Name: Date:

Overview

	Lesson Plan
1	- Recap on what is a list, dictionary - Recap on methods & functions - Recap on Random Library
2	- Revisiting Pandas - Understanding Pandas DateTime Objects - Revisiting Matplotlib
3	- Handling Online Data with Pandas - Bar Graph, Line Graph - Multiplot
4	- Data Analysis: Retrenched employees by industry and occupational Group

Name: Date:

Day 1

	Lesson Plan
1	- Recap on what is a list, dictionary - Recap on functions - Recap on Random Library

Name: Date:

Lesson 1.1

1.1 Learning – Recap on what is a list, dictionary

List

List Methods	Description
append()	Adds an element at the end of the list
remove()	Removes the first item with the specific value
pop()	Removes the element at the specific position
count()	Returns the number of elements with the specific value
sort()	Sorts the list

To recall on the list methods, we will follow the guided tutorial in writing.
Using List Methods, we will

- Remove `South Korea`
- Append `Brunei, Malaysia & Vietnam`
- Count the number of countries in the `countries`
- Sort the list in descending order and display

Step 1:

Create a list containing *Singapore, Thailand & South Korea* in a variable called *countries*

```
countries = ['Singapore', 'Thailand', 'South Korea']
```

Step 2:

Remove the *South Korea* from the list

```
countries.remove("South Korea")
```

.pop() is used to pop by index not the exact element

Step 3:

Add in *Brunei, Malaysia & Vietnam* to the list

```
countries.append("Brunei")
countries.append("Malaysia")
countries.append("Vietnam")
```

Name: Date:

Lesson 1.1

Step 4:

Count the number of countries in the list

```
print(len(countries))
```

To get the length of the list, we will use a built-in function instead of a list method

Step 5:

Sort the list in descending order and display the countries as a list

```
countries.sort(reverse=True)
print(countries)
```

A list is normally sorted in ascending order and not descending order. To sort by descending order, we will need to add an argument stating that reverse is True.

Full Code:

```
# Step 1: Create a list with items
countries = ['Singapore', 'Thailand', 'South Korea']

# Step 2: Remove South Korea
countries.remove("South Korea")

# Step 3: Append Brunei, Malayisa & Vietnam
countries.append("Brunei")
countries.append("Malaysia")
countries.append("Vietnam")
# Step 4: Count
print(len(countries))

# Step 5: Sort & display
countries.sort(reverse=True)
print(countries)
```

Name: Date:

Lesson 1.1

Dictionary

Dictionary Methods	Description
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing dictionary's keys
values()	Returns a list containing dictionary's values
pop()	Removes the element with the specific key
copy()	Returns a copy of dictionary
update()	Updates the dictionary with the specific key-value pairs

To recall on the dictionary methods, we will follow the guided tasks in writing. Please complete the output

Task 1: Getting Items of dictionary

```
player = {
    "id": "001",
    "alias": "the mastermind"
}
print(list(player.items()))
```

Output

list() is used to typecase the player.items() into a list

Task 2: Getting Keys of dictionary

```
player = {
    "id": "001",
    "alias": "the mastermind"
}
print(list(player.keys()))
```

Output

Name: Date:

Lesson 1.1

Task 3: Getting Values of dictionary

```
player = {
    "id": "001",
    "alias": "the mastermind"
}
print(list(player.values()))
```

Output

Task 4: Update dictionary

```
player = {
    "id": "001",
    "alias": "the mastermind"
}
player.update({"status": "alive"})
print(player)
```

Output

Name: Date:

Lesson 1.1

Task 5: Remove item from dictionary

```
player = {
    "id": "001",
    "alias": "the mastermind",
    "state": "ready"
}
player.pop("state") # Pop by key
print(player)
```

Output

Task 6: Modify value in dictionary

```
player = {
    "id": "001",
    "alias": "the mastermind",
    "status": "alive"
}
player["status"] = "dead" # By key
print(player)
```

Output

Other ways of modifying the value in dictionary is `player.update({"status":"dead"})`

Task 7: Modify value in dictionary

```
player = {
    "id": "001",
    "alias": "the mastermind",
    "status": "alive"
}
player.update({"status":"dead"})
print(player)
```

Output

Name: Date:

Lesson 1.1

1.1 Practice – Recap on what is a list, dictionary

Exercise 1

Write a Python code to create a dictionary from a string.

Input
'thelogiccoders'
Output
{'t': 1, 'h': 1, 'e': 2, 'l': 1, 'o': 2, 'g': 1, 'i': 1, 'c': 2, 'd': 1, 'r': 1, 's': 1}

Exercise 2

Write a Python code to print all unique values in a dictionary.

Input
[{"V":"S001"}, {"V": "S002"}, {"VI": "S001"}, {"VI": "S005"}, {"VII":"S005"}, {"V":"S009"},{"VIII":"S007"}]
Output
['S001', 'S002', 'S005', 'S009', 'S007']

Exercise 3

Write a Python code to create a dictionary from a nested lists.

Input
[['yellow', 1], ['blue', 2], ['yellow', 3], ['blue', 4], ['red', 1]]
Output
{'yellow': [1, 3], 'blue': [2, 4], 'red': [1]}

Guided solution found in the next few pages

Name: Date:

Lesson 1.1

Exercise 1

Write a Python code to create a dictionary from a string.

Input
'thelogiccoders'
Output
{'t': 1, 'h': 1, 'e': 2, 'l': 1, 'o': 2, 'g': 1, 'i': 1, 'c': 2, 'd': 1, 'r': 1, 's': 1}

Step 1: Understanding the problem

- Creating a dictionary from a string means counting each character in the given string
- 'hello' - for instance, have 1x h , 1x e, 2x l, 1x o
- We can loop through the string

Step 2: Creating the input

```
string = 'thelogiccoders'
```

Step 3: Creating the empty dictionary

```
d = {} # Empty Dictionary
```

Step 4: Looping through the string

```
for i in string:
```

Step 5: Updating the key within the loop

```
for i in string:
    if i in d.keys():
        d[i] += 1
    else:
        d[i] = 1
```

Step 6: Displaying the dictionary

```
print(d)
```

Name: Date:

Lesson 1.1

Exercise 2

Write a Python code to print all unique values in a dictionary.

Input
[{"V":"S001"}, {"V": "S002"}, {"VI": "S001"}, {"VI": "S005"}, {"VII":"S005"}, {"V":"S009"},{"VIII":"S007"}]
Output
['S001', 'S002', 'S005', 'S009', 'S007']

Step 1: Understanding the problem

- Unique means no duplicates
- Looking at value in {"V":"S001"}, we are looking at "S001"
- We can loop through the list of dictionaries

Step 2: Creating the input

```
data = [{"V":"S001"}, {"V": "S002"}, {"VI": "S001"}, {"VI": "S005"},
{"VII":"S005"}, {"V":"S009"},{"VIII":"S007"}]
```

Step 3: Creating the empty list

```
l = [] # Empty list
```

Step 4: Looping through the string

```
for d in data:
```

Step 5: Updating the list for new value that does not already exist in list

```
for d in data:
    for key, value in d.items():
        if value not in l:
            l.append(value)
```

Step 6: Displaying the dictionary

```
print(l)
```

Name: Date:

Lesson 1.1

Exercise 3

Write a Python code to create a dictionary from a nested lists.

Input
[['yellow', 1], ['blue', 2], ['yellow', 3], ['blue', 4], ['red', 1]]
Output
{'yellow': [1, 3], 'blue': [2, 4], 'red': [1]}

Step 1: Understanding the problem

- Within each inner list, it is always having the key, followed by a value
- We want the **sum** of all the value of each key

Step 2: Creating the input

```
data =  [['yellow', 1], ['blue', 2], ['yellow', 3], ['blue', 4], ['red', 1]]
```

Step 3: Creating the empty dictionary

```
d = [] # Empty dictionary
```

Step 4: Looping through the string to get key & value

```
for key, value in data:
```

Step 5: Updating the key within the loop

```
for key, value in data:
    if key in d.keys():
        d[key].append(value)
    else:
        d[key] = [value]
```

Step 6: Displaying the dictionary

```
print(d)
```

Name: Date:

Lesson 1.1

Solution 1

```
string = 'thelogiccoders'
d = {} # Empty Dictionary

for i in string:
    if i in d.keys():
        d[i] += 1
    else:
        d[i] = 1
print(d)
```

Solution 2

```
data = [{"V":"S001"}, {"V": "S002"}, {"VI": "S001"}, {"VI": "S005"},
{"VII":"S005"}, {"V":"S009"},{"VIII":"S007"}]
l = [] # Empty list

for d in data:
    for key, value in d.items():
        if value not in l:
            l.append(value)
print(l)
```

Solution 3

```
data =  [['yellow', 1], ['blue', 2], ['yellow', 3], ['blue', 4], ['red', 1]]
d = {}

for key, value in data:
    if key in d.keys():
        d[key].append(value)
    else:
        d[key] = [value]
print(d)
```

Name: Date:

Lesson 1.2

1.2 Learning – Recap on function

What is a function?

- A function is a block of organised, reusable code that is used to perform a single, related action
- Parameters are used to pass data into a function
- Functions are classified as either built-in functions or user-defined functions

Examples of common built-in functions

Common Built-in Function	Description
max()	Adds an element at the end of the list
min()	Removes the first item with the specific value
sum()	Removes the element at the specific position
len()	Returns the number of elements with the specific value

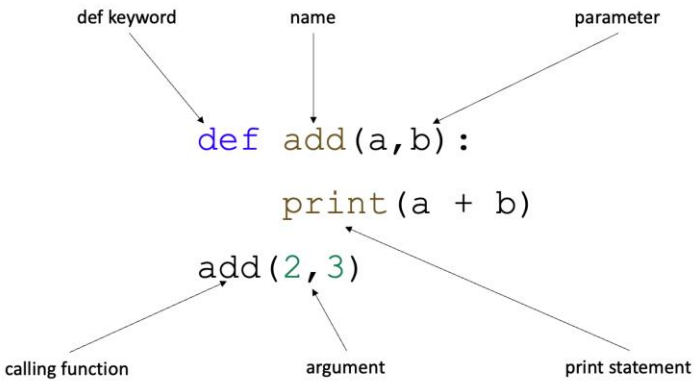
Rules for creating user-defined function

- Function blocks begin with the keyword `def` followed by the function name and parentheses ( ) . The keyword `def` marks the start of the function header.
- Function name is used to uniquely identify it, should not be used in conjunction with variable names
- The code block within every function starts with a colon (:) and is indented
- From the calling side, data is passed to the function as argument
- From the function side, data is passed to the function as parameter

Name: Date:

Lesson 1.2

Anatomy of a function



Example: Understanding the anatomy of a function

```
def add(a,b):
    print(a + b)
add(2,3)
```

Question	**Answer**
What is the function name?	*add*
What is the function argument?	2,3
What is the function parameters?	a,b
What is the output?	5
*Are we calling the function or defining it when we use **def**?*	defining

Name: Date:

Lesson 1.2

Task 1: Understanding the anatomy of a function

```
def div(a,b):
    print(a / b)
add(10,2)
```

Question	Answer
What is the function name?	
What is the function argument?	
What is the function parameters?	
What is the output?	
*Are we calling the function or defining it when we use **def**?*	

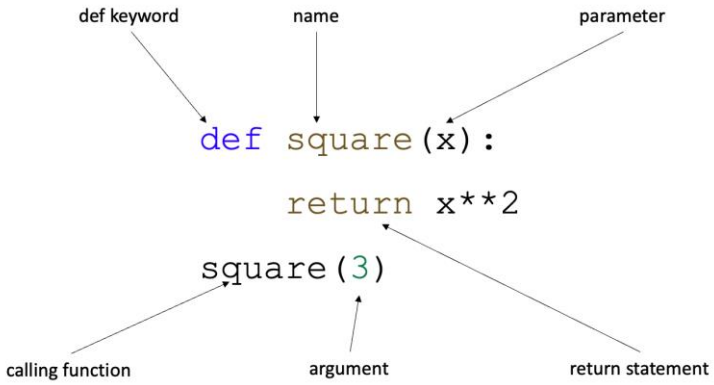
Task 2: Understanding the anatomy of a function

```
def mul(x,y):
    print(x * y)
mul(2,5)
```

Question	Answer
What is the function name?	
What is the function argument?	
What is the function parameters?	
What is the output?	
*Are we calling the function or defining it when we use **def**?*	

Name: Date:

Lesson 1.2



Example: Understanding the anatomy of a function with return

```
def square(x):
    return x**2
square(3)
```

Question	Answer
What is the function name?	*square*
What is the function argument?	3
What is the function parameters?	x
What is the output?	-
*Are we calling the function or defining it when we use **def**?*	defining

Name: Date:

Lesson 1.2

Complete the following tasks to get the output. Do look at the remarks.

Task 1: Returning only within the function

Code	Output	Remarks

`def square(x):`
`  return x**2`
`square(3)`
		There is no print statement. Square(3) will contain a value

Task 2: Printing only within the function

Code	Output	Remarks

`def square(x):`
`  print(x**2)`
`square(3)`
		There is only 1 print statement. Square(3) will contain None

Task 3: Returning with a printing outside the function

Code	Output	Remarks

`def square(x):`
`  return x**2`
`print(square(3))`
		There is only 1 print statement. Square(3) will contain a value

Task 4: Printing with a printing outside the function

Code	Output	Remarks

`def square(x):`
`  print(x**2)`
`print(square(3))`
		A function will always return None unless specified

Name: Date:

Lesson 1.2

1.2 Practice – Recap on function

Exercise 1

Write a Python code to emulate a to-do list in a command line interface (CLI).

The user requirements are as follows:

1. As a user, I want to add in new items to my to-do list
2. As a user, I want to sort my items in ascending order
3. As a user, I want to count my items in my to-do list
4. As a user, I want to remove old items from my to-do list

Exercise 2

Add a search feature to your Python code.

The additional user requirement is as follows:

1. As a busy user, I want to quickly search through my items.

Guided solution found in the next few pages

Name: Date:

Lesson 1.2

Exercise 1

Write a Python code to emulate a to-do list in a command line interface (CLI).

The user requirements are as follows:

1. As a user, I want to **add** in new items to my to-do list
2. As a user, I want to **sort** my items in ascending order
3. As a user, I want to **count** my items in my to-do list
4. As a user, I want to **remove** old items from my to-do list

Step 1: Understanding the problem

- Based on the user requirements, I need to be able to **add, sort, count & remove** from an existing data structure
- A Python List is an ideal data structure because there are available List methods or Built-in function to achieve this
- Even though the user requirements did not specify the end condition, it is always great to add in a termination case. In this case, we will use a 'quit' to terminate the program. Otherwise continuously allow user to edit the to-do list
- We will also want to display our list on command

Step 2: Creating the to-do list

```
l = []
```

Step 3: Creating the loop with termination case

```
while True:
    c = input() # input()
    if c == 'quit':
        break
```

While True is used so that the user can make infinite number of requests. However, when the user wants to end, he/she needs to only enter 'quit'.

Step 4: Adding new items

```
    elif c == 'add':
        d = input("Enter item:\t")
        print(f'{d} has been added to list')
        l.append(d)
```

Appending to the list is the way to add new items into the to-do list

Name: Date:

Lesson 1.2

Step 5: Displaying existing items

```
    elif c == 'display':
        print(l)
```

Step 6: Sorting existing items

```
    elif c == 'sort':
        l.sort()
        print(f"The list is sorted")
```

Step 7: Counting existing items

```
    elif c == 'count':
        print(f"There are {len(l)} items in the list")
```

A built-in function is used.

Step 8: Removing existing items

```
    elif c == 'remove':
        d = input("Enter item:\t")
        if d in l:
            print(f'{d} has been removed')
            l.remove(d)
        else:
            print(f'{d} not found')
```

Know that we can only remove existing items within the list, otherwise we should reject that request

Step 9: Rejecting invalid

```
    else:
        print("invalid")
```

What if the user wrote a command that is not one of the cases? Then by default, we should reject that request and inform the user

Name: Date:

Lesson 1.2

Exercise 2

Add a search feature to your Python code.

The additional user requirement is as follows:

1. As a busy user, I want to quickly search through my items.

Step 1: Understanding the problem

- This is a sub-string searching problem
- Python has a very easy way of implementing this

Step 2: Trying Substring searching in Python

```
s = 'hello'

print('h' in s)
print('a' in s)
```

*Given a string **s = 'hello'**, we will be able to return True or False by checking if a substring exists within a string. For instance, **'h' in s** checks if there is a letter 'h' in s*

Step 3: Adding additional feature

```
    elif c == 'search':
        d = input("Enter item:\t")
        if d in l:
            print(f'{d} has been found')
        else:
            print(f'{d} not found')
```

Name: Date:

Lesson 1.2

Solution

```
l = []

while True:
    c = '0' # input()
    if c == '0':
        break
    elif c == 'display':
        print(l)
    elif c == 'add':
        d = input("Enter item:\t")
        print(f'{d} has been added to list')
        l.append(d)
    elif c == 'remove':
        d = input("Enter item:\t")
        if d in l:
            print(f'{d} has been removed')
            l.remove(d)
        else:
            print(f'{d} not found')
    elif c == 'count':
        print(f"There are {len(l)} items in the list")
    elif c == 'sort':
        l.sort()
        print(f"The list is sorted")
    elif c == 'search':
        d = input("Enter item:\t")
        if d in l:
            print(f'{d} has been found')
        else:
            print(f'{d} not found')
    else:
        print("invalid")
```

Name: Date:

Lesson 1.3

1.3 Learning – Recap on Random Library

What is a module?

Consider a module to be the same as a code library. A file containing a set of functions you want to include in your application. Any text file with the .py extension containing Python Code is basically a module.

What is a library?

To begin, let us first talk about libraries and what they are. A library is a collection of modules that contains function for the use by other programs. They may contain some data values in them, and are often used to make a program's life easier

Python has built-in modules - Primarily, OS module, Sys module, Math module, Statistics module, Collections module & Random Module.

Objective

In this section, we will focus on the random library. The random module is a built-in module to generate the pseudo-random variables. It can also be used to perform some action randomly such as to get a random number, selecting a random elements from a list, shuffle elements randomly.

Name: Date:

Lesson 1.3

Task 1: Learning about random library

```
# To find out more
help(random.random)
```

Fill up the description after using help() to look up each methods

Method	Description
`random.randint`	
`random.randrange`	
`random.choice`	
`random.shuffle`	

Name: Date:

Lesson 1.3

In the next few tasks, use the methods you've learn to write Python codes

Task 2:
Write a Python Code to generate a random integer between 1 to 100.

```
Insert Code here
```

Task 3:
Write a Python Code to generate a random float between 0 to 1.

```
Insert Code here
```

Task 4:
Write a Python Code to generate a random alphabet.

```
import string
lst = string.ascii_letters
print(lst)

Insert Code here
```

lst will contains the alphabets, how do we then pick a random alphabet?

Name: Date:

Lesson 1.3

1.3 Practice – Recap on Random library

Exercise 1

Write a Python code to emulate a student database entry system in a command line interface (CLI) program.

The user requirements are as follows:

1. As a user, I want to add new students into the database
2. As a user, I want to know every student's name & gender (F/M)
3. As a user, I want to **randomly** assign them a house colour (Red, Green, Blue, Yellow)

Exercise 2

Add an additional filter feature to your Python code.

The additional user requirement is that:

1. As a busy user, I want to filter out the students by their house colour (Red, Green, Blue Yellow)

Guided solution found in the next few pages

Name: Date:

Lesson 1.3

Exercise 1

Write a Python code to emulate a student database entry system in a command line interface (CLI) program.

The user requirements are as follows:

1. As a user, I want to add new students into the database
2. As a user, I want to know every student's name & gender (F/M)
3. As a user, I want to **randomly** assign them a house colour (Red, Green, Blue, Yellow)

Step 1: Understanding the problem

- Based on the user requirements, I need to be able to **add** new students existing data structure
- A Python List is an ideal data structure because there are available List methods or Built-in function to store a collection of students
- A Python Dictionary can be used to store the student's details – name, gender, house colour
- Random library is necessary to randomly assign a colour
- Even though the user requirements did not specify the end condition, it is always great to add in a termination case. In this case, we will use a 'quit' to terminate the program.
- We will also want to display our list on command

Step 2: Import random library

```
import random
```

Step 3: Create empty list, house colours

```
l = []
house_colours = ['Red', 'Green', 'Blue', 'Yellow']
```

Step 3: Creating the loop with termination case

```
while True:
    c = input()
    if c == 'quit':
        break
```

While True is used so that the user can make infinite number of requests. However, when the user wants to end, he/she needs to only enter 'quit'.

Name: Date:

Lesson 1.3

Step 4: Adding new students

```
    elif c == 'add':
        name = input("Enter name:\t")
        gender = input("Enter gender(F/M):\t")

        d = {
            'name': name,
            'gender': gender,
            'house colour': random.choice(house_colours)
        }
        l.append(d)
        print(f'{d["name"]} has been added to list')
```

Appending to the list is the way to add new students in the list
We will need to request for the name & gender of the student
For the house colour, we will randomly select from the list – house_colours

Step 5: Displaying existing students

```
    elif c == 'display':
        print(l)
```

Step 6: Rejecting invalid

```
    else:
        print("invalid")
```

What if the user wrote a command that is not one of the cases? Then by default, we should reject that request and inform the user

Name: Date:

Lesson 1.3

Exercise 2

Add an additional filter feature to your Python code.

The additional user requirement is that:

1. As a busy user, I want to filter out the students by their house colour (Red, Green, Blue Yellow)

Step 1: Check if the input is a valid colour

```
        colour = input("Enter colour:\t")
        if colour in house_colours:
```

Step 2: Loop through the list and check if any student has the requested house colour

```
            for d in l:
                if d['house colour'] == colour:
                    print(d['name'], d['gender'])
```

Combining the 2 steps :

```
    elif c == 'filter':
        colour = input("Enter colour:\t")
        if colour in house_colours:
            for d in l:
                if d['house colour'] == colour:
                    print(d['name'], d['gender'])
```

Name: Date:

Lesson 1.3

Solution

```
import random

l = []
house_colours = ['Red', 'Green', 'Blue', 'Yellow']

while True:
    c = input()
    if c == 'quit':
        break
    elif c == 'display':
        print(l)
    elif c == 'add':
        name = input("Enter name:\t")
        gender = input("Enter gender(F/M):\t")

        d = {
            'name': name,
            'gender': gender,
            'house colour': random.choice(house_colours)
        }
        l.append(d)
        print(f'{d["name"]} has been added to list')
    elif c == 'filter':
        colour = input("Enter colour:\t")
        if colour in house_colours:
          for d in l:
            if d['house colour'] == colour:
              print(d['name'], d['gender'])
    else:
        print("invalid")
```

Name: Date:

Day 2

	Lesson Plan
2	- Revisiting Pandas - Understanding Pandas DateTime Objects - Revisiting Matplotlib

Name: Date:

Lesson 2.1

2.1 Learning – Revisiting Pandas

When it comes to creating DataFrame with Pandas in Python, we can always create using a list of dictionaries or a dictionary containing lists. This lesson is a refresher for basics in Pandas.

Task 1: List of Dictionary Entries

Code

```
import pandas as pd

d = [
    {"name": "Apple", "quantity": 3},
    {"name": "Pear", "quantity": 2},
    {"name": "Cucumber", "quantity": 5},
]

df = pd.DataFrame(d)
df
```

Output

Task 2: Understanding a DataFrame

Code

```
dict2 = [
    {"name": "Apple", "quantity": 3},
    {"name": "Pear", "quantity": 2},
    {"name": "Cucumber", "quantity": 5},
]
df = pd.DataFrame(dict2)

print(f"""
    headers:\n{df.columns}\n
    index:\n{df.index}\n
    first row:\n{df.iloc[0]}\n
    columns `Name`:\n{df['name']} \n
    """)
```

Output

Name: Date:

Lesson 2.1

Task 3: Dictionary containing lists

Code

```
import pandas as pd

d = {
    "name": ["Apple", "Pear", "Cucumber"],
    "quantity": [3, 2, 5]
}

df = pd.DataFrame(d)
df
```

Output

Task 4: Grouping Data by Specific Column

Code

```
# Grouping Data
d = [
    {"name": "Apple", "quantity": 3, 'type': 'Food'},
    {"name": "Pear", "quantity": 2, 'type': 'Food'},
    {"name": "Grape", "quantity": 5, 'type': 'Food'},
    {"name": "Kettle", "quantity": 1, 'type':
'Appliance'},
    {"name": "Pan", "quantity": 2, 'type':
'Appliance'},
    {"name": "Fan", "quantity": 3, 'type':
'Appliance'},
]

df = pd.DataFrame(d)
df.groupby(by=['type']).sum()
```

Output

When you df.groupby(by=['type']).sum(), type will become the index of the DataFrame.

Name: Date:

Lesson 2.1

Task 5: Resetting the index

Code	Output

`d = [`
`{"name": "Apple", "quantity": 3, 'type': 'Food'},`
`{"name": "Pear", "quantity": 2, 'type': 'Food'},`
`{"name": "Cucumber", "quantity": 5, 'type': 'Food'},`
`{"name": "Kettle", "quantity": 1, 'type': 'Appliance'},`
`{"name": "Pan", "quantity": 2, 'type': 'Appliance'},`
`{"name": "Fan", "quantity": 3, 'type': 'Appliance'},`
`]`
 	
`df = pd.DataFrame(d)`
`df = df.groupby(by=['type']).sum()`
`df.reset_index()`	

When you df.groupby(by=['type']).sum(), 'type' will become the index of the DataFrame. However, you do not want to have 'type' as the index. You can reset the index back to 0-indexing by using df.reset_index()

Name: Date:

Lesson 2.1

Task 6: Selecting row

Code	Output

`d = [`
`  {"name": "Apple", "quantity": 3, 'type': 'Food'},`
`  {"name": "Pear", "quantity": 2, 'type': 'Food'},`
`  {"name": "Grape", "quantity": 5, 'type': 'Food'},`
`  {"name": "Kettle", "quantity": 1, 'type': 'Appliance'},`
`  {"name": "Pan", "quantity": 2, 'type': 'Appliance'},`
`  {"name": "Fan", "quantity": 3, 'type': 'Appliance'},`
`]`
 	
`df = pd.DataFrame(d)`
`df = df[df['name'] == 'Grape']`
`df`	

Name: Date:

Lesson 2.1

Task 7: Updating the row

Code

```
d = [
    {"name": "Apple", "quantity": 3, 'type': 'Food'},
    {"name": "Pear", "quantity": 2, 'type': 'Food'},
    {"name": "Grape", "quantity": 5, 'type': 'Food'},
    {"name": "Kettle", "quantity": 1, 'type':
'Appliance'},
    {"name": "Pan", "quantity": 2, 'type':
'Appliance'},
    {"name": "Fan", "quantity": 3, 'type':
'Appliance'},
]

df = pd.DataFrame(d)
df.loc[df['type'] == 'Appliance', ['quantity']] = 0
df
```

Output

Task 8: Updating the row

Code

```
d = [
    {"name": "Apple", "quantity": 3},
    {"name": "Pear", "quantity": 2},
    {"name": "Cucumber", "quantity": 5},
]

df = pd.DataFrame(d)
df['profit'] = df['quantity']
df['profit'] *= 1.2
df
```

Output

Name: Date:

Lesson 2.1

2.1 Practice – Revisiting Pandas

Exercise

Item	Type	Day 1	Day 2	Day 3	Day 4	Day 5
Avocado Milk	Milkshake	11	1	7	18	17
Grapefruit Tea	Tea	18	10	9	10	2
Mocha Frappe	Frappe	15	11	19	1	2
Java Chip Frappe	Frappe	2	6	4	10	17
Brown Sugar Milk	Milkshake	19	20	4	17	16
Mango Tea	Tea	20	17	17	12	19
Strawberry Tea	Tea	6	16	12	20	16
Green Tea Yakult	Tea	19	3	7	10	13
Earl Grey Milk Tea	Tea	18	18	17	15	11
Honey Latte Frappe	Frappe	7	12	8	5	16

Write a Python Code using Pandas Library to do the following

1. Load the data into a DataFrame - using a list of dictionaries
2. Select the row containing the 'Grapefruit Tea' item
3. Within the 'Grapefruit Tea' item, select 'Day 3'
4. Insert a new column 'Sales', containing the sum of Day 1, Day 2, Day 3, Day 4 & Day 5
5. Group the types & see the 'Sales' by type
6. Calculate the Earnings from the 'Sales', assuming that
    a. Tea cost $1.50 per sale
    b. Milkshake cost $3.00 per sale
    c. Frappe cost $5.00 per sale

Guided solution found in the next few pages

Name: Date:

Lesson 2.1

Task 1: Load the data into a DataFrame - using a list of dictionaries

```
bbt = [
  {
    'Item': 'Avocado Milk',
    'Type':  'Milkshake',
    'Day1': 11,
    'Day2': 1,
    'Day3': 7,
    'Day4': 18,
    'Day5': 17
  }
]
import pandas as pd

df = pd.DataFrame(bbt)
df
```

Repeat *for the rest*
For every Column, we will have a new key-value pair

Task 2: Select the row containing the 'Grapefruit Tea' item

```
df_2 = df.copy()
df_2 = df_2[(df_2['Item'] == 'Grapefruit Tea')]
df_2
```

We create a copy for the DataFrame so that it will not modify the original DataFrame for the future tasks.

df_2['Item'] == 'Grapefruit Tea' --- creates a filter that is applied to df_2 to only select the row

Name: Date:

Lesson 2.1

Task 3: Within the 'Grapefruit Tea' item, select 'Day 3'

```
df_3 = df.copy()
df_3 = df_3[(df_3['Item'] == 'Grapefruit Tea')]
df_3 = df_3[['Item','Day 3']]
df_3
```

df_3[(df_3['Item'] == 'Grapefruit Tea')] will contain the row.
To select the specific columns, df_3[['Item','Day 3']] is used

Task 4: Inserting the new column 'Sales'

```
df_4 = df.copy()
df_4['Sales'] = df_4['Day 1'] + df_4['Day 2'] + df_4['Day 3'] + df_4['Day 4'] +
df_4['Day 5']
df_4 = df_4[['Item','Type', 'Sales']]
df_4
```

We can add the different columns together by getting each series & adding them together

df_4['Day 1'] --- refers to the 'Day 1' column

df_4['Day 2'] --- refers to the 'Day 2' column

df_4['Day 3'] --- refers to the 'Day 3' column

df_4['Day 4'] --- refers to the 'Day 4' column

df_4['Day 5'] --- refers to the 'Day 5' column

df_4['Sales'] --- contains the sum. It is a newly created column

Name: Date:

Lesson 2.1

Task 5: Group the types & see the 'Sales' by type

```
df_5 = df_4.copy()
df_5 = df_5.groupby(by='Type').sum()
df_5
```

df_5.groupby(by='Type').sum() will set the 'Type' as the index

Task 6: Calculate the Earnings from the 'Sales'

```
df_6 = df_4.copy()
df_6['Earnings'] = df_6['Sales']
df_6.loc[df_6['Type'] == 'Tea', ['Earnings']] *= 1.5
df_6.loc[df_6['Type'] == 'Milkshake', ['Earnings']] *= 3
df_6.loc[df_6['Type'] == 'Frape', ['Earnings']] *= 5
df_6
```

df_6['Earnings'] = df_6['Sales'] makes a new column with the 'Sales' data

df_6.loc[df_6['Type'] == 'Tea', ['Earnings']] gets the column with the condition that df_6['Type'] == 'Tea'

*df_6.loc[df_6['Type'] == 'Tea', ['Earnings']] *= 1.5 essentially multiplies the column by 1.5*

Name: Date:

Lesson 2.2

2.2 Learning – Understanding Pandas DateTime Object

Sometimes when we handle data, we need to handle time-series data. These data might come in all formats, sometimes even in different columns. In this lesson, we will understand how to combine different columns to create a datetime object.

Month	Day	Year
1	1	2022
2	1	2022
3	1	2022
4	1	2022
5	1	2022

Task 1: Loading data into DataFrame

Code

```
dates = [
    {'Month': 1, 'Day': 1, 'Year': 2022},
    {'Month': 2, 'Day': 1, 'Year': 2022},
    {'Month': 3, 'Day': 1, 'Year': 2022},
    {'Month': 4, 'Day': 1, 'Year': 2022},
    {'Month': 5, 'Day': 1, 'Year': 2022}
]
import pandas as pd
df = pd.DataFrame(dates)
df
```

Output

Notice that the data loaded in is in Integer form

Name: Date:

Lesson 2.2

Task 2: Understanding the DataFrame

Code	Output
`df.info()`	

What is the Dtype of the Columns?
In order to combine the different columns, we need to convert the Dtype to a string before we can concatenate the string

Task 3: Converting the datatype

Code	Output

`df = df.astype({`
`"Day": str,`
`"Month": str,`
`"Year": str`
`})`	

It converts each Dtype into str

Name: Date:

Lesson 2.2

Task 4: Combining to create a string format MM/DD/YYYY

Code	Output
`df['Date_Raw'] = df['Month'] + '/' + df['Day'] + '/' + df['Year']`	

A new column 'Date_Raw' will be created and contain the data obtained from the other columns

Task 5: Using Pandas to convert to datetime object

Code	Output
`df['Date'] = pd.to_datetime(df['Date_Raw'])` `df`	

The Dtype of 'Date_Raw' is still a string. Therefore, we need to use pd.to_datetime(df['Date_Raw']) to convert it into a Pandas Datetime object.

Name: Date:

Lesson 2.2

2.2 Practice – Understanding Pandas DateTime Object

Exercise

Now we want to combine get the Date & Sales from the table below.

Sales	Month	Day	Year
10	1	15	2022
5	2	15	2022
3	3	15	2022
7	4	15	2022
2	5	15	2022

Write a Python code to

1. *Create a new column 'Date' that is a Pandas DateTime object*
2. *Filter the DataFrame to only show sales & Pandas Datetime object*

Guided solution found in the next few pages

Name: Date:

Lesson 2.2

Step 1: Load the data into a DataFrame - using a list of dictionaries

```
dates = [
    {'Sales':10, 'Month': 1, 'Day': 15, 'Year': 2022},
    {'Sales': 5, 'Month': 2, 'Day': 15, 'Year': 2022},
    {'Sales': 3, 'Month': 3, 'Day': 15, 'Year': 2022},
    {'Sales': 7, 'Month': 4, 'Day': 15, 'Year': 2022},
    {'Sales': 2, 'Month': 5, 'Day': 15, 'Year': 2022}
]

import pandas as pd

df = pd.DataFrame(dates)
df
```

Add a new key-value pair into the dictionary entries

Step 2: Convert the datatype from numeric to string

```
df = df.astype({
    "Day": str,
    "Month": str,
    "Year": str
    })
```

It converts each Dtype into str for the next step of string concatenation

Name: Date:

Lesson 2.2

Step 3: Combining to create a string format MM/DD/YYYY

Code	Output
`df['Date_Raw'] = df['Month'] + '/' + df['Day'] + '/' + df['Year']`	

A new column 'Date_Raw' will be created and contain the data obtained from the other columns

Step 4: Using Pandas to convert to datetime object

Code	Output
`df['Date'] = pd.to_datetime(df['Date_Raw'])` `df`	

The Dtype of 'Date_Raw' is still a string. Therefore, we need to use pd.to_datetime(df['Date_Raw']) to convert it into a Pandas Datetime object.

Step 5: Filter out 'Sales', 'Date' columns

Code	Output
`df = df[['Sales', 'Date']]` `df`	

Recall back to Lesson 2.1

Name: Date:

Lesson 2.3

2.3 Learning – Revisiting Matplotlib

Item	Type	Day 1	Day 2	Day 3	Day 4	Day 5
Avocado Milk	Milkshake	11	1	7	18	17
Grapefruit Tea	Tea	18	10	9	10	2
Mocha Frappe	Frappe	15	11	19	1	2
Java Chip Frappe	Frappe	2	6	4	10	17
Brown Sugar Milk	Milkshake	19	20	4	17	16
Mango Tea	Tea	20	17	17	12	19
Strawberry Tea	Tea	6	16	12	20	16
Green Tea Yakult	Tea	19	3	7	10	13
Earl Grey Milk Tea	Tea	18	18	17	15	11
Honey Latte Frappe	Frappe	7	12	8	5	16

In this example, we want to create a plot for the Avocado Milk throughout the 5 days – Bar Graph & Line Graph

Name: Date:

Lesson 2.3

Step 1: Loading in DataFrame for **only Avocado Milk**

Code	Output

`bbt = {`
`'Day': [1,2,3,4,5],`
`'Avocado Milk': [11, 1, 7, 18, 17]`
`}`
`import pandas as pd`
 	
`df = pd.DataFrame(bbt)`
`df`	

In this example, we're using a dictionary containing lists

Step 2: Import the relevant libraries

Code	Output
`import matplotlib.pyplot as plt` `import numpy as np`	

Step 3: Create the figure and axes

Code	Output
`fig1,ax1= plt.subplots(1,1)`	

Name: Date:

Lesson 2.3

Step 4: Setting the width of bar & indexes of the bar on the x-axis

Code	Output
`w = 0.2` `x = np.asarray([1,2,3,4,5])`	

w is the width, x is the indexes of the plot

x = np.asarray([1,2,3,4,5]) means I'm creating the indexes on the x-axis

Step 5: Plot the bar graphs

Code	Output
`ax1.bar(x, df['Avocado Milk'], width = w)`	

ax1.bar() creates a bar graph. It requires the x indexes, the heights and width.

The x indexes give the information on where to place the individual bars whereas the heights are essentially the value of each bar on the scale.

Width refers to the width of the bar. To make the bar narrower in width, w must be less than 1.

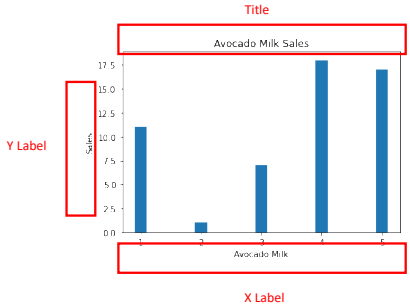
Name: Date:

Lesson 2.3

Step 6: Set the title & x-label, y-label

Code	Output

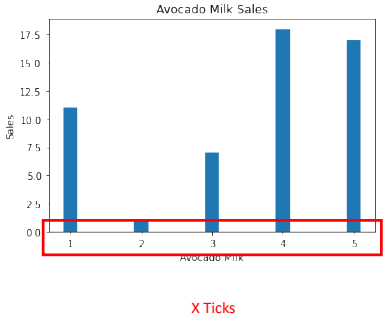
`ax1.set_title("Avocado Milk Sales")`
`ax1.set_xlabel("Avocado Milk")`
`ax1.set_ylabel("Sales")`	



Step 7: Set the x ticks

Code	Output
`ax1.set_xticks([1,2,3,4,5])`	

Setting x ticks is the labels on the x-axis



Step 8: Display the plot

Code	Output
`plt.show()`	

Name: Date:

Lesson 2.3

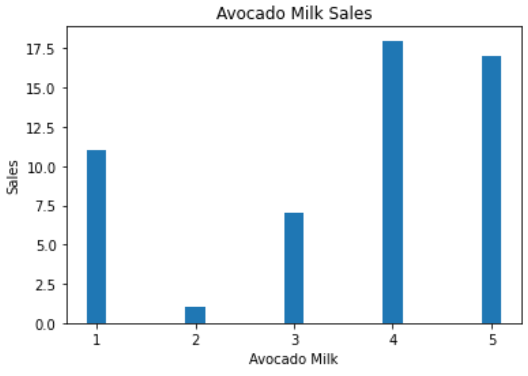
Full:

Code

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

bbt =  {
    'Day': [1,2,3,4,5],
    'Avocado Milk': [11, 1, 7, 18, 17]
  }

df = pd.DataFrame(bbt)
fig1,ax1= plt.subplots(1,1)

w = 0.2
x = np.asarray([1,2,3,4,5])

ax1.bar(x, df['Avocado Milk'], width = w)

ax1.set_title("Avocado Milk Sales")
ax1.set_xlabel("Avocado Milk")
ax1.set_ylabel("Sales")

ax1.set_xticks([1,2,3,4,5])

plt.show()
```

Output

Expected Graph



Name: Date:

Lesson 2.3

For plotting line graphs, we can repeat steps 1 – 3

Task 1: Loading in DataFrame for **only Avocado Milk**

Code	Output

```
bbt =  {
    'Day': [1,2,3,4,5],
    'Avocado Milk': [11, 1, 7, 18, 17]
  }
import pandas as pd

df = pd.DataFrame(bbt)
df
```

In this example, we're using a dictionary containing lists. This is because we want the following format for the dictionary.

	Day	Avocado Milk
0	1	11
1	2	1
2	3	7
3	4	18
4	5	17

Task 2: Import the relevant libraries

Code	Output

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

Numpy is not necessary here

Name: Date:

Lesson 2.3

Task 3: Create the figure and axes

Code	Output
`fig1,ax1= plt.subplots(1,1)`	

Task 4: Plot the line graph

Code	Output
`ax1.plot(x, df['Avocado Milk'])`	

x is the x-axis;

df['Avocado Milk'] is the y-axis

Task 5: Set the title, x-label & y-label

Code	Output

`ax1.set_title("Avocado Milk Sales")`
`ax1.set_xlabel("Avocado Milk")`
`ax1.set_ylabel("Sales")`	

Task 6: Set x ticks

Code	Output
`ax1.set_xticks([1,2,3,4,5])`	

Task 7: Display the plot

Code	Output
`plt.show()`	

Name: Date:

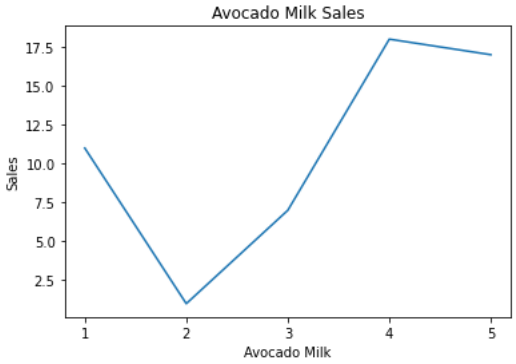
Lesson 2.3

Full:

Code	Output

`import matplotlib.pyplot as plt`
 	
`fig1,ax1= plt.subplots(1,1)`
 	
`ax1.plot(x, df['Avocado Milk'])`
`ax1.set_title("Avocado Milk Sales")`
`ax1.set_xlabel("Avocado Milk")`
`ax1.set_ylabel("Sales")`
`ax1.set_xticks([1,2,3,4,5])`
 	
`plt.show()`	

Expected Graph

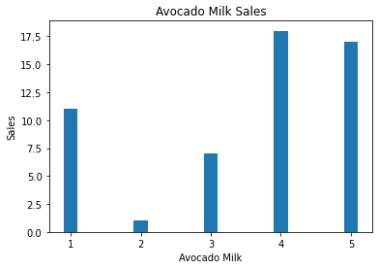
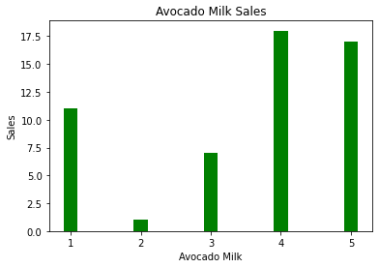
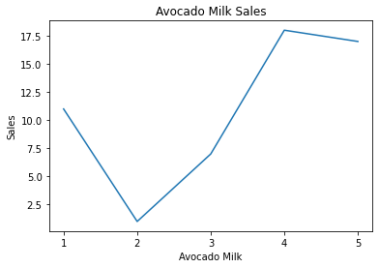
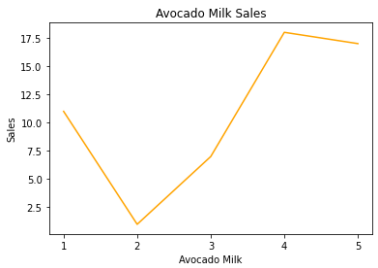


Name: Date:

Lesson 2.3

Decorating the graphs – uses examples from the previous section

Setting Colour

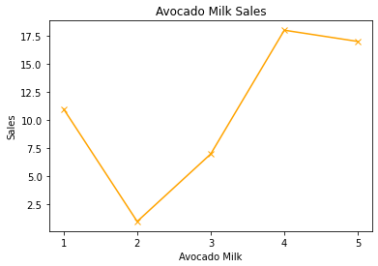
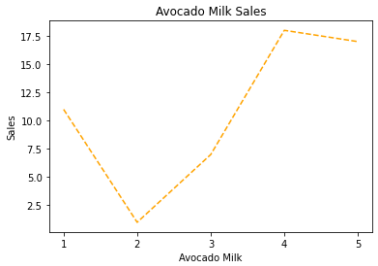
`ax1.bar(x, df['Avocado Milk'], width = w)`	
`ax1.bar(x, df['Avocado Milk'], width = w, color="Green")`	
`ax1.plot(x, df['Avocado Milk'])`	
`ax1.plot(x, df['Avocado Milk'], color="Orange")`	

Name: Date:

Lesson 2.3

Decorating the graphs – uses examples from the previous section

Setting Markers & Linestyle (Only Applicable to line graph not bar graph)

`ax1.plot(x, df['Avocado Milk'], color="Orange", marker='x')`	
`ax1.plot(x, df['Avocado Milk'], color="Orange", linestyle='--')`	

Name: Date:

Lesson 2.3

Decorating the graphs – List

Marker	Description
'o'	Circle
'*'	Star
'.'	Point
','	Pixel
'x'	X
'X'	X (filled)
'+'	Plus
'P'	Plus (filled)
's'	Square
'D'	Diamond
'd'	Diamond (thin)
'p'	Pentagon
'H'	Hexagon
'h'	Hexagon
'v'	Triangle Down
'^'	Triangle Up
'<'	Triangle Left
'>'	Triangle Right
'1'	Tri Down

Name: Date:

Lesson 2.3

Decorating the graphs – List

Style	Or
'solid' (default)	'-'
'dotted'	':'
'dashed'	'--'
'dashdot'	'-.'
'None'	'' or ' '

Color Syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Name: Date:

Lesson 2.3

2.3 Practice – Revisiting Matplotlib

Exercise

Item	Type	Day 1	Day 2	Day 3	Day 4	Day 5
Avocado Milk	Milkshake	11	1	7	18	17
Grapefruit Tea	Tea	18	10	9	10	2
Mocha Frappe	Frappe	15	11	19	1	2
Java Chip Frappe	Frappe	2	6	4	10	17
Brown Sugar Milk	Milkshake	19	20	4	17	16
Mango Tea	Tea	20	17	17	12	19
Strawberry Tea	Tea	6	16	12	20	16
Green Tea Yakult	Tea	19	3	7	10	13
Earl Grey Milk Tea	Tea	18	18	17	15	11
Honey Latte Frappe	Frappe	7	12	8	5	16

Write a Python Code to

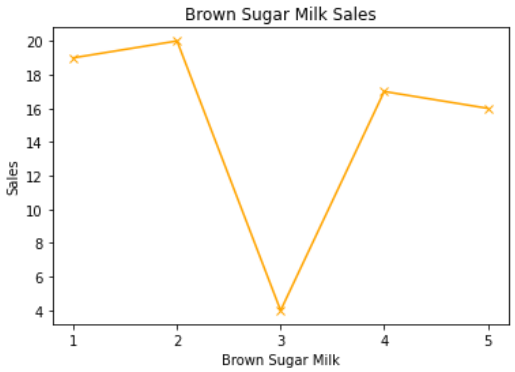
1. Create a line plot for **Brown Sugar Milk –** Orange in Color, with x as Marker
2. Create a line plot for **Mocha Frappe –** Black in Color, with '--' as Linestyle
3. Create a bar plot for **Green Tea Yakult –** Green in Color

Name: Date:

Lesson 2.3

Task 1: Create a line plot for **Brown Sugar Milk**

```
import pandas as pd
import matplotlib.pyplot as plt

bbt =  {
    'Day': [1,2,3,4,5],
    'Brown Sugar Milk': [19, 20, 4, 17, 16]
  }

df = pd.DataFrame(bbt)
fig1,ax1= plt.subplots(1,1)
ax1.plot(x, df['Brown Sugar Milk'], color="Orange", marker='x')

ax1.set_title("Brown Sugar Milk Sales")
ax1.set_xlabel("Brown Sugar Milk")
ax1.set_ylabel("Sales")
ax1.set_xticks([1,2,3,4,5])

plt.show()
```

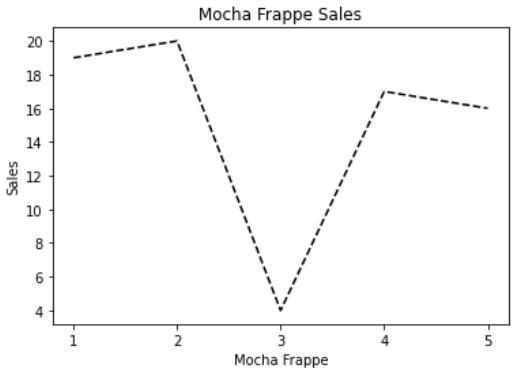


Name: Date:

Lesson 2.3

Task 2: Create a line plot for **Mocha Frappe**

```
import matplotlib.pyplot as plt
import pandas as pd

bbt =  {
    'Day': [1,2,3,4,5],
    'Mocha Frappe': [19, 20, 4, 17, 16]
  }

df = pd.DataFrame(bbt)
fig1,ax1= plt.subplots(1,1)
ax1.plot(x, df['Mocha Frappe'], color="Black", linestyle='--')

ax1.set_title("Mocha Frappe Sales")
ax1.set_xlabel("Mocha Frappe")
ax1.set_ylabel("Sales")
ax1.set_xticks([1,2,3,4,5])

plt.show()
```

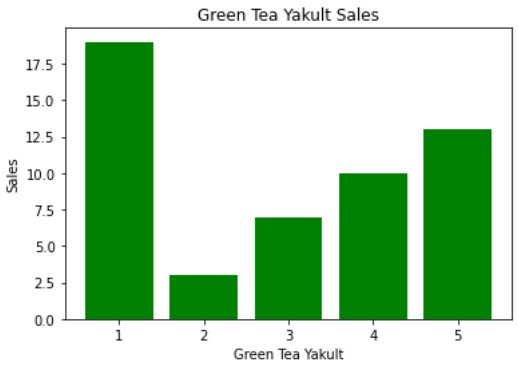


Name: Date:

Lesson 2.3

Task 3: Create a bar plot for **Green Tea Yakult**

```
import pandas as pd
import matplotlib.pyplot as plt

bbt =  {
    'Day': [1,2,3,4,5],
    'Green Tea Yakult': [19, 3, 7, 10, 13]
  }

df = pd.DataFrame(bbt)

fig1,ax1= plt.subplots(1,1)
ax1.bar(x, df['Green Tea Yakult'], color="Green")

ax1.set_title("Green Tea Yakult Sales")
ax1.set_xlabel("Green Tea Yakult")
ax1.set_ylabel("Sales")
ax1.set_xticks([1,2,3,4,5])

plt.show()
```



Name: Date:

Day 3

	Lesson Plan
3	- Exploration of data analytics - Bar Graph, Line plot - Understanding the problem

Name: Date:

Lesson 3.1

3.1 Learning – Handling Online Data with Pandas

Task 1: Import online dataset

```
import pandas as pd

url ="https://raw.githubusercontent.com/tlc-
datascience/datasets/main/asia_city_temperature.csv"

df = pd.read_csv(url)
df
```

When you enter the URL into the browser, you will find a CSV file containing the raw contents. Read CSV get the data. Alternatively, you can replace the URL to the path to your CSV file. In Google Colab, it will usually be the name of the file.

Task 2: Filter out the Columns

```
df = df[['Region', 'Country', 'Month', 'Day', 'Year', 'AvgTemperature']]
df
```

You can filter out the columns in the DataFrame.

Name: Date:

Lesson 3.1

Getting **average** temperature of **Singapore** between 2011 to 2015

Task 3: Filter out Singapore

```
df_singapore = df[(df["Country"] == 'Singapore')]
```

df["Country"] – checks in country column

df["Country"] == 'Singapore' – filters all Country that contains 'Singapore' as its value

df[(df["Country"] == 'Singapore')] – creates the DataFrame using the filter

Task 4: Grouping Data

```
df_singapore =
df_singapore.groupby(by=['Year']).mean()[['AvgTemperature']].loc[[i for i in
range(2011, 2016)]]
```

[i for i in range(2011, 2016)] –	*Creates a list containing 2011, 2012, 2013, 2014, 2015*
df_singapore.groupby(by=['Year']).mean() –	*Group all the values by the column 'Year' and gets the mean of the remaining values*
[['AvgTemperature']].loc[[i for i in range(2011, 2016)]] –	*Gets all the Average Temperature values where index is 2011, 2012, 2013, 2014, 2015*

Task 5: Convert Fahrenheit to Celsius

```
df_singapore["AvgTemperature"] = round(( df_singapore["AvgTemperature"] - 32 ) *
(5/9), 1)
df_singapore
```

*The formula between Fahrenheit to Celsius is (**x**°F − 32) × 5/9 = **y**°C*

*Therefore, (df_singapore["AvgTemperature"] - 32) * (5/9)*

Name: Date:

Lesson 3.1

Getting **average** temperature of **Japan** between 2011 to 2015

Task 6: Filter out Japan

```
df_japan = df[(df["Country"] == 'Japan')]
```

df["Country"] – checks in country column

df["Country"] == 'Japan' – filters all Country that contains 'Japan' as its value

df[(df["Country"] == 'Japan')] – creates the DataFrame using the filter

Task 7: Grouping Data

```
df_japan = df_japan.groupby(by=['Year']).mean()[['AvgTemperature']].loc[[i for i
in range(2011, 2016)]]
```

[i for i in range(2011, 2016)] –	*Creates a list containing 2011, 2012, 2013, 2014, 2015*
df_japan.groupby(by=['Year']).mean() –	*Group all the values by the column 'Year' and gets the mean of the remaining values*
[['AvgTemperature']].loc[[i for i in range(2011, 2016)]] –	*Gets all the Average Temperature values where index is 2011, 2012, 2013, 2014, 2015*

Task 8: Convert Fahrenheit to Celsius

```
df_japan["AvgTemperature"] = round(( df_japan["AvgTemperature"] - 32 ) * (5/9),
1)
df_japan
```

*The formula between Fahrenheit to Celsius is (**x**°F − 32) × 5/9 = **y**°C*

*Therefore, (df_japan ["AvgTemperature"] - 32) * (5/9)*

Name: Date:

Lesson 3.1

Combining average temperature of **Singapore & Japan** between 2011 to 2015

Task 9: Combining Singapore & Japan DataFrame

```
df_s = pd.DataFrame()
df_s['Singapore'] = df_singapore['AvgTemperature']
df_s['Japan'] = df_japan['AvgTemperature']
df_s
```

Year	Singapore	Japan
2011	27.6	13.7
2012	27.6	13.4
2013	27.7	13.9
2014	27.7	13.4
2015	27.8	13.9

Name: Date:

Lesson 3.1

3.1 Practice – Handling Online Data with Pandas

Year	Singapore	Japan	South Korea	Vietnam
2011	27.6	13.7		
2012	27.6	13.4		
2013	27.7	13.9		
2014	27.7	13.4		
2015	27.8	13.9		

Using Pandas Library, find the average annual temperature for South Korea & Vietnam between 2011 to 2015. **Fill up the table**

Guided solution found in the next few pages

Name: Date:

Lesson 3.1

Getting **average** temperature of **South Korea** between 2011 to 2015

Task 1: Filter out South Korea

```
df_south_korea = df[(df["Country"] == 'South Korea')]
```

df["Country"] – checks in country column

df["Country"] == 'Japan' – filters all Country that contains 'Japan' as its value

df[(df["Country"] == 'Japan')] – creates the DataFrame using the filter

Task 2: Group By

```
df_south_korea =
df_south_korea.groupby(by=['Year']).mean()[['AvgTemperature']].loc[[i for i in
range(2011, 2016)]]
```

[i for i in range(2011, 2016)] –	*Creates a list containing 2011, 2012, 2013, 2014, 2015*
df_south_korea.groupby(by=['Year']).mean() –	*Group all the values by the column 'Year' and gets the mean of the remaining values*
[['AvgTemperature']].loc[[i for i in range(2011, 2016)]] –	*Gets all the Average Temperature values where index is 2011, 2012, 2013, 2014, 2015*

Task 3: Convert Fahrenheit to Celsius

```
df_south_korea["AvgTemperature"] = round(( df_south_korea["AvgTemperature"] - 32
) * (5/9), 1)
df_south_korea
```

*The formula between Fahrenheit to Celsius is (**x**°F − 32) × 5/9 = **y**°C*

*Therefore, (df_south_korea ["AvgTemperature"] - 32) * (5/9)*

Name: Date:

Lesson 3.1

Getting **average** temperature of **Vietnam** between 2011 to 2015

Task 4: Filter out Vietnam

```
df_vietnam = df[(df["Country"] == 'Vietnam')]
```

Task 5: Group by

```
df_vietnam = df_vietnam.groupby(by=['Year']).mean()[['AvgTemperature']].loc[[i
for i in range(2011, 2016)]]
```

[i for i in range(2011, 2016)] –	*Creates a list containing 2011, 2012, 2013, 2014, 2015*
df_south_korea.groupby(by=['Year']).mean() –	*Group all the values by the column 'Year' and gets the mean of the remaining values*
[['AvgTemperature']].loc[[i for i in range(2011, 2016)]] –	*Gets all the Average Temperature values where index is 2011, 2012, 2013, 2014, 2015*

Task 6: Convert Fahrenheit to Celsius

```
df_vietnam["AvgTemperature"] = round(( df_vietnam["AvgTemperature"] - 32 ) *
(5/9), 1)
df_vietnam
```

*The formula between Fahrenheit to Celsius is (**x**°F – 32) × 5/9 = **y**°C*

*Therefore, (df_vietnam ["AvgTemperature"] - 32) * (5/9)*

Name: Date:

Lesson 3.1

Combining **average** temperature of **Singapore, Japan, South Korea & Vietnam** between 2011 to 2015

Task 7: Combining Singapore, Japan, South Korea & Vietnam DataFrame

```
df_s = pd.DataFrame()
df_s['Singapore'] = df_singapore['AvgTemperature']
df_s['Japan'] = df_japan['AvgTemperature']
df_s['South Korea'] = df_south_korea['AvgTemperature']
df_s['Vietnam'] = df_vietnam['AvgTemperature']
df_s
```

Year	Singapore	Japan	South Korea	Vietnam
2011	27.6	13.7	**11.5**	**23.1**
2012	27.6	13.4	**11.5**	**24.0**
2013	27.7	13.9	**11.7**	**23.3**
2014	27.7	13.4	**12.3**	**23.9**
2015	27.8	13.9	**12.4**	**23.8**

Name: Date:

Lesson 3.2

3.2 Learning – Bar Graph, Line Graph

Year	Singapore	Japan	South Korea	Vietnam
2011	27.6	13.7	11.5	23.1
2012	27.6	13.4	11.5	24.0
2013	27.7	13.9	11.7	23.3
2014	27.7	13.4	12.3	23.9
2015	27.8	13.9	12.4	23.8

In this example, we will write a Python Code using Matplotlib to

1. Create a line plot for **Singapore –** Red in Color, with ‘o’ as Marker, ‘dotted’ as Linestyle
2. Create a bar plot for **Japan –** Blue in Color

Name: Date:

Lesson 3.2

Create a line plot for **Singapore**

Step 1: Load in DataFrame into df_s

Step 2: Import the libraries

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

Step 3: Create the figure and axes

```
fig1,ax1= plt.subplots(1,1,figsize=(10,5))
```

Step 4: Plot the line graph

```
ax1.plot(df_s.index, df_s['Singapore'], color='R', marker='o',
linestyle='dotted')
```

Attributes – color='R', marker='o', linestyle='dotted'

Name: Date:

Lesson 3.2

Step 5: Set title & x-label, y-label

```
ax1.set_title("Singapore")
ax1.set_xlabel("Year")
ax1.set_ylabel("Temperature")
```

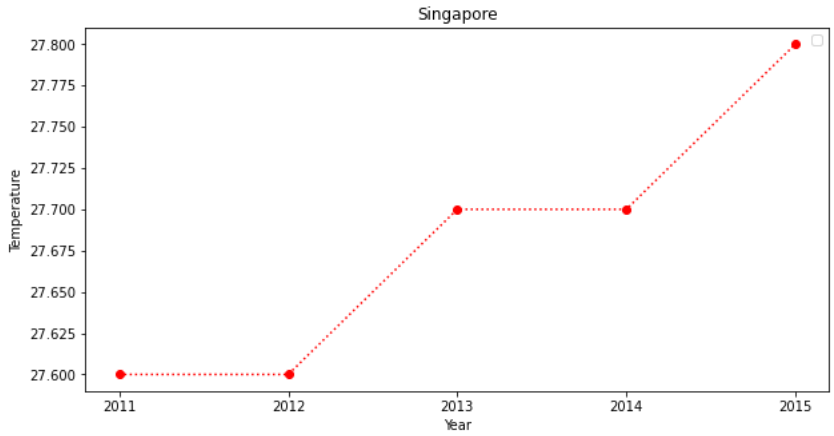
Step 6: Set the x ticks

```
ax1.set_xticks([i for i in range(2011, 2016)])
```

[i for i in range(2011, 2016)] gives [2011, 2012, 2013, 2014, 2015]

Step 7: Display the plot

```
plt.show()
```



Name: Date:

Lesson 3.2

Create a bar plot for **Japan**

Step 1: Load in DataFrame

Step 2: Import the libraries

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

Step 3: Create the figure and axes

```
fig1,ax1= plt.subplots(1,1,figsize=(10,5))
```

Step 4: Setting the width of the bar

```
w = 0.5
x = np.asarray([i for i in range(2011, 2016)])
```

w is not 1 because 1 means it occupies the entire space. It is only useful when we are doing single bar plots side by side like a histogram

Step 5: Plot the bar graph

```
ax1.bar(df_s.index, df_s['Japan'], color='Blue', width=w)
```

Name: Date:

Lesson 3.2

Step 6: Set the title & x-label, y-label

```
ax1.set_title("Japan")
ax1.set_xlabel("Year")
ax1.set_ylabel("Temperature")
```

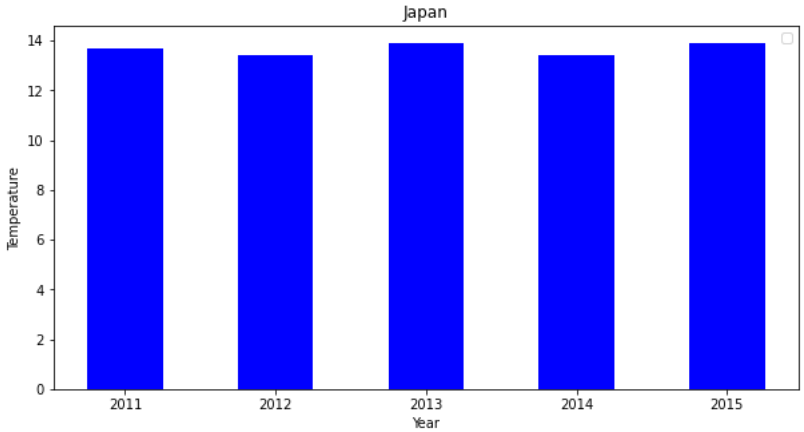
Step 7: Set x ticks

```
ax1.set_xticks([i for i in range(2011, 2016)])
```

[i for i in range(2011, 2016)] gives [2011, 2012, 2013, 2014, 2015]

Step 8: Display the graph

```
plt.show()
```



Name: Date:

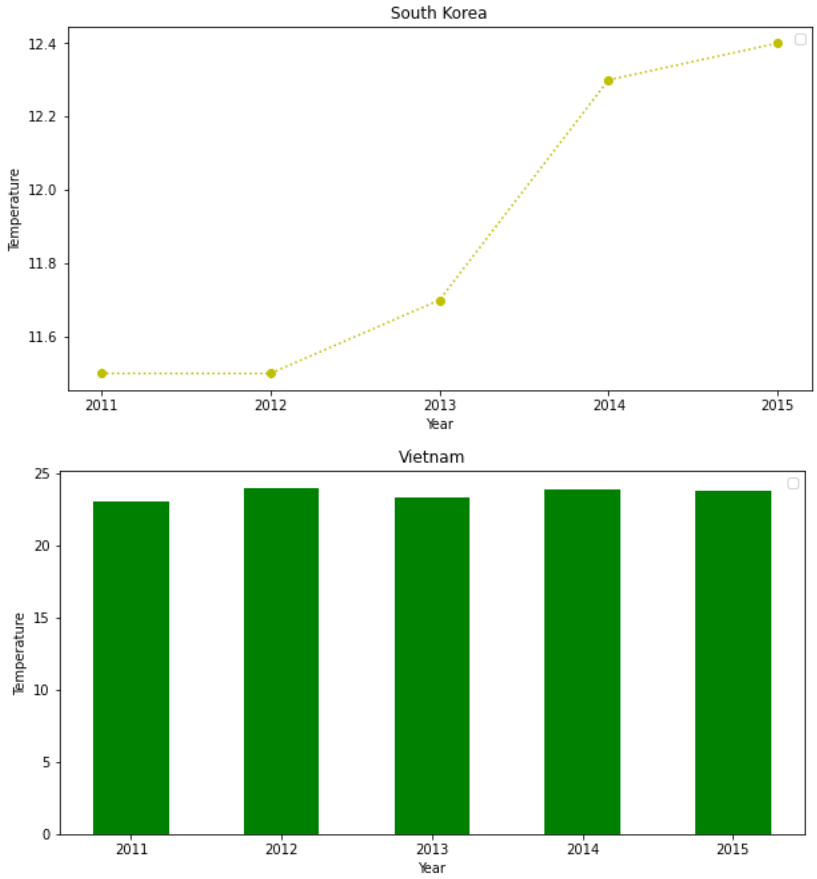
Lesson 3.2

3.2 Practice – Bar Graph, Line Graph

Year	Singapore	Japan	South Korea	Vietnam
2011	27.6	13.7	11.5	23.1
2012	27.6	13.4	11.5	24.0
2013	27.7	13.9	11.7	23.3
2014	27.7	13.4	12.3	23.9
2015	27.8	13.9	12.4	23.8

Write a Python Code using Matplotlib to

1. Create a line plot for **South Korea –** Purple in Color, with 'o' as Marker, 'dotted' as Linestyle
2. Create a bar plot for **Vietnam –** Green in Color



No Guided Solution Provided

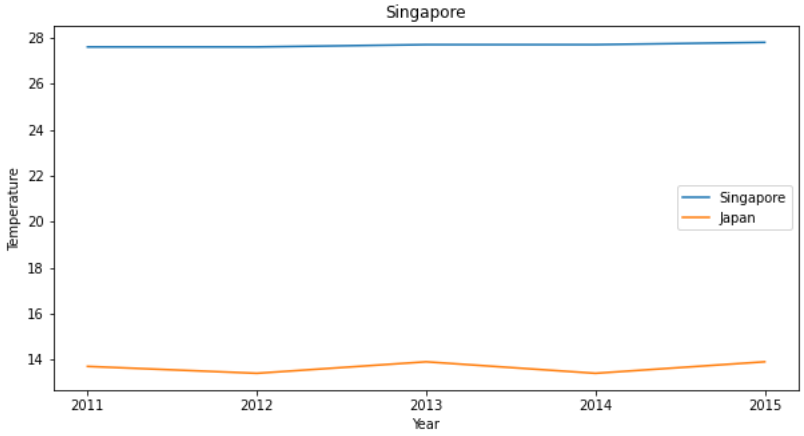
Name: Date:

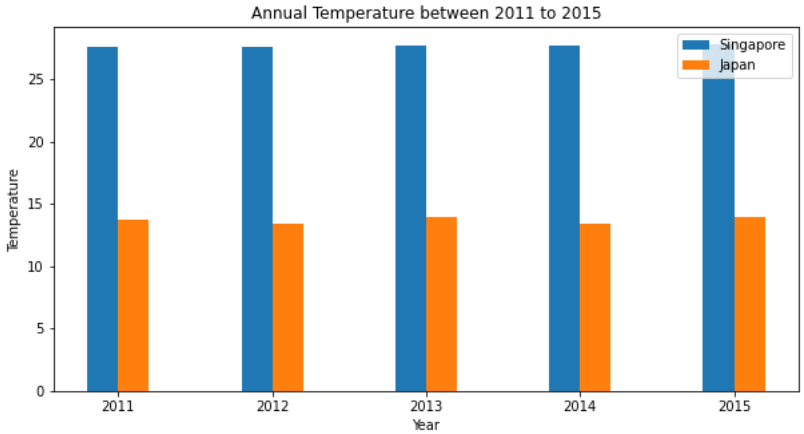
Lesson 3.3

3.3 Learning – Multiplot

Year	Singapore	Japan	South Korea	Vietnam
2011	27.6	13.7	11.5	23.1
2012	27.6	13.4	11.5	24.0
2013	27.7	13.9	11.7	23.3
2014	27.7	13.4	12.3	23.9
2015	27.8	13.9	12.4	23.8

In this example, we will write a Python Code using Matplotlib to create a multiplot for Singapore & Japan in line graphs & bar graphs





Name: Date:

Lesson 3.3

Line graph

Task 1: Load in DataFrame into df_s

Task 2: Import libraries

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

Task 3: Create Figure & Axes

```
fig1,ax1= plt.subplots(1,1,figsize=(10,5))
```

Task 4: Plot the line graph

```
ax1.plot(df_s.index, df_s['Singapore'], label = "Singapore")
ax1.plot(df_s.index, df_s['Japan'], label = "Japan")
```

To overlay the different graphs on the same line plot, we will just need to plot and give them different labels

Name: Date:

Lesson 3.2

Task 5: Set the title, x-label & y label

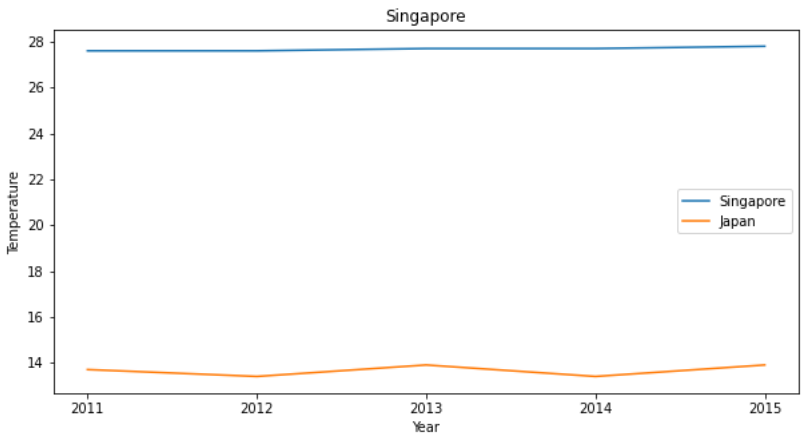
```
ax1.set_title("Annual Temperature between 2011 to 2015 ")
ax1.set_xlabel("Year")
ax1.set_ylabel("Temperature")
```

Task 6: Set x ticks

```
ax1.set_xticks([i for i in range(2011, 2016)])
```

Task 7: Display the plot & legend

```
plt.legend()
plt.show()
```



Name: Date:

Lesson 3.3

Bar graph

Task 1: Load in DataFrame

Task 2: Import relevant libraries

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

Numpy is required because we want to create

Task 3: Create the figure and axes

```
fig1,ax1= plt.subplots(1,1,figsize=(10,5))
```

Name: Date:

Lesson 3.3

Task 4: Setting the width of bar & indexes of the bar on the x-axis

```
w = 0.2
x = np.asarray([i for i in range(2011, 2016)])
```

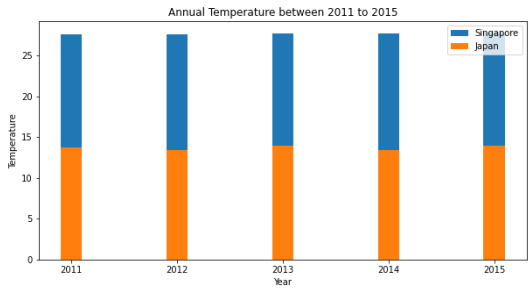
w is not 1 because 1 means it occupies the entire space. It is only useful when we are doing single bar plots side by side like a histogram

Task 5: Plotting the bar graphs

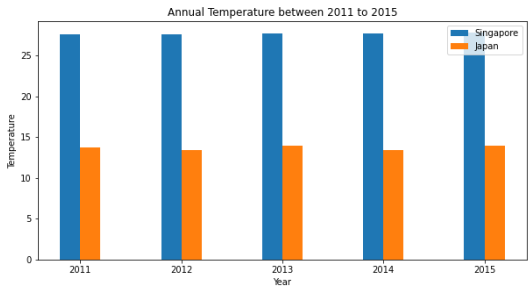
```
ax1.bar(x-w/2, df_s['Singapore'], width = w, label = "Singapore")
ax1.bar(x+w/2, df_s['Japan'], width = w, label = "Japan")
```

x-w/2 & x+w/2 are essentially the offset. If we were to use x for both, we are overlaying them directly on each other which is not what we want to display. Therefore, +w/2 and -w/2 helps to shift the bar to the left or right of each other

Without offset



With offset



Name: Date:

Lesson 3.3

*matplotlib.pyplot.bar(x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)*

How can I effectively have multiple bars graph on the same plot?

We will need to understand the parameters when plotting bar

1. *x*
   - *The x coordinates of the bars*
2. *height*
   - *The height(s) of the bars*
3. *Width*
   - *The width(s) of the bars*

Normally, you will notice that we do

```
w = 0.2
x = np.asarray([i for i in range(2011, 2016)])
ax1.bar(x, df_s['Singapore'], width = w)
```

This means that we are creating an array containing 2011, 2012, 2013, 2014, 2015. This is the x-coordinates for the respective bars.

df_s['Singapore'] will give the respective value or height value for each x-coordinate 2011, 2012, 2013, 2014, 2015.

Width will indicate how wide it is, left to right

When we want multiplot, we do not want to have them all align to specific x coordinate otherwise it will overlay each other. That is why

```
w = 0.2
x = np.asarray([i for i in range(2011, 2016)])
ax1.bar(x-0.1, df_s['Singapore'], width = w)
ax1.bar(x+0.1, df_s['Japan'], width = w)
```

x – 0.1 means shift to the left by 0.1
x + 0.1 means shift to the right by 0.1

Name: Date:

Lesson 3.3

Task 6: Set the title, x-label & y-label

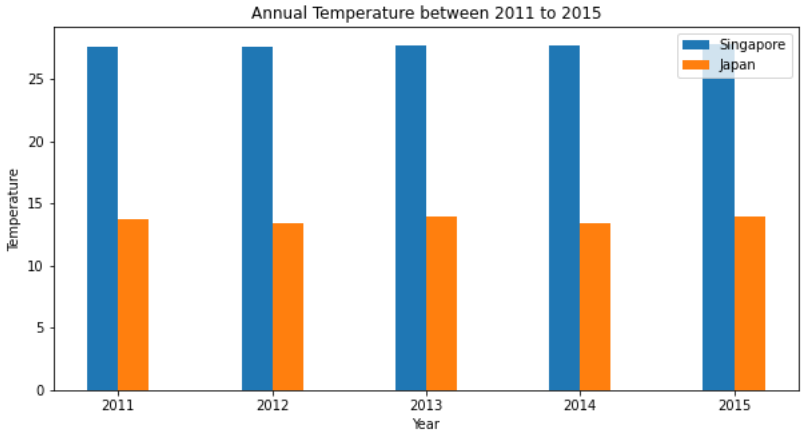
```
ax1.set_title("Annual Temperature between 2011 to 2015 ")
ax1.set_xlabel("Year")
ax1.set_ylabel("Temperature")
```

Task 7: Set x-ticks

```
ax1.set_xticks([i for i in range(2011, 2016)])
```

Task 8: Display the plot

```
plt.show()
```



Name: Date:

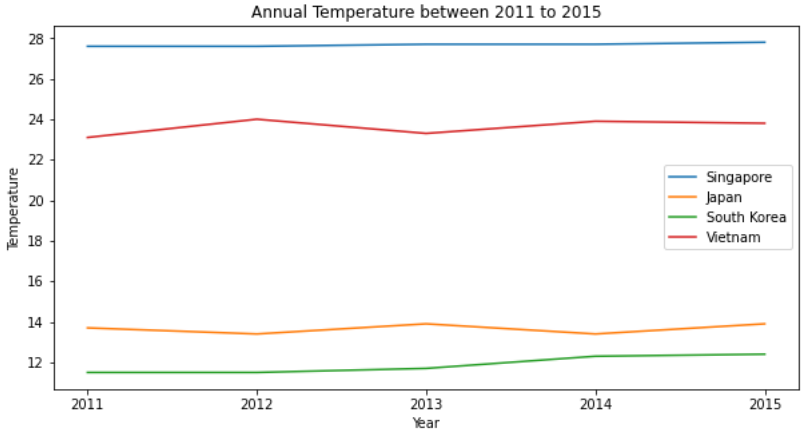
Lesson 3.3

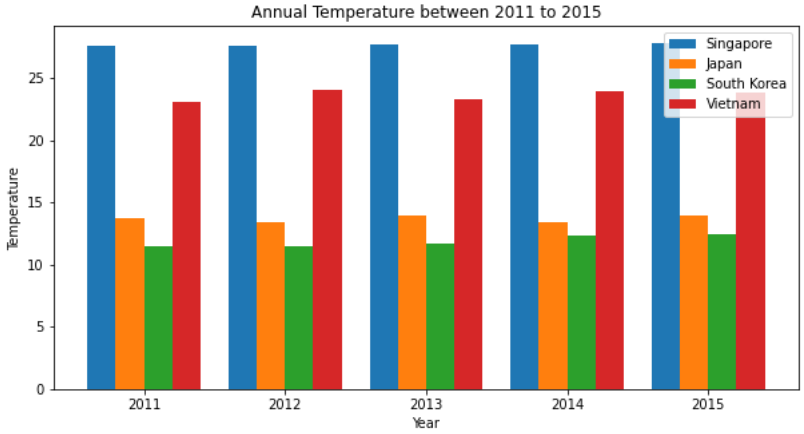
3.3 Practice – Multiplot

Year	Singapore	Japan	South Korea	Vietnam
2011	27.6	13.7	11.5	23.1
2012	27.6	13.4	11.5	24.0
2013	27.7	13.9	11.7	23.3
2014	27.7	13.4	12.3	23.9
2015	27.8	13.9	12.4	23.8

Write a Python Code using Matplotlib to create a multiplot for Singapore, Japan, South Korea & Vietnam in line graphs & bar graphs.

Do note that for the bar graph, we will need to consider the offset and width to prevent overlaps.



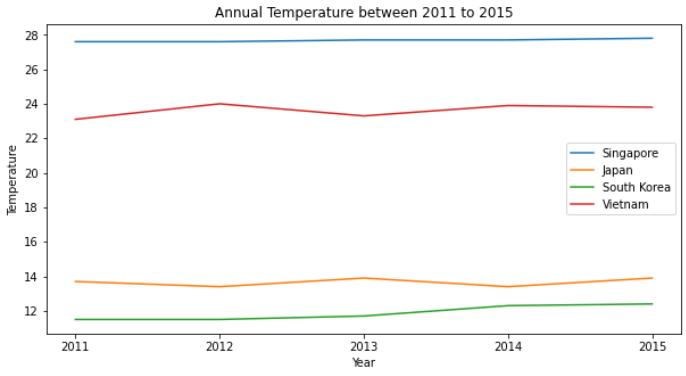


No Guided Solution Provided

Name: Date:

Lesson 3.3

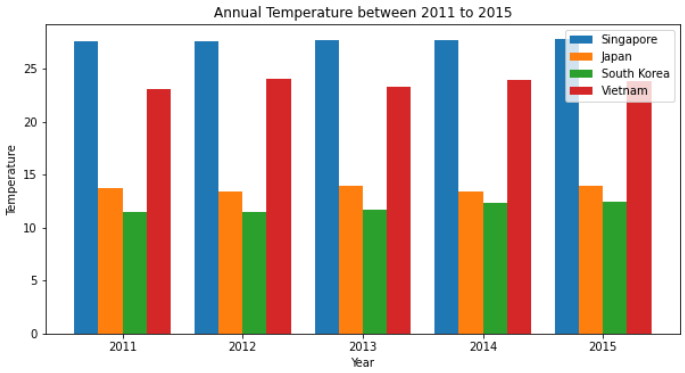
```
# Step 2: Import the relevant libraries
import matplotlib.pyplot as plt

# Step 3: Create the figure and axes
fig1,ax1= plt.subplots(1,1, figsize=(10,5))

# Step 4: Plot the line graphs
ax1.plot(df_s.index, df_s['Singapore'], label='Singapore')
ax1.plot(df_s.index, df_s['Japan'], label='Japan')
ax1.plot(df_s.index, df_s['South Korea'], label='South Korea')
ax1.plot(df_s.index, df_s['Vietnam'], label='Vietnam')

# Step 5: Set the title, x-label & y-label
ax1.set_title("Annual Temperature between 2011 to 2015")
ax1.set_xlabel("Year")
ax1.set_ylabel("Temperature")

# Step 6: Set x-ticks
ax1.set_xticks([i for i in range(2011, 2016)])

# Step 7: Display the plot
plt.legend()
plt.show()
```



Name: Date:

Lesson 3.3

```
# Step 2: Import the relevant libraries
import matplotlib.pyplot as plt
import numpy as np

# Step 3: Create the figure and axes
fig1,ax1= plt.subplots(1,1, figsize=(10,5))

# Step 4: Setting the width of bar & indexes of the bar on the x-axis
w = 0.2
x = np.asarray([i for i in range(2011, 2016)])

# Step 5: Plot the bar graphs
ax1.bar(x-w/2*3, df_s['Singapore'], width = w, label = "Singapore")
ax1.bar(x-w/2, df_s['Japan'], width = w, label = "Japan")
ax1.bar(x+w/2, df_s['South Korea'], width = w, label = "South Korea")
ax1.bar(x+w/2*3, df_s['Vietnam'], width = w, label = "Vietnam")

# Step 6: Set the title, x-label & y-label
ax1.set_title("Japan")
ax1.set_xlabel("Year")
ax1.set_ylabel("Temperature")

# Step 7: Set x-ticks
ax1.set_xticks([i for i in range(2011, 2016)])

# Step 8: Display the plot
plt.legend()
plt.show()
```



Name: Date:

Day 4

	Lesson Plan
4	- Data Analysis: Retrenched employees by industry and occupational Group

**Note: The CSV files obtained are from the data.gov.sg source and it is clean, therefore data cleaning is not required.*

https://data.gov.sg/dataset/retrenched-employees-by-industry-and-occupational-group-annual

Name: Date:

Lesson 4

Exercise

https://data.gov.sg/dataset/retrenched-employees-by-industry-and-occupational-group-annual
There are several files inside – let's use the file named "retrenchment-by-industry-level-2"

Exercise 1a:

Task 1:

Let's analyse the data from the CSV files using the following:

Write the output/description in the space below.

df.head()
df.tail()
df.info()
df.describe()
df.columns
In your own words, explain what the data is about Which type of graph do we use?

Name: Date:

Lesson 4.1

Recall steps 1 to 9

We would like to plot a bar graph to compare the retrenchment numbers against all the industry for the year 2021.

Keep in mind our goal for the plot:

1 bar graph of industry names (x-axis) vs retrenchment numbers (y-axis) for 2021

Let's start the code step by step:

Step 1:

```
import matplotlib.pyplot as plt
import pandas as pd
```

Step 2:

#Create 1 Pandas DataFrame for the entire CSV file.

2	df=pd.read_csv('dataname')
Ans	df =pd.read_csv('retrenchment-by-industry-level-2.csv') df

Step 3:

#Remove NA and "-"for value columns

3	df=df[df['columnname']!= 'na']
Ans	df = df[df['retrench'] != 'na']
Ans	df = df[df['retrench'] != '-']

Name: Date:

Lesson 4

Step 4:

#Change all columns you would like to plot in the graph with "numerical values" to numerics *(they are not in integers format)*

4	df["columnname"]=pd.to_number(df["columnname"])
Ans	df['retrench'] = pd.to_numeric(df['retrench'])

Let's go back to our goal for the plot:

1 line graph of industry names (x-axis) vs retrenchment numbers (y-axis) for 2021

What is all the data in our CSV file? How do we get the data for our x and y axis?

We have the main industry category – Manufacturing, Construction, Services, Others So What do we need to do? Filter the main industry category for year 2021 (Step 5) Sum up the numbers of the sub-category for each of main category (Step 6)

Step 5: Filter and extract the year

5	df_newname = df_oldname[(df_oldname["columnname"] == criteria & (df_oldname["columnname"]==criteria)]
Ans	df_2021 = df[(df['year'] == 2021)] df_2021

*Step 6: Sum the sub-categories of the industries

6	df_newname=df_newname.groupby(["columnname"]).sum()
Ans	df_2021 = (df_2021.groupby(['industry1']).sum()) df_2021

Name: Date:

Lesson 4.1

Step 7: Remove duplicates

7	df=df.drop_duplicates()
Ans	df_2021 = df_2021.drop_duplicates()

Step 8b: Plot bar graph

**Take note of the difference between what you have plot so far in the previous dataset*

9	fig1.ax1=plt.subplot(1,1,figsize=(10,5)) ax1.bar(df.index , df[”columnname”])
Ans	ax1.bar(df_2021.index ,df_2021["retrench"])

**Note: Bar graph has no plt.legend()*

Why do we use df.index above?

**Previous dataset*

Step 8: Plot line graph

9	fig1.ax1=plt.subplots(1,1,figsize=(10,5)) ax1.plot(df.”columnnmane” , df.”columnname”, label= “labelname”) plt.legend() plt.show()
Ans	*fig1.ax1=plt.subplots(1,1,figsize=(10,5))* *ax1.plot(df_malay.year, df_malay.percentage_progress_postsec, label="Malay")* *ax1.plot(df_indian.year, df_indian.percentage_progress_postsec, label="Indian")*

Name: Date:

Lesson 4

What is the difference between

df.columnname vs df[“columnname’]

When do we use which one?

Step 9: Display plot with labels

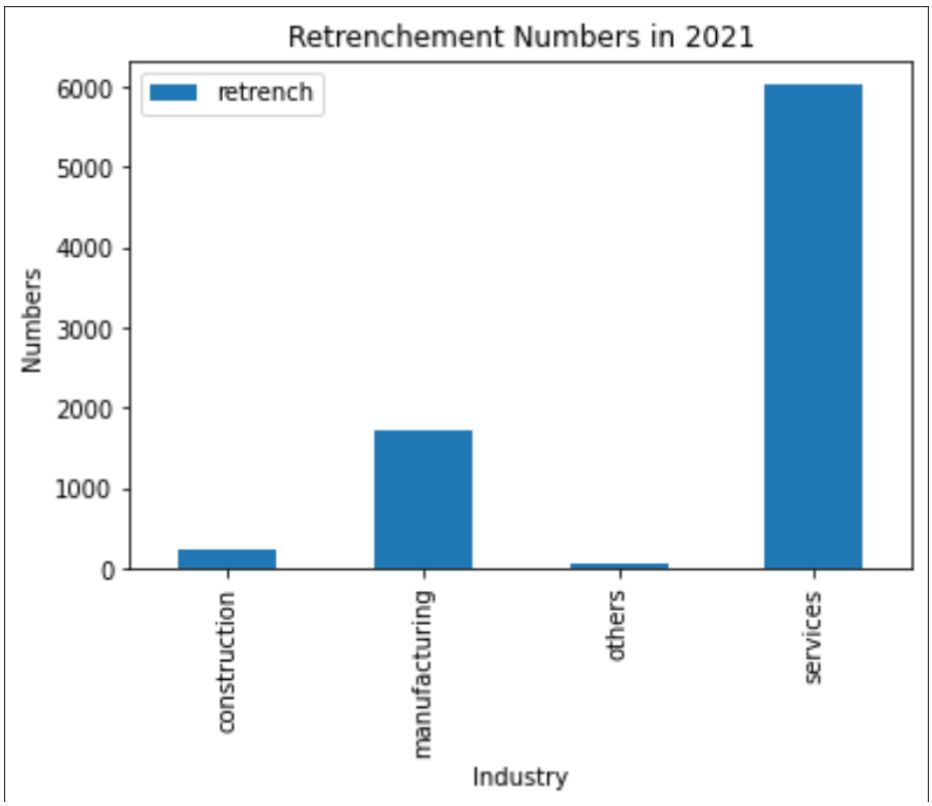
8	ax1.set_xlabel(“columnname”) ax1.set_ylabel(“columnname”) ax1.set_title(“titlename”) plt.show()
Ans	ax1.set_xlabel("Industry") ax1.set_ylabel("Retrenchment Numbers") ax1.set_title("Retrenchment") plt.show()

Name: Date:

Lesson 4

Task 1b:

Output:



Name: Date:

Lesson 4

Task 1b: (Independent)

Plot a bar graph to compare the retrenchment numbers against all the industry for the year 2020.

Note: Be sure to annotate each step to getting your graph and conclusion

Do the following analysis:
Question 1: Which industry had the highest retrenchment numbers in year 2020?

Name: Date:

Lesson 4

Task 2:

We would like to have 2 bar graphs in 1 plot to compare the retrenchment numbers against all between 2020 (covid) and 2019 (pre-covid)

Write your code here – Recap steps 1 to 9

Recap you will need to do Step 8a – Define x positions for 2 bar graphs

```
x=np. arange(len(df_2020.index))
```

Step 8b – Plot the graph

```
ax1.bar(x-0.2,df_2020.retrench,0.4,label=”2020”)
ax1.bar(x+0.2,df_2019.retrench,0.4,label=”2020”)
```

Step 9 – Display plot with labels

```
ax1.set_xlabel("Industry")
ax1.set_ylabel("Retrenchment Numbers")
ax1.set_title("Retrenchment 2020 vs 2019")
plt.xticks(x,df_2020,index)
plt.legend()
plt.show()
```

Conclusion:

For which industry were there higher retrenchment numbers in 2020 (covid) compared to 2019?

Name: Date:

Lesson 4

Task 3: (Independent)

We would like to have 2 bar charts in 1 plot to compare the retrenchment numbers in the different sub-industry under the Services industry for year 2020 vs year 2019

Note: Be sure to annotate each step to getting your graph and conclusion

Write your code here – Recap steps 1 to 9

Conclusion:

Which sub industry in the Service industry had a higher retrenchment number in 2020 vs 2019?

Which sub industry in the Service industry had a lower retrenchment number in 2020 vs 2019?

Name: Date:

Lesson 4

Task 4: (Independent)

We would like to have 2 line graphs in one plot to compare the retrenchment numbers in the manufacturing – food, beverage and tobacco and services- hotels and restaurant over the years 2017, 2018,2019,2020 and 2021

Note: Be sure to annotate each step to getting your graph and conclusion

Write your code here – Recap steps 1 to 9

Conclusion: Are there any corelation between the 2 sub industry?

Name: Date:

Lesson 4

Task 5: (Independent)

We would like to have 3 line graphs in one plot to compare the retrenchment numbers in the manufacturing- sub- industry – electronic, computer and optical products over the years from 2006 to 2020.

Note: Be sure to annotate each step to getting your graph and conclusion

Write your code here – Recap steps 1 to 9

Conclusion:

Which year did the manufacturing - electronic, computer and optical products, have the highest retrenchment number?

Which year did the manufacturing - electronic, computer and optical products, have the lowest retrenchment number?

Name: Date:

Lesson 4

Task 6: (Independent)

We would like to have a 3 line graphs in 1 plot to compare the retrenchment numbers in the services -sub- industry – wholesale and retail trade, accommodation and food services, real estate services from the years from 1998 to 2021.

Note: Be sure to annotate each step to getting your graph and conclusion

Write your code here – Recap steps 1 to 9