MIS 740: Software Concepts
Fall 2022

Reading and Writing Files

Purpose

- Understand how a Python program handles path and file
- Learn how to read the content of a text file
- Lean how to write data to a text file

1. Preparation

(1) Please download **13_lab files.zip** from WebCampus. Unzip the files and import them to Jupyter Notebook.

2. Read File Content

(2) This program reads the content of a file and show it on the screen.

First, import the **os** module

```
# import the os module
import os
```

(3) We use a while loop here to make sure the user enters a text file. The **endswith**() function can be used to check file extension.

```
# Check whether the file is a txt file
# is not, repeat the while loop
while not fileName.endswith('.txt'):
    # prompt the user to enter a txt file
    print('Please enter the name of a text file: ')
    # assign the user's input to a variable
    fileName = input()
```

(4) Then, verify whether the file exists before open the file.

```
# verify whether the file exists
if not os.path.exists(fileName):
    print('The file does not exist.')
# The file exists
else:
    # open the file
```

(5) If the file exists, open, read, and close the file. Then print the content.

In this example, we first try to read the entire file as a string.

```
# The file exists
else:
    # open the file
    inFile = open(fileName)

    # read the file content as a string
    fileContent = inFile.read()

    # close the file
    inFile.close()

    # print the string
    print(fileContent)
```

(6) When you run the test the program, you can see that the entire file is printed as a very long string.

(7) Please revise the program to read the content into a list.

```
# The file exists
else:
    # open the file
    inFile = open(fileName)

    # read the file content into a list
    # each line will be an item in the list
    fileContent = inFile.readlines()

    # close the file
    inFile.close()

    # print the string
    print(fileContent)
```

(8) Run the program and compare the outcome.

3. Degree Courses

(9) In this program, the user will enter names of the courses he/she is taking this semester. Write the input value to a file.

Use the file name entered by the user to open the file, in append mode.

```
# promot the user to enter the file name
print('Please enter the file you\'d like to save the data.')
fileName = input()

# open the file, in append mode
courseFile = open(fileName, 'a')
```

(10) Use a **for** loop to get all the courses. Please note that the loop should end at **numOfCourses+1** (exclusive)

```python
# ask the user how many couses he/she is taking
print('How many courses are you taking?')
numOfCourses = int(input())

# use a for loop to get all the course names
for i in range (1, numOfCourses+1):
    # ask for the course name
    print('Please enter the name of course '+ str(i)+': ')
    courseName = input()

```

(11) Write the course name to the file.

```python
# use a for loop to get all the course names
for i in range (1, numOfCourses+1):
    # ask for the course name
    print('Please enter the name of course '+ str(i)+': ')
    courseName = input()

    # write to the file
    courseFile.write(courseName)
    # write a new line character
    courseFile.write('\n')

```

(12) Close the file once all the courses are entered.

```python
# use a for loop to get all the course names
for i in range (1, numOfCourses+1):
    # ask for the course name
    print('Please enter the name of course '+ str(i)+': ')
    courseName = input()

    # write to the file
    courseFile.write(courseName)
    # write a new line character
    courseFile.write('\n')

# after all the courses are entered, close the file
courseFile.close()

```

Exercise: Movie Record

Please write a program that reads the movie record from a file (MovieBoxOffice.txt) and shows the content to the user. The user can add a record by entering a move name and its box office.

```
Please enter the name of a text file:
D:\backup\Fall2019_MIS740\MovieBoxOffice.txt
Avengers: Endgame                                       858,373,000
The Lion King                                           543,187,716
Toy Story 4                                             433,804,674
Captain Marvel                                          426,829,839
Spider-Man: Far from Home                               390,532,085
Aladdin                                                 355,559,216
Joker                                                   277,931,557
It Chapter Two                                          210,758,107
Us                                                      175,005,930
Fast & Furious Presents: Hobbs & Shaw                   173,788,145
John Wick: Chapter 3 - Parabellum                       171,015,687
How to Train Your Dragon: The Hidden World              160,799,505
The Secret Life of Pets 2                               158,257,265
Pokemon Detective Pikachu                               144,105,346
Once Upon a Time in Hollywood                           140,428,363


Please enter the name of the movie you would like to add:
Gemini Man
Please enter the box office
856,895
file saved.
```

MIS 740: Software Concepts
Fall 2022

Numpy (1)

Purpose

- Understand how to create an array
- Practice how to traverse an array
- Apply array arithmetic

1. Preparation

(1) Please download **14_lab files.zip** from WebCampus. Unzip the files and import them to Jupyter Notebook.

2. Grade Curving

(2) This program reads a file with scores and applied an equation to curve the grade. After getting the file name from the user, open and read the file content.

In this program, we will read the entire file as a string

```python
# open the file and assign the file object
gradeFile = open(fileName)
# read the content as a whole to a string variable
content = gradeFile.read()
# close the file
gradeFile.close()
```

(3) Then split the string using \n as the separator to get a list.

Use the list to create a NumPy array, and designate the datatype as float16.

```python
# split the string with \n as the separator, into a list
# use the list to create a NumPy array, convert the datatype to float16
gradeArray = np.array(content.split('\n'), dtype = 'float16')
```

(4) We can always print the first few element to see whether the operation is correct. Please uncomment lines 20-21.

```python
19 # print the first 5 scores to see whethe it is correct
20 print('The first five score in the file')
21 print(gradeArray[:5])
```

(5) Apply the arithmetic ufunc to curve the grade. The equation is taking a square root and multiply by 10.

```
# apply the Arithmetic ufunc to curve the grade
curvedArray = np.sqrt(gradeArray)*10
```

(6) Again, we can print out the first few curved score to see whether everything is correct.

Please uncomment line 27, and add line 28.

```
26 # print the first 5 scores to see whethe it is correct
27 print('The first five curved score')
28 print(curvedArray[:5])
```

(7) We can now print the original score and the new score side by side.

```
# print the original score and the new score side-by-side
for i in range(gradeArray.size):
    #use i to index the item in the two arrays.
    # with comma to separate the items from the two arrays
    print(str(gradeArray[i])+', '+str(curvedArray[i]))
```

(8) Now, you can save and test the program.

Exercise: Height Converter

Please write a program that read a file with some heights in centimeters. Please convert the heights into feet and inches. The result contains the original and converted height should be written to a new file and separated with a comma.

MIS 740: Software Concepts
Fall 2022

Numpy (2)

Purpose

- •Apply array broadcasting
- •Get familiarize with Boolean Array, logical operation, and masking

1. Preparation

(1) Please download **15_lab files.zip** from WebCampus. Unzip the files and import them to Jupyter Notebook.

2. Average Height of US Presidents

(2) This program reads the height data from a csv file and show the statistics.

The program starts with reading the file and create an array.

(3) Then, we can apply the aggregation function of Numpy to show the statistics

```
# Use the aggregations to show the statistics of the height
print("Mean height:        ", heights.mean())
print("Standard deviation:", heights.std())
print("Minimum height:     ", heights.min())
print("Maximum height:     ", heights.max())
print("25th percentile:    ", np.percentile(heights, 25))
print("Median:             ", np.median(heights))
print("75th percentile:    ", np.percentile(heights, 75))
```

(4) The program also uses matplotlib to visualize the data. We will cover matplotlib later in the class.

Exercise: Grade Curing

In the cervedGrade.csv file, the first column is the original scores and the second column shows the curved scores. Use the curvedGrade.csv and show how the curving changes the distribution of the scores, including the min, max, mean, standard deviation, and the median.

3. Curving a Series of Scores

(5) The program reads the original scores from a file and then ask the user to enter the percentage of curving he/she wants to apply.

The program then prints the updated score.

NOTE: This program uses pandas that we will cover later in this class

(6) The data has been read from the csv file. You can uncomment line 12 and run the program to see a sample of the data.

```
1  # import pandas, set the alias as pd
2  import pandas as pd
3  # import numpy, set the alias as np
4  import numpy as np
5
6  # read the csv file to read the data
7  data = pd.read_csv('originalScores.csv')
8  # use the entire data frame create a numpy array
9  scores = np.array(data[:])
10
11 # print the first five rows to see whether it looks right.
12 print(scores[:5])
```

(7) We need an array to represent the curving percentage. It will contain the curving percentage of the four tests. To declare a one-dimensional array and fill it with ones, please enter the code:

```
13
14 # declare an one dimensional array representing the curving percentage
15 percentages = np.ones((4), dtype='float')
16 print (percentages)
```

You can also uncomment line 16 and run the program to verify the content of the new array.

(8) Next, we will ask the user to enter the percentages. Since we want to set values for the elements in an array, we can use a **for** loop to traverse the array.

Please make sure you uncomment lines 19-20. You can also uncomment line 26 to see the updated array.

```
18 # promt the user to enter the curving percentage for the four tests
19 for i in range (4):
20     print('Please enter the curving percentage (%) for Test '+str(i+1)+': ')
21     # convert the input to float, add to 1
22     # then assign the value to the corresponding element in the array
23     # there is only one
24     percentages[i] = float(input())/100+1
25
26 # print (percentages)
```

(9) Now we need to multiply the curving percentage and the original score. Then print the calculation result.

```
28 # calculate teh updated score
29 # The one-dimensional array a is broadcast across the second dimension
30 # in order to match the shape of scores
31 updatedScores = scores*percentages
32
33 # print the curved scores
34 print('The scores after curving are: ')
35 print(updatedScores)
```

4. Counting Rainy Days

(10) The program read a file containing a series of data that represents the amount of precipitation each day for a year in a given city.

NOTE: This program uses pandas and matplotlib that we will cover later in this class.

(11) We can use the **np.sum()** function to count the number of days (counting the Boolean Array) for certain conditions

```
# show some statistics
print("Number days without rain:      ", np.sum(inches == 0))
print("Number days with rain:         ", np.sum(inches != 0))
print("Days with more than 0.5 inches:", np.sum(inches > 0.5))
print("Rainy days with < 0.2 inches  :", np.sum((inches > 0) &
                                                (inches < 0.2)))

```

(12) Now run and test the program.

MIS 740: Software Concepts
Fall 2022

Numpy (3)

Purpose

- Get familiarize with array operation and masking
- Practice the creating of structured array and data type

1. Preparation

(1) Please download **16_lab files.zip** from WebCampus. Unzip the files and import them to Jupyter Notebook.

2. Counting Summer Rainy Days

(2) The program read a file containing a series of data that represents the amount of precipitation each day for a year in a given city.

NOTE: This program uses pandas and matplotlib that we will cover later in this class.

(3) We can create mask to define rainy days and summer days.

```

# construct a Boolean array (mask) of all rainy days
rainy =(inches > 0)

# construct a oolean array (mask) of all summer days
# (June 21st is the 172nd day)
# (September 19th is the 262nd day)
days = np.arange(365)
summer = (days > 172) & (days <262)

```

(4) Then, use the two masks individually or together to filter/ mask the data for only data we need.

Please note that in the last two, we apply the Boolean operator for two conditions.

```
# show median of rainy days
print("Median precip on rainy days in 2014 (inches):   ", np.median(inches[rainy]) )
# show median of summer days
print("Median precip on summer days in 2014 (inches):  ", np.median(inches[summer]) )
# show max of summer days
print("Maximum precip on summer days in 2014 (inches): ",np.max(inches[summer])  )
# show media of summer rany dayes
print("Median precip on summer rainy days (inches):", np.median(inches[rainy & summer]) )
# show media of non-summer rany dayes
print("Median precip on non-summer rainy days (inches):", np.median(inches[rainy & ~summer]) )

```

3. Loyal Customer Lookup

(5) This program read the customer data from a file. The user can enter a criterion to filter customer data by the loyalty points earned.

First, define the data type of the structured array.

```
7  # Define the Structured Array Data Types, with name, email, and points
8  customerDType = np.dtype([('name', 'U10'), # unicode 10-character string
9                            ('email', 'U50'), # unicode 50-character string
10                           ('points', 'i4')])# 4-bit integer
11
```

(6) Declare the structured array and specify the size and data type.

```
12 # count the number of rows in the data
13 recordCount = len(rawData)
14 # create the structured array,
15 # size is the recordCount
16 # data type is the compound data type define at lines 8-10
17 data = np.zeros(recordCount, dtype=customerDType)
```

(7) Once the structured array is declared, we can assign the data to this structured array.

```
18 # first column of the csv file is assigned as the value of 'name' column
19 data['name'] = rawData[:,0]
20 # second column of the csv file is assigned as the value of 'email' column
21 data['email'] = rawData[:,1]
22 # third column of the csv file is assigned as the value of 'points' column
23 data['points']=rawData[:,2]
```

(8) After ask the user for the criterion, we can then filter the data by applying the Boolean mask.

```
34 # filter the data and save to a new array
35 # data['points'] >= criterion is the Boolean mask
36 # retreive the 'email' only
37 result = data[data['points'] >= criterion]['email']
38
```

(9) To get the number of customers in the result, we can apply the **len()** function.

```
39 # get the number of records in the result
40 resultCount = len(result)
41 # show number of records to the user
42 print("Number of customers found: "+ str(resultCount))
```

(10) If the user chooses to save the result, write each email in the result to the file with a for loop.

```
    # write each email in the result array to the file,
    # with a new line character
    for email in result:
        outFile.write(email+"\n")
    # close the file
    outFile.close()
    # show confirm to the user
    print('File saved to '+fileName)
```

(11) You can run and test the program.

MIS 740: Software Concepts
Fall 2022

Pandas (1)

Purpose

- Practice the creation of a DataFrame object from a file
- Practice selection and indexing of DataFrame
- Practice operation of DataFrame

1. Preparation

(1) Please download **17_lab files.zip** from WebCampus. Unzip the files and import them to Jupyter Notebook.

2. Demographic Statistics

(2) The program read the data from a CSV file into a DataFrame and show the statistics.

You can open the file **Demographic_Statistics_By_Zip_Code.csv** to see the structure.

(3) First, use the read_csv() function to read the entire file.

You can show the DataFrame to see what it looks like. The default index is assigned.

```
# read the csv file for the data
data = pd.read_csv('Demographic_Statistics_By_Zip_Code.csv')

data
```

Out[45]:

	JURISDICTION NAME	FEMALE	MALE	GENDER UNKNOWN	PACIFIC ISLANDER	HISPANIC LATINO	AMERICAN INDIAN	ASIAN NON HISPANIC	WHITE NON HISPANIC	BLACK NON HISPANIC	OTHER ETHNICITY	ETHNICITY UNKNOWN	PERMANENT RESIDENT ALIEN
0	10001	22	22	0	0	16	0	3	1	21	3	0	2
1	10002	19	16	0	0	1	0	28	6	0	0	0	2
2	10003	1	0	0	0	0	0	1	0	0	0	0	0
3	10004	0	0	0	0	0	0	0	0	0	0	0	0
4	10005	2	0	0	0	0	0	1	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
231	12788	39	44	0	0	0	0	0	81	0	2	0	0
232	12789	115	157	0	0	0	0	0	262	0	6	4	1
233	13731	2	15	0	0	0	0	0	15	0	2	0	0
234	16091	0	0	0	0	0	0	0	0	0	0	0	0
235	20459	0	0	0	0	0	0	0	0	0	0	0	0

236 rows × 19 columns

(4) Instead of the default index, we would like to use the column 'JURISDICTION NAME' as the index.

```
# set the index to 'JURISDICTION NAME'
data = data.set_index('JURISDICTION NAME')

data

```

Out[3]:

JURISDICTION NAME	FEMALE	MALE	GENDER UNKNOWN	PACIFIC ISLANDER	HISPANIC LATINO	AMERICAN INDIAN	ASIAN NON HISPANIC	WHITE NON HISPANIC	BLACK NON HISPANIC	E
10001	22	22	0	0	16	0	3	1	21	
10002	19	16	0	0	1	0	28	6	0	
10003	1	0	0	0	0	0	1	0	0	
10004	0	0	0	0	0	0	0	0	0	
10005	2	0	0	0	0	0	1	0	1	
...	...	...	...	...	...	...	...	...	...	
12788	39	44	0	0	0	0	0	81	0	
12789	115	157	0	0	0	0	0	262	0	
13731	2	15	0	0	0	0	0	15	0	
16091	0	0	0	0	0	0	0	0	0	
20459	0	0	0	0	0	0	0	0	0	

236 rows × 18 columns

(5) In order to show the first five rows, use the **iloc** to apply the implicit indexing to slice the data.

```
# show the first 5 rows of data
firstFive = data.iloc[:5, :]
```

(6) To get only the residency data, use the **loc** to slice the data by the column name.

```
# show the data for residency.
# i.e., between the columns 'PERMANENT RESIDENT ALIEN' and 'CITIZEN STATUS UNKNOWN'
residency = data.loc[:, 'PERMANENT RESIDENT ALIEN':'CITIZEN STATUS UNKNOWN']
```

(7) We can also apply the masking operation.

Define the mask first and then use it in the selection.

```
# apply mask on the column 'RECEIVES PUBLIC ASSISTANCE'
# show the number of male and female for districts
# that have more than 20 people receiving public assistance
mask = data['RECEIVES PUBLIC ASSISTANCE'] > 20
receivingAssistance = data[mask].loc[:,'FEMALE':'MALE']
```

(8) To make sure the masking result is correct, we want to add the column 'RECEIVES PUBLIC ASSISTANCE' as well. In order to do so, we can apply fancy indexing.

```
# apply fancy index to get the columns of 'FEMALE', 'MALE' and 'RECEIVES PUBLIC ASSISTANCE'
colomnsToSee = ['FEMALE', 'MALE', 'RECEIVES PUBLIC ASSISTANCE']
receivingAssistance = data[mask].loc[:,colomnsToSee]
# receivingAssistance
```

(9) To add a new key-value pair (i.e., adding a new column), we can specify the column name and the calculate the value.

```
# add a new key-value pair by calculating the total
# the new key is 'Total' (i.e., column name)
# and the new value is the calculation result
data['total'] = data['FEMALE']+ data['MALE']

# calculate the percentage of female and male
# add the key-value pairs
data['percentage female'] = data['FEMALE']/data['total']
data['percentage male'] = data['MALE']/data['total']
```

MIS 740: Software Concepts
Fall 2022

Pandas (2)

Purpose

- Handling missing data in a DataFrame object from a file

1. Preparation

(1) Please download **18_lab files.zip** from WebCampus. Unzip the files and import them to Jupyter Notebook.

2. Medical Tracking Data

(2) The program read a csv file containing the data for a medical experiment. Several thousands of people received a treatment and came back for 37 tests. The test results were recorded. There are missing data in the file.

(3) The file is read by using the read_csv() function.

You can show the DataFrame to see what it looks like. The default index is assigned.

```python
# read the csv file for the data
allData = pd.read_csv('longitudinal_tracking_scores.csv')
allData

```

	ID	Group	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8	...	Test28	Test29	Test30	Test31	Test32	Test33
0	5320069	Adult	76.0	97.0	72.0	73.0	72.0	93.0	81.0	69.0	...	85.0	63.0	69.0	97.0	89.0	97.0
1	5320254	Adult	97.0	69.0	79.0	74.0	77.0	82.0	62.0	70.0	...	68.0	72.0	97.0	94.0	64.0	90.0
2	5320276	Adult	89.0	85.0	86.0	72.0	89.0	60.0	71.0	87.0	...	66.0	96.0	63.0	95.0	92.0	61.0
3	5320323	Adult	86.0	92.0	80.0	64.0	82.0	92.0	80.0	73.0	...	99.0	98.0	86.0	68.0	98.0	62.0
4	5320335	Adult	99.0	67.0	72.0	100.0	94.0	77.0	81.0	98.0	...	64.0	89.0	97.0	86.0	96.0	81.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2576	5559538	Teen	79.0	80.0	78.0	67.0	82.0	80.0	74.0	71.0	...	68.0	79.0	89.0	88.0	64.0	85.0
2577	5559626	Teen	76.0	68.0	NaN	90.0	73.0	98.0	73.0	62.0	...	62.0	83.0	65.0	76.0	91.0	100.0
2578	5559714	Teen	100.0	97.0	97.0	81.0	96.0	75.0	NaN	84.0	...	66.0	68.0	70.0	92.0	97.0	63.0
2579	5559718	Teen	92.0	74.0	65.0	95.0	77.0	70.0	90.0	91.0	...	78.0	70.0	61.0	64.0	61.0	95.0
2580	5559790	Teen	99.0	95.0	70.0	97.0	62.0	70.0	79.0	98.0	...	84.0	66.0	98.0	88.0	100.0	65.0

2581 rows × 39 columns

(4) We can check how many participants were missing (not returning to take the test) for the first test.

```python
# read the csv file for the data
allData = pd.read_csv('longitudinal_tracking_scores.csv')
# allData

# How many missing values for test1?
print(sum(allData['Test1'].isnull()))

```

(5) Follow the same logic, we can find out how many participants are missing from each test. We can use a for loop to iterate through each column.

```
# How many missing values for test1?
print(sum(allData['Test1'].isnull()))

# How many missing values for each test?
for i in range(1,38,1):
    print('Missing data for Test'+str(i)+': ', sum(allData['Test'+str(i)].isnull()))
```

(6) We also would like to find out on average, how many times a participant missed the test. We can sum the missing count by rows, and add a new column to the data frame to record the number.

Then, based on the new column, get the maximum value.

```
# How many missing values for each individual?
allData['missing_count']=allData.isnull().sum(axis=1)

# See the max of missing coount for individuals
# print('Count of missing values for individuals')
print('Max: ', allData['missing_count'].max())
```

(7) If we want to find out how many participant has ever missed a test, we can create a mask for missing_count !=0 to do so.

```
# How many individuals with missing values?
# Define a mask for missing_count !=0
mask = allData['missing_count']!=0
print('Number of participants with missing value: ', mask.sum())
```

(8) We can get some statistics on the missing count. For example, see the average missing count and the median.

```
# How many individuals with missing values?
# Define a mask for missing_count !=0
mask = allData['missing_count']!=0
print('Number of participants with missing value: ', mask.sum())
# for individuals with missing values, what's the median and avarage missing count?
print('For individuals with missing values')
print('Mean: ', allData[mask]['missing_count'].mean())
print('Median: ', np.median(allData[mask]['missing_count']))
```

(9) If the participants missed some tests, on average, they would miss two tests.

Therefore, we would like to keep the data with at least 20 valid values.

```
# Keep individuals with more than 20 valid values
cleanedData = data.dropna(axis='rows', thresh=20)
```

(10) Finally, we can write the cleaned dataset to a csv file, by using the **to_csv()** method.

```
# write the cleaned result to a file
cleanedData.to_csv('cleanedData.csv')
```

(11) You can open the new file to see the result.

3. Vaccination Data

(12) The program read a csv file containing number of people received vaccinations (in thousands). There are missing data in the file.

(13) First, use the read_csv() function to read the entire file.

You can show the DataFrame to see what it looks like. The default index is assigned.

```
# import pandas, set the alias as pd
import pandas as pd
# import numpy, set the alias as np
import numpy as np

# read the csv file for the data
data = pd.read_csv('2017 Vaccinations (thousand).csv')

data
```

	Jurisdiction	Black Non-Hispanic	Hispanic	White Non-Hispanic	Asian Non-Hispanic
0	Allegany	NaN	NaN	32.6	NaN
1	Anne Arundel	37.3	NaN	48.9	NaN
2	Baltimore City	40.8	NaN	46.2	NaN
3	Baltimore County	40.6	NaN	54.0	NaN
4	Calvert	NaN	NaN	51.5	NaN
5	Caroline	NaN	NaN	31.6	NaN
6	Carroll	NaN	NaN	50.7	NaN

(14) We can set the Jurisdiction as the index of the DataFrame:

```
# set the index to the first column
data = data.set_index('Jurisdiction')
```

(15) Try out the **dropna**() function to drop rows with any missing value. By doing so, we can get rows that do not have missing values.

```
# is there any county does not have missing values?
completeData = data.dropna(how='any', axis = 'rows')
completeData
```

	Black	Hispanic	White	Asian
Jurisdiction				
Montgomery	39.5	38.0	65.3	54.5

(16) Similarly, use the **dropna**() function to get records with only one missing value (i.e., three valid values).

```
# is there any county does not have missing values?
completeData = data.dropna(how='any', axis = 'rows')

# countries with only one missing value?
particalData = data.dropna(thresh=3, axis="rows")
particalData
```

	Black	**Hispanic**	**White**	**Asian**
Jurisdiction				
Montgomery	39.5	38.0	65.3	54.5
Prince George's	38.5	38.5	50.7	NaN

(17) When we attempt to calculate the total population, the result would appear to be NaN, given the missing values in the columns.

```
# countries with only one missing value?
particalData = data.dropna(thresh=3, axis="rows")

# calculate the total
data['total'] = data['Black']+ data['Hispanic']+data['White']+data['Asian']
data
```

	Black	**Hispanic**	**White**	**Asian**	**total**
Jurisdiction					
Allegany	NaN	NaN	32.6	NaN	NaN
Anne Arundel	37.3	NaN	48.9	NaN	NaN
Baltimore City	40.8	NaN	46.2	NaN	NaN
Baltimore County	40.6	NaN	54.0	NaN	NaN
Calvert	NaN	NaN	51.5	NaN	NaN
Caroline	NaN	NaN	31.6	NaN	NaN

(18) Thus, we need to fill the data with 0 before we can calculate the total.

```python
# countries with only one missing value?
particalData = data.dropna(thresh=3, axis="rows")

# fill the missing data with 0
data = data.fillna(0)

# calculate the total
data['total'] = data['Black']+ data['Hispanic']+data['White']+data['Asian']
data
```

Jurisdiction	Black	Hispanic	White	Asian	total
Allegany	0.0	0.0	32.6	0.0	32.6
Anne Arundel	37.3	0.0	48.9	0.0	86.2
Baltimore City	40.8	0.0	46.2	0.0	87.0
Baltimore County	40.6	0.0	54.0	0.0	94.6

(19) Now we can show the maximum, minimum, and average number of people get vaccination in the jurisdicton.

```python
# now, we can print the min, max, average of the data
print("Max: "+ str(np.max(data['total'])))
print("Min: "+ str(np.min(data['total'])))
print("Average: "+ str(np.mean(data['total'])))
```

```
Max: 197.3
Min: 31.6
Average: 74.91666666666667
```

(20) Applying mask operation, we can also show the name of counties with the most and least people vaccinated.

```python
# create masks for min and mix
maskMax = data['total']==np.max(data['total'])
maskMin = data['total']==np.min(data['total'])

# use the mask to filter data. Use index attitbue to print the attribute.
# the index attribute will return an integer array. Get the first (and only) index
print('The county with the most people vaccinated is: '+data[maskMax].index[0])
print('The county with the least people vaccinated is: '+data[maskMin].index[0])
```

Exercise: BMI for US Presidents

Write a program that reads **president_heights_updated.csv**, and shows the names of US presidents with the highest and lowest BMI from the dataset.

The formula is **BMI = kg/m^2** where kg is a person's weight in kilograms and m is their height in meters

MIS 740: Software Concepts
Fall 2022

Pandas (3)

Purpose

- Practice combing datasets

1. Preparation

(1) Please download **19_lab files.zip** from WebCampus. Unzip the files and import them to Jupyter Notebook.

2. US States Data

(2) The program reads data from three csv files, representing data from different sources.

In the program, the data will be combined/merged into a single DataFrame. The program will rank US states and territories by their 2010 population density.

(3) At lines 8-10, the three files are read into the three DataFrames.

We can start with merging **pop** and **abbrevs** to get the full name of the states.

Since different column keys are used, we need to use the **left_on** and **right_on** keyword.

```
12 # Many-to-one merge that will give us the full state name within the pop DataFrame
13 #  merge based on the state/region column of pop, and the abbreviation column of abbrevs
14 merged = pd.merge(pop, abbrevs, how='inner',
15                   left_on='state/region', right_on='abbreviation')
```

(4) After merge, the two of the same column appear. We can drop one.

```
16 # drop duplicate info
17 merged = merged.drop('abbreviation', axis="columns")
18
```

(5) Then, merge with the **areas** DataFrame. We can do a left join to make sure all of the states remain in the DateFrame even if some area data are missing.

```
19 # merge the result with the area data
20 # merge based on the state column, and use the pop DataFrame as the main
21 # i.e., if the data are missing in areas DataFrame, NaN will be inserted
22 final = pd.merge(merged, areas, on='state', how='left')
23
```

(6) If you look into the data set, it contains data from different years and age group.

```
# merge the result with the area data
# merge based on the state column, and use the pop DataFrame as the main
# i.e., if the data are missing in areas DataFrame, NaN will be inserted
final = pd.merge(merged, areas, on='state', how='left')

final
```

	state/region	ages	year	population	state	area (sq. mi)
0	AL	under18	2012	1117489.0	Alabama	52423
1	AL	total	2012	4817528.0	Alabama	52423
2	AL	under18	2010	1130966.0	Alabama	52423
3	AL	total	2010	4785570.0	Alabama	52423
4	AL	under18	2011	1125763.0	Alabama	52423

(7) In order to answer the question of "2010 population density," we need to select only the needed data.

We can create a mask to filter the data.

Or, like what line 32 shows, use a **query**() function.

```
24 # To answer the question of interest,
25 # select the portion of the data corresponding with the year 2000, and the total population.
26 # create a mask to select the needed data
27 mask = (final['year']==2010) & (final['ages']== 'total')
28 # save the masked data to a new variable data2010
29 data2010 = final[mask]
30
31 # Use the query() function to do this quickly
32 #data2010 = final.query("year == 2010 & ages == 'total'")
33
```

(8) We can then compute the density and add it as a new column to the DataFrame.

```
# compute the population, and assign it as the value of the new column 'density'
#  this line of code adds a new key-pair pair to the DataFream
data2010['density'] = data2010['population'] / data2010['area (sq. mi)']
```

(9) To sort the DataFrame by a particular column, we can use **sort_value()** and specify the column and whether in ascending order or not.

```
# sort the result in descending order 'density'
data2010=data2010.sort_values('density', ascending=False)
```

(10) The result is now available in **data2010**.

	state/region	ages	year	population	state	area (sq. mi)	density
389	DC	total	2010	605125.0	District of Columbia	68	8898.897059
1445	NJ	total	2010	8802707.0	New Jersey	8722	1009.253268
1914	RI	total	2010	1052669.0	Rhode Island	1545	681.339159
293	CT	total	2010	3579210.0	Connecticut	5544	645.600649
1050	MA	total	2010	6563263.0	Massachusetts	10555	621.815538

Exercise: How to print just the state and density.

In the US States Data program, how to print just the state and the density to the user?

3. Player Salary Data

(11) The program reads data from a csv files, representing data of NBA players' salaries in different years. This program can show the salary for a certain year, and allows the user to update the salary.

(12) At line 12, set the index of the DataFrame to the players' names.

```
6  # use the read_csv() funcito to read the data from three file into three DataFrame
7  # Note: The files should be at the same directory as the program
8  salary = pd.read_csv('player-salary.csv')
9  # salary
10 
11 # set index to player's name
12 salary = salary.set_index('PLAYER')
```

(13) If the player exists in the DataFrame and the year is entered correctly, show the value of the certain cell.

```
# if the name does not exist, show error
if playerName not in salary.index:
    print("Can't find the data for the player", playerName)
# if the name exists
else:
    # ask the user which year of data to see
    print("Which year of salary? 2022/23,2023/24,2024/25, or 2025/26?")
    year = input()
    # input validation
    while year not in('2022/23','2023/24','2024/25', '2025/26'):
        print('Please enter a valid year. 2022/23,2023/24,2024/25, or 2025/26? ')
        year = input()

    # show the value from the cell.
    print("The salary is $", salary.at[playerName,year])
```

(14) If the user enters a value to update the salary, assign the new value to the cell.

```
    # ask the user whether they would like to update the salary
    print('Please enter the udpated salary. Enter -1 if you do not wish to update the value.')
    newValue = int(input())
    # if the user did not enter -1
    if newValue != -1:
        # update the cell value
        salary.at[playerName,year] = newValue
```

Visualization with Matplotlib (1)

UNLV

Outline

- Introduction to Matplotlib
- Simple Line Plots
- Simple Scatter Plots
- Histograms, Binnings, and Density
- Customizing Plot Legends and Colorbars

UNLV

Introduction to Matplotlib (1)

- Data visualization library built on NumPy arrays
  - Allows visual access to huge amounts of data in easily digestible visuals
  - Large user base and an active developer base
  - Predated Pandas by more than a decade, and thus is not designed for use with Pandas DataFrames
- Seaborn
  - Provides an API on top of Matplotlib that offers choices for plot style and color defaults
  - Defines simple high-level functions for common statistical plot types
  - Integrates with the functionality provided by Pandas DataFrames

Introduction to Matplotlib (2)

- Importing Matplotlib

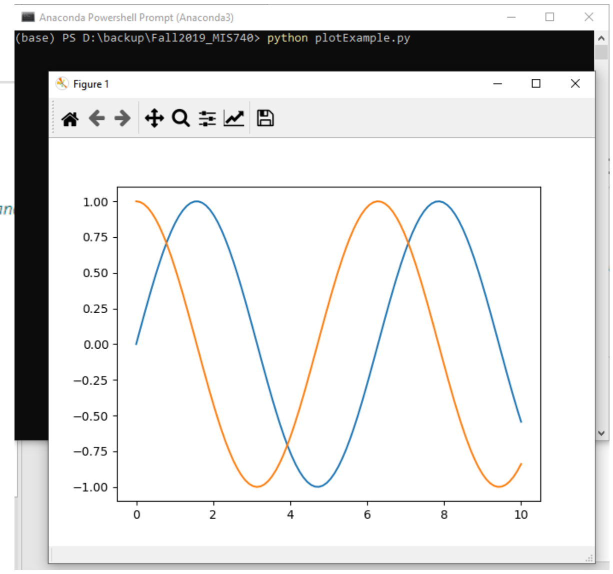
```
# import matplotlib, set the alias as mpl
import matplotlib as mpl
# import the pyplot module, set the alias as plt
import matplotlib.pyplot as plt
```

  - Pyplot is the most used module of Matplotlib
    - Provides an interface like MATLAB but instead, it uses Python and it is open source
- IPython is built to work well with Matplotlib

Introduction to Matplotlib (3)

- `plt.show()`
  - If we run the .py file from the shell, the `show()` function is needed to opens a window that display the figure

```python
# import numpy, set the alias as np
import numpy as np
# import the pyplot module, set the alias as plt
import matplotlib.pyplot as plt

# an array of 100 numbers evenly distributed between 0 an
x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.show()
```



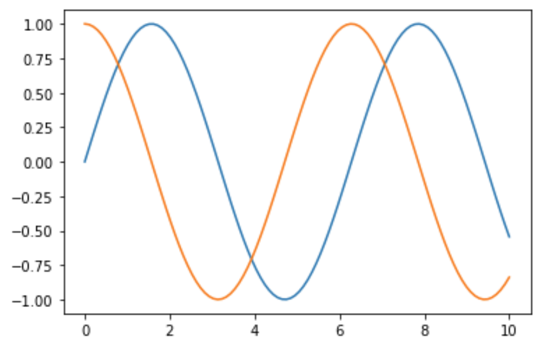
Introduction to Matplotlib (4)

- IPython is built to work well with Matplotlib if we specify Matplotlib mode
  - `%matplotlib inline` will lead to static images of the plot embedded in the notebook
  - Creating a plot will embed a PNG image of the resulting graphic
  - It needs to be done only once per kernel/session

Introduction to Matplotlib (5)

```
# import numpy, set the alias as np
import numpy as np
# import the pyplot module, set the alias as plt
import matplotlib.pyplot as plt

# an array of 100 numbers evenly distributed between 0 and 100
x = np.linspace(0, 10, 100)

%matplotlib inline
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
```

```
[<matplotlib.lines.Line2D at 0x1763436eac8>]
```



Simple Line Plots (1)

- For all Matplotlib plots, we start by creating a figure and an axes

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

# assign the figure object to a variable
fig = plt.figure()
# assign the axes object to a variable
ax = plt.axes()
```

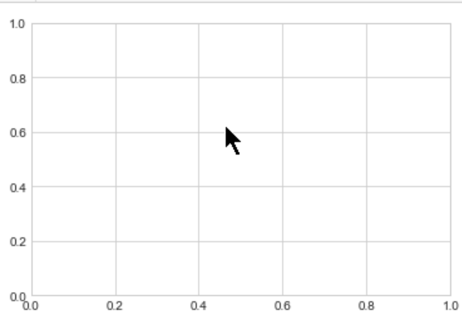


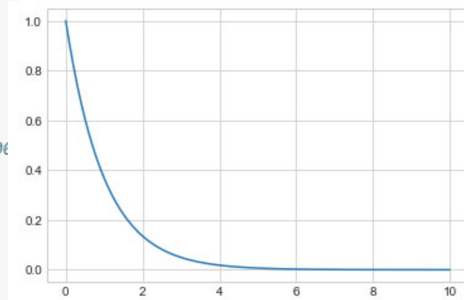
figure is a single container that contains all the objects representing axes, graphics, text, and labels.

axes is a bounding box with ticks and labels, which will eventually contain the plot elements that make up our visualization

Simple Line Plots (1)

- Use the `plot()` function to draw the plot

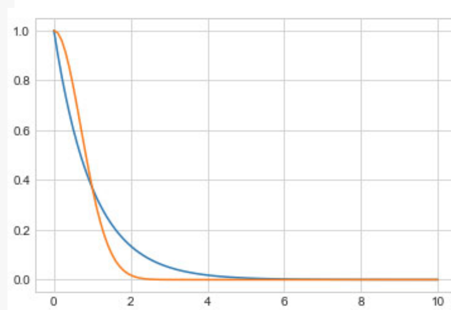
```python
# import numpy, set the alias as np
import numpy as np
# import the pyplot module, set the alias as plt
import matplotlib.pyplot as plt

%matplotlib inline

# an array of 100 numbers evenly distributed between 0 and 100
a = np.linspace(0, 10, 100)
# an array of exponential values of -a
b = np.exp(-a)

# use a as x axis and b as y axis for a line plot
plt.plot(a, b)
```



Simple Line Plots (3)

- To create a single figure with multiple lines, just simply call the plot function multiple times

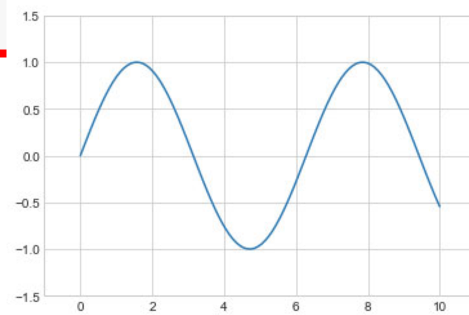
```python
# import numpy, set the alias as np
import numpy as np
# import the pyplot module, set the alias as plt
import matplotlib.pyplot as plt

# an array of 100 numbers evenly distributed between 0 and 100
a = np.linspace(0, 10, 100)
# an array of exponential values of -a
b = np.exp(-a)
c = np.exp(-a*a)
# use a as x axis and b as y axis for a line plot
plt.plot(a, b)
# call plot() again to add another line
plt.plot(a, c)
```



Adjusting the Plot: Axes Limits

- To adjust axis limits is to use the `plt.xlim()` and `plt.ylim()` methods

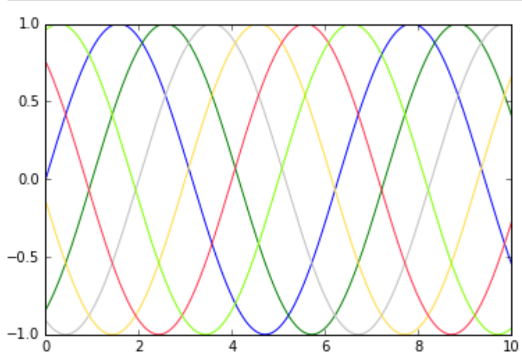
```
# import numpy, set the alias as np
import numpy as np
# import the pyplot module, set the alias as plt
import matplotlib.pyplot as plt

# an array of 100 numbers evenly distributed between 0 and 100
x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))

plt.xlim(-1, 11) # set the x axis to -1 and 11
plt.ylim(-1.5, 1.5) # set the y axis to -1.5 and 1.5
```



Line Colors and Styles (1)

- `color` keyword: accepts a string argument representing virtually any imaginable color
  – If no color is specified, Matplotlib will automatically cycle through a set of default colors for multiple lines

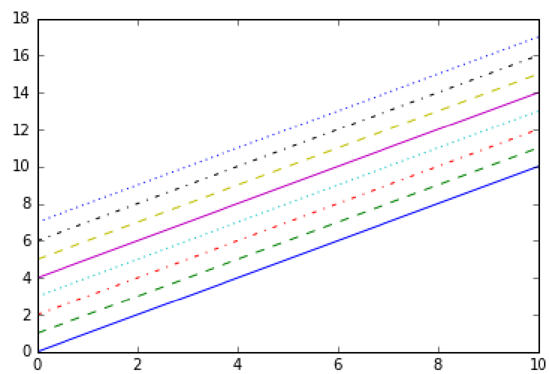
```
x = np.linspace(0, 10, 1000)
plt.plot(x, np.sin(x - 0), color='blue')        # specify color by name
plt.plot(x, np.sin(x - 1), color='g')           # short color code (rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75')        # Grayscale between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44')     # Hex code (RRGGBB from 00 to FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 to 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
```



Line Colors and Styles (2)

- `linestyle` keyword: Specify the line style

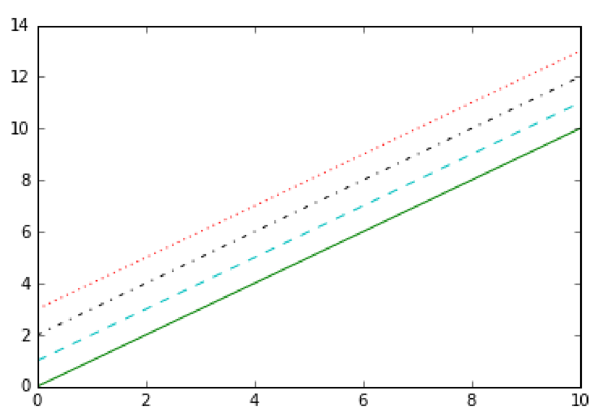
```
x = np.linspace(0, 10, 1000)
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');

# For short, you can use the following codes:
plt.plot(x, x + 4, linestyle='-')  # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':');  # dotted
```



Line Colors and Styles (3)

- `linestyle` and `color` codes can be combined into a single non-keyword argument

```
x = np.linspace(0, 10, 1000)
plt.plot(x, x + 0, '-g')  # solid green
plt.plot(x, x + 1, '--c') # dashed cyan
plt.plot(x, x + 2, '-.k') # dashdot black
plt.plot(x, x + 3, ':r');  # dotted red
```

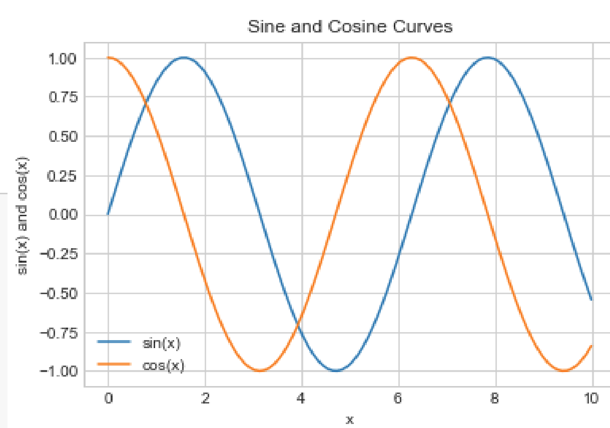


Labeling Plots

- **title()**, **xlabel()**, and **ylabel()**
  - Set the title and axis labels
- **legend()** and the keyword **label** in plot()
  - Set the plot legend

UNLV

Labeling Plots: Example



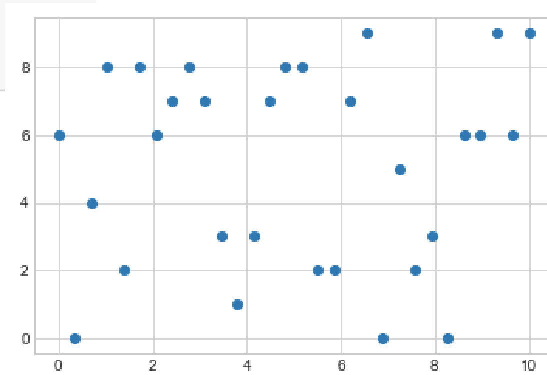

```
%matplotlib inline
# import numpy, set the alias as np
import numpy as np
# import the pyplot module, set the alias as
import matplotlib.pyplot as plt

# an array of 100 numbers evenly distributed
x = np.linspace(0, 10, 100)

# set the title of the plot
plt.title("Sine and Cosine Curves")
# set the label of x axis
plt.xlabel("x")
# set the label of y axis
plt.ylabel("sin(x) and cos(x)");

# set the plot legend for each line on the plot
plt.plot(x, np.sin(x), label='sin(x)')
plt.plot(x, np.cos(x), label='cos(x)')
plt.legend() # show the legend
```


UNLV

Simple Scatter Plots

- `scatter()` function

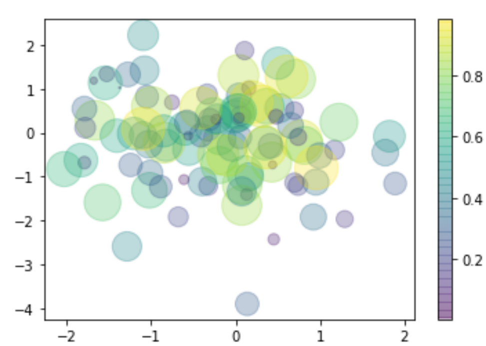
```
%matplotlib inline
# import numpy, set the alias as np
import numpy as np
# import the pyplot module, set the alias as plt
import matplotlib.pyplot as plt

# an array of 30 numbers evenly distributed between 0 and 100
x = np.linspace(0, 10, 30)
# an array of 30 random integers between 0 and 9
y = np.random.randint(0, 10, 30)

plt.scatter(x, y)

```



Scatter Plots with Color and Size

- c keyword: assign different colors to the dots
  - array or list of colors or color
- s keyword: assign different size to the dot

```
x = np.random.randn(100)
y = np.random.randn(100)
colors = np.random.rand(100)
sizes = 1000 * colors

# c is defined by the numbers in colors
# s is defined by the numbers in size
# cmap specifies the color map
plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
            cmap='viridis')
plt.colorbar()  # show color scale
```



Colormaps

- Matplotlib has a number of built-in colormaps
  - They are accessible via at `matplotlib.colormaps`

```
import matplotlib.pyplot as plt


plt.colormaps()
```

```
['Accent',
 'Accent_r',
 'Blues',
 'Blues_r',
 'BrBG',
 'BrBG_r',
 'BuGn',
 'BuGn_r',
 'BuPu',
 'BuPu_r',
 'CMRmap',
 'CMRmap_r',
 'Dark2',
 'Dark2_r',
 'GnBu',
```

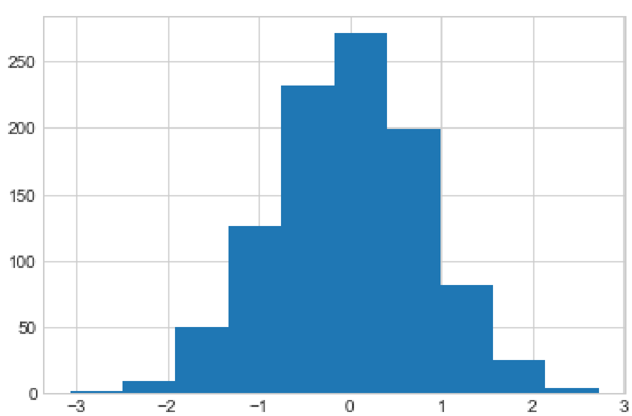

- Lab
  - Height and Weight by Age Group
    This program reads the data from a csv file and then plot the relationships between height and weight

Histograms

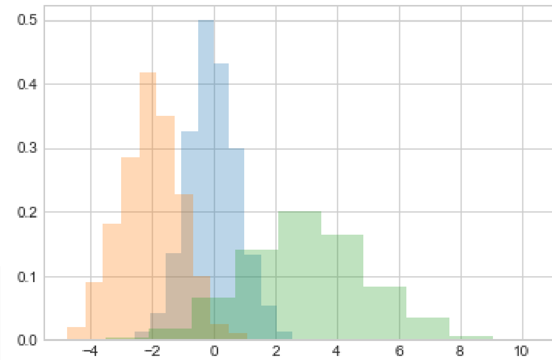
- A simple histogram can be a great first step in understanding a dataset
- `hist()` function
  - `bins` keyword: specify the number of bins
  - `histtype` keyword:
    - '`bar`' is a traditional bar-type histogram. If multiple data are given the bars are arranged side by side.
    - '`barstacked`' is a bar-type histogram where multiple data are stacked on top of each other.
    - '`step`' generates a lineplot that is by default unfilled.
    - '`stepfilled`' generates a lineplot that is by default filled.
  - `alpha` keyword: set the opacity
  - `density` keyword: The area (or integral) under the histogram will sum to 1

Histograms: Example (1)

```python
# import numpy, set the alias as np
import numpy as np
# import the pyplot module, set the alias as plt
import matplotlib.pyplot as plt
%matplotlib inline
# 1000 random numbers with mean=0, sd = 0.8
x1 = np.random.normal(0, 0.8, 1000)
plt.hist(x1, histtype='stepfilled', bins=10)
plt.show()
```



Histograms: Example (2)



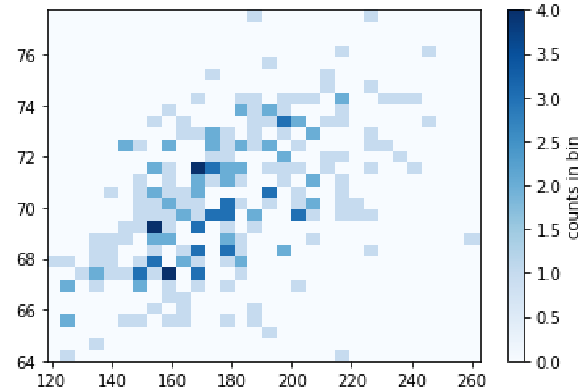

```python
# import numpy, set the alias as np
import numpy as np
# import the pyplot module, set the alias as
import matplotlib.pyplot as plt
%matplotlib inline

# three random number arrays, with 1000 numbres each
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)

# a shared set of keywords
kwargs = dict(histtype='stepfilled', alpha=0.3, density=True, bins=10)

# draw the three subsets of data, with the same set of keywords
plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs)
```


Two-Dimensional Histograms

- We can also create histograms in two-dimensions by dividing points among two-dimensional bins
  – Create a heat map of the data
- `hist2d()` function
  – `bins` keyword: specify the number of bins for the two dimensions
  – `cmap` keyword: color theme

Two-Dimensional Histograms: Example

```
# draw the 2-d scatter plot
# 30 bins on each dimension. Color map as blue
plt.hist2d(weight, height, bins=30, cmap='Blues')
# show the colorar on the side with labels
plt.colorbar().set_label('counts in bin')
```



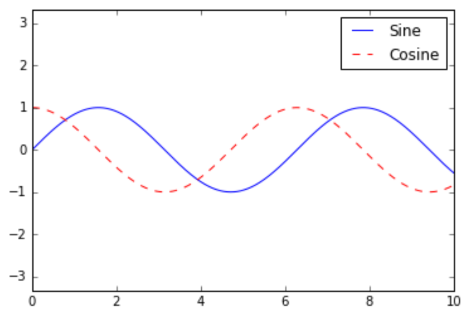
Customizing Plot Style

- `plt.style.use()`
  - Specify the style you would like to apply
  - You can use `plt.style.available` to see all the available styles
    - ['seaborn-ticks', 'ggplot', 'dark_background', 'bmh', 'seaborn-poster', 'seaborn-notebook', 'fast', 'seaborn', 'classic', 'Solarize_Light2', 'seaborn-dark', 'seaborn-pastel', 'seaborn-muted', '_classic_test', 'seaborn-paper', 'seaborn-colorblind', 'seaborn-bright', 'seaborn-talk', 'seaborn-dark-palette', 'tableau-colorblind10', 'seaborn-darkgrid', 'seaborn-whitegrid', 'fivethirtyeight', 'grayscale', 'seaborn-white', 'seaborn-deep']

Customizing Plot Style: Example

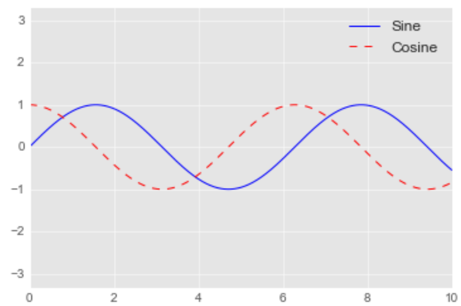
```python
plt.style.use('seaborn-muted')

x = np.linspace(0, 10, 1000)
# Get the fiure and axes from the plot
fig, ax = plt.subplots()
# draw the two lines with respective style and label
ax.plot(x, np.sin(x), '-b', label='Sine')
ax.plot(x, np.cos(x), '--r', label='Cosine')

# adjust plots with equal axis ratios
ax.axis('equal')
# show the legend
ax.legend()
```

```
<matplotlib.legend.Legend at 0x1763a39ba08>
```




```python
plt.style.use('ggplot')

x = np.linspace(0, 10, 1000)
# Get the fiure and axes from the plot
fig, ax = plt.subplots()
# draw the two lines with respective style and labels
ax.plot(x, np.sin(x), '-b', label='Sine')
ax.plot(x, np.cos(x), '--r', label='Cosine')

# adjust plots with equal axis ratios
ax.axis('equal')
# show the legend
ax.legend()
```

```
<matplotlib.legend.Legend at 0x276d1c53b48>
```



UNLV

Customizing Plot Legends

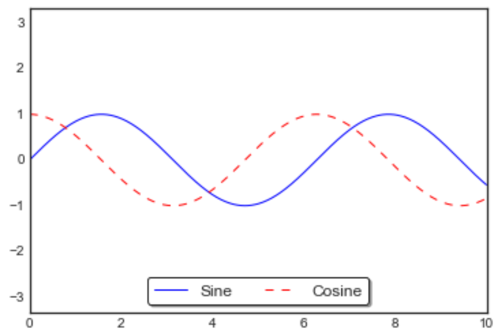
- `loc` keyword: specify the location
- `frameon` keyword: turn on or off the frame
- `ncol` keyword: specify the number of columns
- `fancybox` keyword: use a rounded box or not
- `shadow` keyword: add a shadow

UNLV

Customizing Plot Legends: Example

```python
plt.style.use('seaborn-white')

x = np.linspace(0, 10, 1000)
# Get the fiure and axes from the plot
fig, ax = plt.subplots()
# draw the two lines with respective style and labels
ax.plot(x, np.sin(x), '-b', label='Sine')
ax.plot(x, np.cos(x), '--r', label='Cosine')

# adjust plots with equal axis ratios
ax.axis('equal')
# show the legend to use fancybox, turn the frame on, add shadow
# make the location as lower center, two columns
ax.legend(fancybox=True, frameon=True, shadow=True, loc='lower center', ncol=2)
```

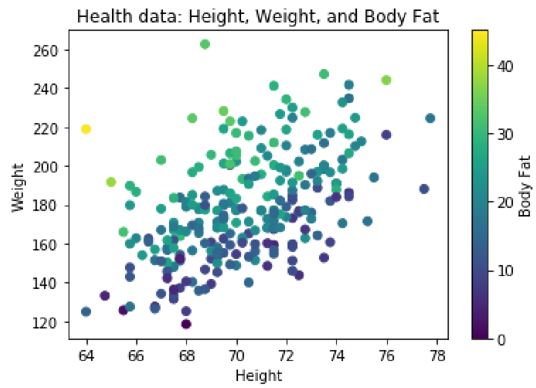
```
<matplotlib.legend.Legend at 0x276d2fd0f08>
```



- Lab
  - California Cities
    The program reads data from a csv file and plot the California cites. The size of the dots represents the area, and the color shows the population

Exercise

– Height Weight & BodyFat
  Please use the data bodyData.csv to visualize the height, weight, and percentage body fat data. For example, create a figure as shown below



MIS 740: Software Concepts
Fall 2022

Matplotlib (2)

Purpose

- Visualize data by Matplotlib with subplots
- Visualize data by using Seaborn

1. Preparation

(1) Please download **21_lab files.zip** from WebCampus. Unzip the files and import them to Jupyter Notebook.

2. Sales History Comparison

(2) The program reads a file containing the quarterly sales data and plot histogram for comparison.

The program starts with reading the data and define arrays to be used in labelling the subplot.

The data contains data in 2017 and 2018 for four quarters. You can preview the data.

```python
# import pandas, set the alias as pd
import pandas as pd
# import the pyplot module, set the alias as plt
import matplotlib.pyplot as plt

# define the array to be used in the title
year = ['2017','2018']
# define the array to be used in the title
quarter = ['Q1', 'Q2', 'Q3','Q4']

# read data from the csv file
sales = pd.read_csv('1718quarterly_sales.csv')
sales
```

	2017Q1	2017Q2	2017Q3	2017Q4	2018Q1	2018Q2	2018Q3	2018Q4
0	8704.0	9398.0	5145	6011	7915.0	9781.0	4062	10773
1	6751.0	8935.0	9875	8207	6529.0	9395.0	7515	6284
2	6884.0	6361.0	9011	6328	4117.0	6470.0	8337	7059
3	8113.0	6484.0	6114	6055	7779.0	8653.0	9610	8293
4	7504.0	8541.0	5947	7178	7090.0	8871.0	9308	8185
...	...	...	...	...	...	...	...	...
87	8443.0	7917.0	7695	7551	10813.0	5407.0	4903	8843
88	9609.0	9499.0	7890	8147	7933.0	10336.0	10102	8106
89	8164.0	6477.0	9783	5213	6923.0	6311.0	5143	6177
90	NaN	5194.0	8997	8410	NaN	7479.0	8665	4794
91	NaN	NaN	6034	9756	NaN	NaN	4811	10166

92 rows × 8 columns

(3) Note that the data contain some missing values, we need to fill them with 0.

```
# read data from the csv file
sales = pd.read_csv('1718quarterly_sales.csv')

# fill the missing data with 0
sales = sales.fillna(0)

```

(4) Then, create the subplots as 2 X 4.

```
# create subplots, number of rows as 2, number of columns as 4
fig, axs = plt.subplots(2,4)
```

A more scalable way is to use the length of the **year** and **quarter** array, in case we expand the scope of the dataset.

```
# create subplots, number of rows as 2, number of columns as 4
fig, axs = plt.subplots(len(year), len(quarter))
```

(5) We will use a variable to index the columns, so that we can plot the data by columns one after another.

```
# index the column of data to be plotted, starting from 0
index = 0
# for loop to loop through the years

    # for loop to loop through the quarters

        # set the title for each subplot. showing YXXXX QX
        # the axs represent the two dimensional array indexed by row and column

        # sales.iloc is used to indicate which column of data to plot

        # move to the next column
        index +=1
```

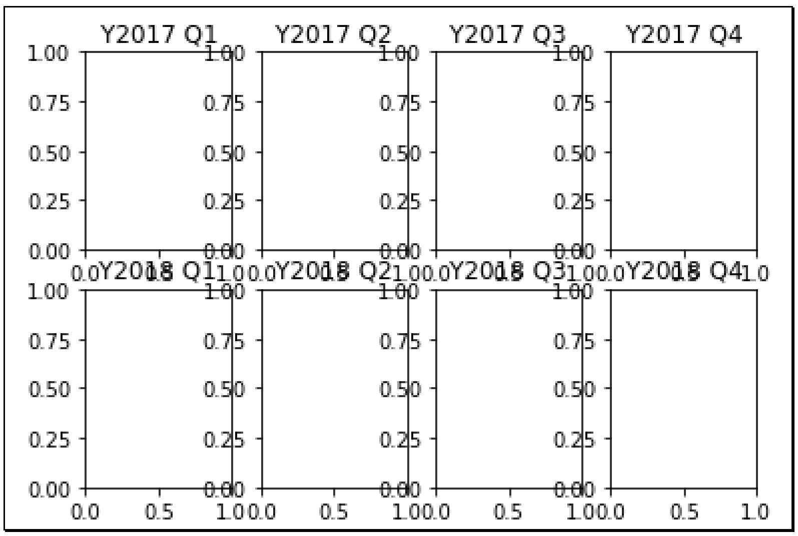
(6) A nested loop will be used to go through the years and the quarters.

```
# index the column of data to be plotted, starting from 0
index = 0
# for loop to loop through the years
for y in range(len(year)):
    # for loop to loop through the quarters
    for q in range(len(quarter)):
        # set the title for each subplot. showing YXXXX QX
        # the axs represent the two dimensional array indexed by row and column

        # sales.iloc is used to indicate which column of data to plot

        # move to the next column
        index +=1
```

(7) The axs variable is a two-dimensional array. We thus need to index it by the year and quarter. Add the title for each subplot accordingly.

```
25 # index the column of data to be plotted, starting from 0
26 index = 0
27 # for loop to loop through the years
28 for y in range(len(year)):
29     # for loop to loop through the quarters
30     for q in range(len(quarter)):
31         # set the title for each subplot. showing YXXXX QX
32         # the axs represent the two dimensional array indexed by row and column
33         axs[y, q].set_title('Y'+year[y]+" "+quarter[q])
34         # sales.iloc is used to indicate which column of data to plot
35 
36         # move to the next column
37         index +=1
```

(8) Now when you run the program, the titles are added. However, the figure is too small that cannot be read clearly.



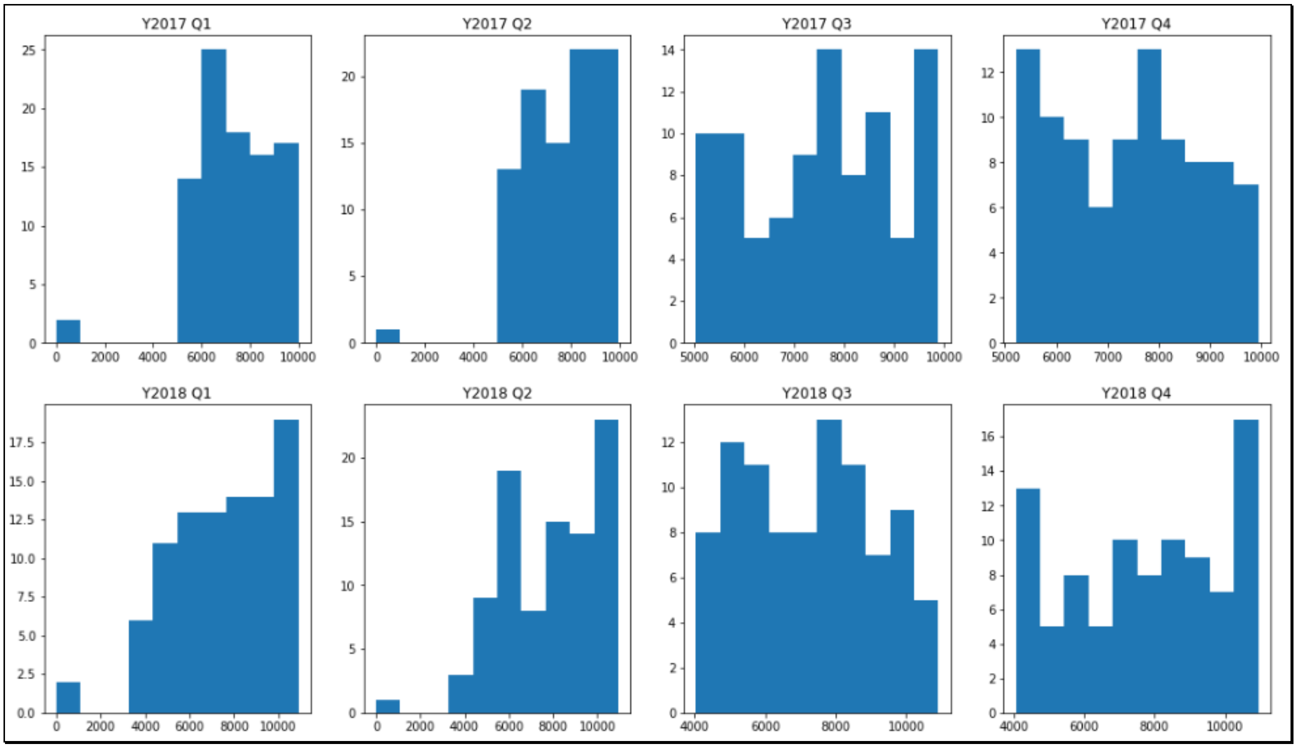
(9) Please move back up to lines 20-23. We can space out the figures and increase the size of the figure.

```
20 # add spaces to the subplots
21 fig.subplots_adjust(hspace=0.2, wspace=0.2)
22 # increate the size of the figure
23 fig.set_size_inches(18.5, 10.5)
```

(10) Finally, plot the histogram using **sales** data column by column with the **index**.

```
27 # for loop to loop through the years
28 for y in range(len(year)):
29     # for loop to loop through the quarters
30     for q in range(len(quarter)):
31         # set the title for each subplot. showing YXXXX QX
32         # the axs represent the two dimensional array indexed by row and column
33         axs[y, q].set_title('Y'+year[y]+" "+quarter[q])
34         # sales.iloc is used to indicate which column of data to plot
35         axs[y, q].hist(sales.iloc[:,index], histtype='bar', bins=10)
36         # move to the next column
37         index +=1
```

(11) When you run the program, you will see the data being plotted.



(12) However, the axes are not equivalent on the subplots. It makes the comparison difficult. We can set the axes to the same using **sharex** and **sharey** keywords. Please modify line 19.

```
17 # create subplots, number of rows as 2, number of columns as 4
18 # share teh ticks on x and y axis
19 fig, axs = plt.subplots(len(year), len(quarter), sharex='col', sharey='row')
```

(13) Run the program to see the difference.

3. Three-Dimensional Body Data

(14) This program reads the data from a csv file and then plot the data points.

To enable 3D plotting, please import **mplot3d.**

```
# import pandas, set the alias as pd
import pandas as pd
# import the pyplot module, set the alias as plt
import matplotlib.pyplot as plt
# to enable the 3d plotting
from mpl_toolkits import mplot3d

# show the plot wih inline mode
%matplotlib inline
```

(15) It is the same data file we used in the previous lab. We follow the same process to read the height, weight, and percentage of body fat. Then, create the 3D axes:

```
# Extract the data we're interested in
height = data['Height']
weight = data['Weight']
bodyFat = data['Percent body fat']

# create 3d axes
ax = plt.axes(projection='3d')

```

(16) Once the axes is created, we can call the **scatter3D**() function and pass x, y, and z data.

```
# Scatter the points
# body fat as x, height as y, weight as y
ax.scatter3D(bodyFat, height, weight)

```

(17) To set the labels for the axes, call the **set_xlabel**(), **set_ylebel**() and **set_zlabel**() function.

```
# set the x and y labels
ax.set_xlabel('Percentage body fat')
ax.set_ylabel('Height')
ax.set_zlabel('Weight')
```

(18) Run the program to see the result.

Exercise: Revise Three-Dimensional Body Data program

(19) Please create three data sets: people less than or equal 30 years old, people between 30 and 60, and people over 60. Plot the three datasets on the 3D scatter plot.

4. Tip Distribution

(20) The program is a test of many useful functions in Seaborn, including pair plot, faceted histograms, factor plots, and joint distribution.

First, we will be using the built-in data in the online repository to practice the different plots

You can see all the available data sets using **sns.get_dataset_names()**

```
# show the plot wih inline mode
%matplotlib inline

# show all the dataset available in the online repository
sns.get_dataset_names()

```

It shows several datasets that you can play around with.

```
['anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri',
 'gammas',
 'iris',
 'mpg',
 'planets',
 'tips',
 'titanic']
```

(21) We will use the dataset **tips**. Please load it and preview the data.

```
# show all the dataset available in the online repository
#sns.get_dataset_names()

# load the dataset "tips"
tips = sns.load_dataset('tips')
tips
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

(22) We can calculate and add a column to the dataset that can be used later in the plots.

```
# calculate and create a new column tip_pct
tips['tip_pct'] = 100 * tips['tip'] / tips['total_bill']
```

(23) First, create pair plots, with gender as the segregator

```
# 1. show the pairplot for all the variables
sns.pairplot(tips, hue='sex', height=2.5)
```

The **hue** keyword here would accept a categorical variable.

(24) Second, the facet grid is used to show the histograms of subsets. We want to categorize the data by sex and time (i.e., lunch or dinner time). We also specify the bins.

```
# 2. create a FacetGrid that using the tips dataset
#  in the plot, with row as sex and col as dinner or lunch time
grid = sns.FacetGrid(tips, row="sex", col="time", margin_titles=True)
# show the paired histgram, using tip_pct,
# the bins are 15 numbers between 0 and 40
grid.map(plt.hist, "tip_pct", bins=np.linspace(0, 40, 15))
```

(25) The factor plot can be used to compare the distribution along a variable.

```
# 3. Factor plots
# for each day, compare the total bill amount by sex.
# Use box plot to show the distribution
g = sns.catplot("day", "total_bill", "sex", data=tips, kind="box")
# set the x and y labels
g.set_axis_labels("Day", "Total Bill")
```

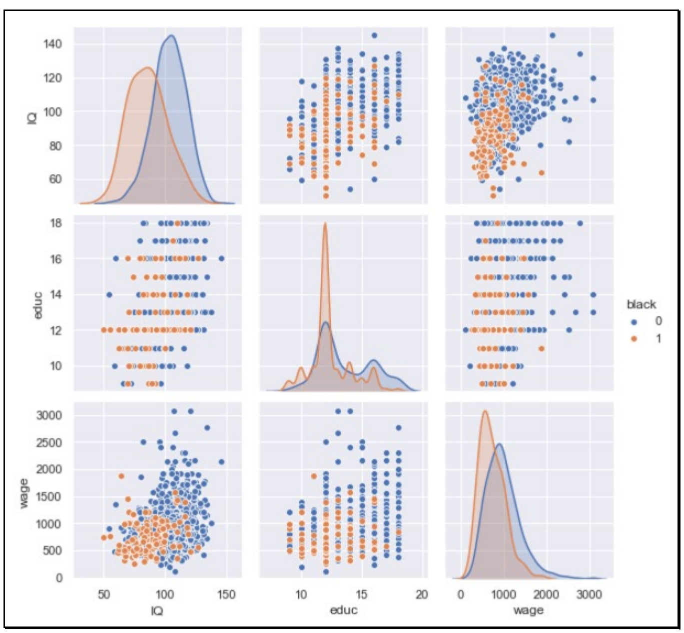
(26) Finally, the joint plot is used to show the correlation between two variables.

```
# 4. Joint Plot
# show the correlation of the totoal bill amount and the tip
sns.jointplot("total_bill", "tip", data=tips, kind='reg')
```

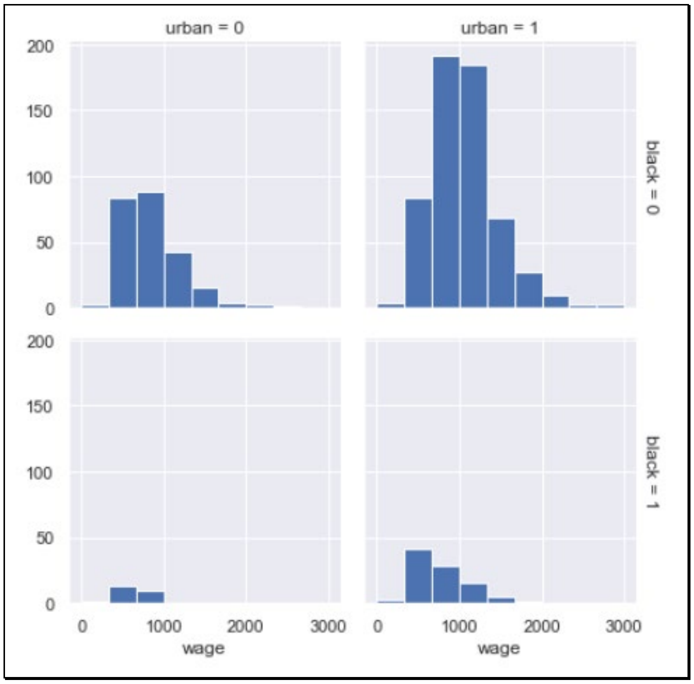
Exercise: Wage in 1980

Write a program that read the data “wage.csv” and create the following diagrams:

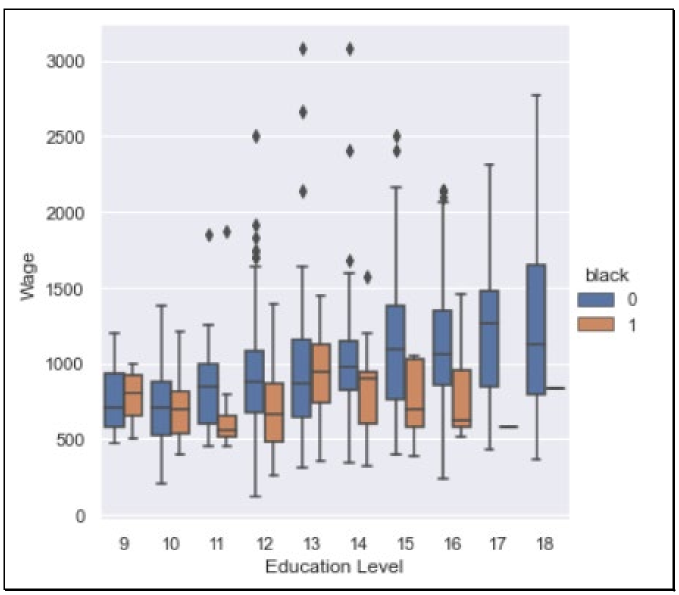
- pair plot for four variables: IQ, education, black, and wage.



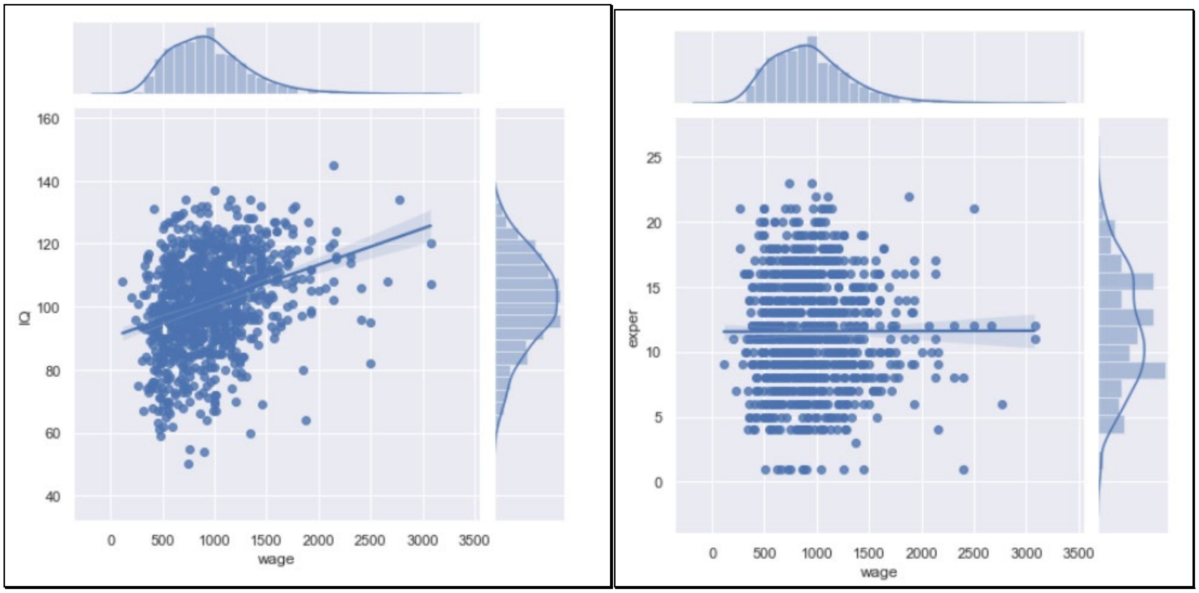
- faceted histograms for wage, categorized by black and urban (10 bins between 0 and 3000)



- factor plot: for each education level, compare the wage by black or not.



- joint distribution on (1) IQ and wage and (2) experience and wage



Data source: Introductory Econometrics: A Modern Approach, 6e by Jeffrey M. Wooldridge.