedded in the notebook.
- After running this command (it needs to be done only once per kernel/session), any cell within the notebook that creates a plot will embed a PNG image of the resulting graphic:

Details:

- %matplotlib inline sets the backend of matplotlib to the 'inline' backend: With this backend, the output of plotting commands is displayed inline within frontends like the Jupyter notebook, directly below the code cell that produced it. The resulting plots will then also be stored in the notebook document.

> ### Keep practicing on it from W3Schools. We will revisit Matplotlib.
>
> A Task: Visit Source for more info and practice. Complete

References

- Shortcuts Cheatsheet for Jupyter-lab
- Simple Shortcuts for Jupyter-lab
- W3Schools

Your Notes:

...

In [ ]: