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: Graduate Employment Survey - NTU, NUS, SIT, SMU, SUSS & SUTD

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.

Step 1:

Create a list containing *Apple, Google, Facebook* in a variable called *tech*

```
tech = ['Apple', 'Google', 'Facebook']
```

Step 2:

Remove the *Facebook* from the list

```
tech.remove("Facebook")
```

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

Step 3:

Add in *Meta, Amazon, Netflix* to the list

```
tech.append("Meta")
tech.append("Amazon")
tech.append("Netflix")
```

Name: Date:

Lesson 1.1

Step 4:

Count the number of tech companies in the list

```
print(len(tech))
```

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 tech companies as a list

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

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.

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": "456",
    "alias": "the chosen one"
}
print(list(player.items()))
```

Output

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

Task 2: Getting Keys of dictionary

```
player = {
    "id": "456",
    "alias": "the chosen one"
}
print(list(player.keys()))
```

Output

Name: Date:

Lesson 1.1

Task 3: Getting Values of dictionary

```
player = {
    "id": "456",
    "alias": "the chosen one"
}
print(list(player.values()))
```

Output

Task 4: Update dictionary

```
player = {
    "id": "456",
    "alias": "the chosen one"
}
player.update({"status": "alive"})
print(player)
```

Output

Name: Date:

Lesson 1.1

Task 5: Remove item from dictionary

```
player = {
    "id": "456",
    "alias": "the chosen one",
    "state": "not ready"
}
player.pop("state") # Pop by key
print(player)
```

Output

Task 6: Modify value in dictionary

```
player = {
    "id": "456",
    "alias": "the chosen one",
    "status": "alive"
}
player["status"] = "win!" # By key
print(player)
```

Output

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

Task 7: Modify value in dictionary

```
player = {
    "id": "456",
    "alias": "the chosen one",
    "status": "alive"
}
Player.update({"status" : "win!"}
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. Do not count the white spaces.

Input
'squid game'
Output
{'s': 1, 'q': 1, 'u': 1, 'i': 1, 'd': 1, 'g': 1, 'a': 1, 'm': 1, 'e': 1}

Exercise 2

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

Input
[{"A": "S001"}, {"B": "S002"}, {"C": "S001"}, {"B": "S005"}, {"C":"S005"}, {"B":"S009"},{"C":"S007"}]
Output
['S001', 'S002', 'S005', 'S009', 'S007']

Exercise 3

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

Input
[['yellow', 3], ['blue', 4], ['yellow', 1], ['blue', 2], ['red', 5]]
Output
{'yellow': [1, 3], 'blue': [2, 4], 'red': [5]}

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
'squid game'
Output
{'s': 1, 'q': 1, 'u': 1, 'i': 1, 'd': 1, 'g': 1, 'a': 1, 'm': 1, 'e': 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 = 'squid game'
```

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

```
    if i != ' ':
      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
[{"A": "S001"}, {"B": "S002"}, {"C": "S001"}, {"B": "S005"}, {"C":"S005"}, {"B":"S009"},{"C":"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 = [{"A": "S001"}, {"B": "S002"}, {"C": "S001"}, {"B": "S005"}, {"C":"S005"},
{"B":"S009"},{"C":"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', 3], ['blue', 4], ['yellow', 1], ['blue', 2], ['red', 5]]
Output
{'yellow': [1, 3], 'blue': [2, 4], 'red': [5]}

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', 5]]
```

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: Sort the list whenever new data is entered

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

Step 7: Displaying the dictionary

```
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

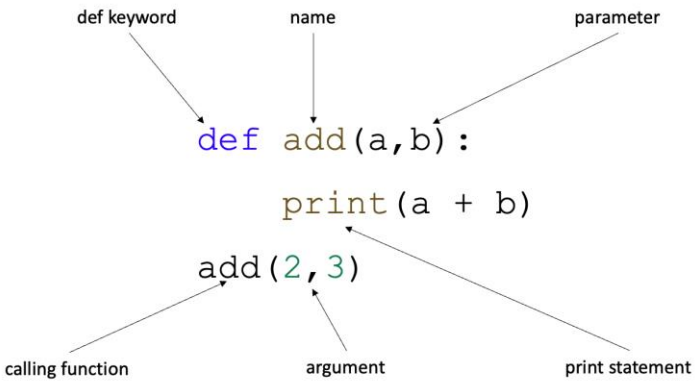
Rules for creating user-defined function

- 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 sub(a,b):
    print(a - b)
sub(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**?*	

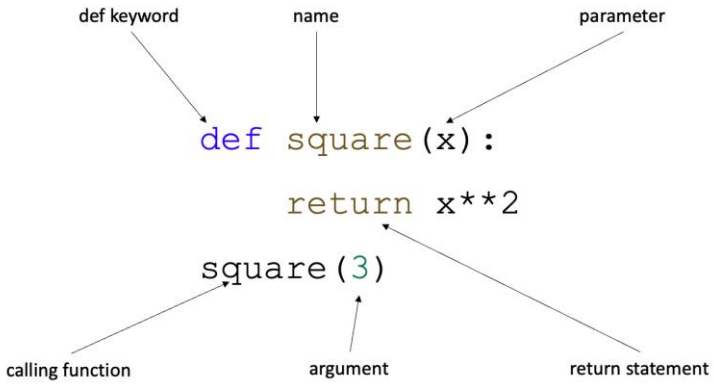
Task 2: Understanding the anatomy of a function

```
def exp(x,y):
    print(x ** y)
exp(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 cube(x):`
`  return x**3`
`cube(3)`
		There is no print statement. Cube(3) will contain a value

Task 2: Printing only within the function

Code	Output	Remarks

`def cube(x):`
`  print(x**3)`
`cube(3)`
		There is only 1 print statement. Cube(3) will contain None

Task 3: Returning with a printing outside the function

Code	Output	Remarks

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

Task 4: Printing with a printing outside the function

Code	Output	Remarks

`def cube(x):`
`  print(x**3)`
`print(cube(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 grading system in a command line interface (CLI).

Grades include 'A', 'B', 'C' and reject the other types of grades

The user requirements are as follows:

1. As a user, I want to add in new items to list
2. As a user, I want to count my items in my grades list
3. As a user, I want to remove wrong items from my grades 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 know the count of a requested grade.

Guided solution found in the next few pages

Name: Date:

Lesson 1.2

Exercise 1

Write a Python code to emulate a grading system in a command line interface (CLI).

Grades include 'A', 'B', 'C' and reject the other types of grades

The user requirements are as follows:

1. As a user, I want to add in new items to list
2. As a user, I want to count my items in my grades list
3. As a user, I want to remove wrong items from my grades list

Step 1: Understanding the problem

- Based on the user requirements, I need to be able to **add, 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 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 Grade:\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 list

Name: Date:

Lesson 1.2

Step 5: Displaying existing items

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

Step 6: Counting existing items

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

A built-in function is used.

Step 7: 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 8: 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 know the count of a requested grade.

Step 1: Understanding the problem

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

Step 2: Using list methods to count

```
l = ['A', 'A', 'B', 'A']


print(l.count('A'))
```

Step 3: Adding additional feature

```
    elif c == 'search':
        d = input("Enter grade:\t")
        if d in l:
            print(f'{d} appears {l.count(d)} times')
        else:
            print(f'{d} not found')
```

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 housemaster, I want to add new students into the database
2. As a housemaster, I want to know every student's name & gender (witch/wizard)
3. As a housemaster, I want to **randomly** assign them a house (Gryffindor, Hufflepuff, Ravenclaw, Slytherin)

Exercise 2

Add an additional filter feature to your Python code.

The additional user requirement is that:

1. As a busy housemaster, I want to filter out the students by their house (Gryffindor, Hufflepuff, Ravenclaw, Slytherin)

Guided solution found in the next few pages

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 housemaster, I want to add new students into the database
2. As a housemaster, I want to know every student's name & gender (witch/wizard)
3. As a housemaster, I want to **randomly** assign them a house (Gryffindor, Hufflepuff, Ravenclaw, Slytherin)

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, houses
- 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, houses

```
l = []
houses = ['Gryffindor', 'Hufflepuff', 'Ravenclaw', 'Slytherin']
```

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(witch/wizard):\t")

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

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, we will randomly select from the list – houses

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 housemaster, I want to filter out the students by their house (Gryffindor, Hufflepuff, Ravenclaw, Slytherin)

Step 1: Check if the input is a valid colour

```
h = input("Enter house:\t")
if h in houses:
```

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

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

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

Task 1: List of Dictionary Entries

Code

```
import pandas as pd

d = [
    {"name": "Lion", "quantity": 3},
    {"name": "Puma", "quantity": 12},
    {"name": "Meerkat", "quantity": 15},
]

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

Output

Task 2: Understanding a DataFrame

Code

```
dict2 = [
    {"name": "Lion", "quantity": 3},
    {"name": "Puma", "quantity": 12},
    {"name": "Meerkat", "quantity": 15},
]
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	Output

```
import pandas as pd

d = {
    "name": ["Lion", "Puma", "Meerkat"],
    "quantity": [3, 12, 15]
}

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

Task 4: Dictionary containing lists

Code	Output

```
# Grouping Data
d = [
    {"name": "Lion", "quantity": 3, 'diet':
'Carnivore'},
    {"name": "Puma", "quantity": 12, 'diet':
'Carnivore'},
    {"name": "Meerkat", "quantity": 15, 'diet':
'Omnivore'},
    {"name": "Elephant", "quantity": 7, 'diet':
'Herbivore'},
    {"name": "Zebra", "quantity": 6, 'diet':
'Herbivore'}
]

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

Name: Date:

Lesson 2.1

Task 5: Resetting the index

Code	Output

`# Grouping Data`
`d = [`
`{"name": "Lion", "quantity": 3, 'diet': 'Carnivore'},`
`{"name": "Puma", "quantity": 12, 'diet': 'Carnivore'},`
`{"name": "Meerkat", "quantity": 15, 'diet': 'Omnivore'},`
`{"name": "Elephant", "quantity": 7, 'diet': 'Herbivore'},`
`{"name": "Zebra", "quantity": 6, 'diet': 'Herbivore'}`
`]`
 	
`df = pd.DataFrame(d)`
`df = df.groupby(by=['diet']).sum()`
`df.reset_index()`	

Name: Date:

Lesson 2.1

Task 6: Selecting row

Code	Output

`# Grouping Data`
`d = [`
`{"name": "Lion", "quantity": 3, 'diet': 'Carnivore'},`
`{"name": "Puma", "quantity": 12, 'diet': 'Carnivore'},`
`{"name": "Meerkat", "quantity": 15, 'diet': 'Omnivore'},`
`{"name": "Elephant", "quantity": 7, 'diet': 'Herbivore'},`
`{"name": "Zebra", "quantity": 6, 'diet': 'Herbivore'}`
`]`
 	
`df = pd.DataFrame(d)`
`df = df[df['name'] == 'Zebra']`
`df`	

Name: Date:

Lesson 2.1

Task 7: Updating the row

Code

```
# Grouping Data
d = [
    {"name": "Lion", "quantity": 3, 'diet':
'Carnivore'},
    {"name": "Puma", "quantity": 12, 'diet':
'Carnivore'},
    {"name": "Meerkat", "quantity": 15, 'diet':
'Omnivore'},
    {"name": "Elephant", "quantity": 7, 'diet':
'Herbivore'},
    {"name": "Zebra", "quantity": 6, 'diet':
'Herbivore'}
]

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

Output

Task 8: Updating the row

Code

```
d = [
    {"name": "Lion", "quantity": 3},
    {"name": "Puma", "quantity": 12},
    {"name": "Meerkat", "quantity": 15},
]

df = pd.DataFrame(d)
df['females'] = df['quantity']
df['females'] *= 0.5 # assuming 50% of population are
females
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	Day 6	Day 7
Hello Panda	Snack	20	21	16	15	18	12	18
Waffle	Snack	10	13	9	20	2	23	2
Coke	Drink	15	11	19	21	22	26	12
Sprite	Drink	12	23	14	6	13	16	27
Fanta	Drink	19	20	5	17	16	17	16

Write a Python Code using Pandas Library to do the following

1. Load the data into a DataFrame
2. Select the row containing the 'Waffle' item
3. Within the 'Waffle' item, select 'Day 4'
4. Insert a new column 'Sales', containing the sum of all 7 Days
5. Group the types & see the 'Sales' by type
6. Calculate the Earnings from the 'Sales', assuming that
    a. Drink cost $1.50 per sale
    b. Snack cost $3.00 per sale
7. Insert a new column 'Max', containing the highest among all 7 Days
8. Insert a new column 'Min', containing the lowest among all 7 Days

Guided solution found in the next few pages

Name: Date:

Lesson 2.1

Task 1: Load the data into a DataFrame

```
store =  [
  {
    'Item': 'Hello Panda',
    'Type':  'Snack',
    'Day 1': 20,
    'Day 2': 21,
    'Day 3': 16,
    'Day 4': 15,
    'Day 5': 18,
    'Day 6': 12,
    'Day 7': 17
  }
]

import pandas as pd

df = pd.DataFrame(store)
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'] == 'Waffle')]
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'] == 'Waffle' --- creates a filter that is applied to df_2 to only select the row

Name: Date:

Lesson 2.1

Task 3: Within the 'Waffle' item, select 'Day 4'

```
df_3 = df.copy()
df_3 = df_3[(df_3['Item'] == 'Waffle')]
df_3 = df_3[['Item','Day 4']]
df_3
```

df_3[(df_3['Item'] == 'Waffle')] will contain the row.
To select the specific columns, df_3[['Item','Day 4']] is used

Task 4:

```
df_4 = df.copy()
df_4['Sales'] = df_4.sum(axis = 1)
df_4 = df_4[['Item','Type', 'Sales']]
df_4
```

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

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'] == 'Drink', ['Earnings']] *= 1.5
df_6.loc[df_6['Type'] == 'Snack', ['Earnings']] *= 3
df_6
```

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

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

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

Name: Date:

Lesson 2.1

Task 7: Insert new column 'Max'

```
df_7 = df.copy()
df_7['Max'] = df_7.max(axis = 1)
df_7 = df_7[['Item','Type', 'Max']]
df_7
```

df_7.min(axis = 1) gets the minimum value across the row

Task 8: Insert new column 'Min'

```
df_8 = df.copy()
df_8['Min'] = df_8.max(axis = 1)
df_8 = df_8[['Item','Type', 'Min']]
df_8
```

df_8.min(axis = 1) gets the minimum value across the row

Name: Date:

Lesson 2.2

2.2 Learning – Understanding Pandas DateTime Object

Sales	Month	Day	Year
10	1	1	2022
12	1	2	2022
23	1	3	2022
4	1	4	2022
9	1	5	2022
12	1	6	2022
42	1	7	2022
23	1	8	2022
6	1	9	2022
9	1	10	2022

Task 1: Loading data into DataFrame

Code

```
dates = [
    {'Sales': 10, 'Month': 1, 'Day': 1, 'Year': 2022},
    {'Sales': 12,  'Month': 1, 'Day': 2, 'Year': 2022},
    {'Sales': 23,  'Month': 1, 'Day': 3, 'Year': 2022},
    {'Sales': 4,  'Month': 1, 'Day': 4, 'Year': 2022},
    {'Sales': 9,  'Month': 1, 'Day': 5, 'Year': 2022},
    {'Sales': 12, 'Month': 1, 'Day': 6, 'Year': 2022},
    {'Sales': 42,  'Month': 1, 'Day': 7, 'Year': 2022},
    {'Sales': 23,  'Month': 1, 'Day': 8, 'Year': 2022},
    {'Sales': 6,  'Month': 1, 'Day': 9, 'Year': 2022},
    {'Sales': 9,  'Month': 1, 'Day': 10, 'Year': 2022}]

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

Output

As per usual. You may choose to store data in a CSV format.

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.

Task 6: Understanding the DataFrame

Code	Output
`df.info()`	

What is the Dtype of the Columns now?

Name: Date:

Lesson 2.2

2.2 Practice – Understanding Pandas DateTime Object

Exercise

Sales	Month	Day
10	2	11
5	2	12
3	2	13
7	2	14
2	2	15

Write a Python code to

1. *Insert a new column 'Year' with values 2022*
2. *Create a new column 'Date' that is a Pandas DateTime object*
3. *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': 2, 'Day': 11},
    {'Sales': 5, 'Month': 2, 'Day': 12},
    {'Sales': 3, 'Month': 2, 'Day': 13},
    {'Sales': 7, 'Month': 2, 'Day': 14},
    {'Sales': 2, 'Month': 2, 'Day': 15}
]

import pandas as pd

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

Add a new key-value pair into the dictionary entries

Step 2: Add in Year 2022

```
df['Year'] = 2022
```

Step 3: 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 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

Step 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.

Step 6: Filter

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	Day 6	Day 7
Hello Panda	Snack	20	21	16	15	18	12	18
Waffle	Snack	10	13	9	20	2	23	2
Coke	Drink	15	11	19	21	22	26	12
Sprite	Drink	12	23	14	6	13	16	27
Fanta	Drink	19	20	5	17	16	17	16

In this example, we want to create a plot for the Waffle throughout the 7 days – Bar Graph & Line Graph

Name: Date:

Lesson 2.3

Task 1: Loading in DataFrame for **only Waffle**

Code	Output

`store = {`
`'Day': [1,2,3,4,5,6,7],`
`'Waffle': [10,13,9,20,2,23,2]`
`}`
`import pandas as pd`
 	
`df = pd.DataFrame(store)`
`df`	

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

Task 2:

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

Task 3:

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

Name: Date:

Lesson 2.3

Task 4:

Code	Output
`w = 0.2` `x = np.asarray(df['Day'])`	

Task 5:

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

Task 6:

Code	Output

`ax1.set_title("Waffle Sales")`
`ax1.set_xlabel("Days")`
`ax1.set_ylabel("Sales")`	

Name: Date:

Lesson 2.3

Task 7:

Code	Output
`ax1.set_xticks(df['Day'])`	

Task 8:

Code	Output
`plt.show()`	

Full:

Code	Output

`import matplotlib.pyplot as plt`
`import numpy as np`
 	
`fig1,ax1= plt.subplots(1,1)`
`w = 0.2`
`x = np.asarray(df['Day'])`
 	
`ax1.bar(x, df['Waffle'], width = w)`
 	
`ax1.set_title("Waffle Sales")`
`ax1.set_xlabel("Days")`
`ax1.set_ylabel("Sales")`
 	
`ax1.set_xticks(df['Day'])`
 	
`plt.show()`	

Name: Date:

Lesson 2.3

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

Task 1: Loading in DataFrame for **only Waffle**

Code	Output

`store = {`
`'Day': [1,2,3,4,5,6,7],`
`'Waffle': [10,13,9,20,2,23,2]`
`}`
`import pandas as pd`
 	
`df = pd.DataFrame(store)`
`df`	

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

Task 2:

Code	Output
`import matplotlib.pyplot as plt`	

Numpy is not necessary here

Task 3:

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

Name: Date:

Lesson 2.3

Task 5:

Code	Output
`ax1.plot(df['Day'], df['Waffle'])`	

Task 6:

Code	Output

`ax1.set_title("Waffle Sales")`
`ax1.set_xlabel("Days")`
`ax1.set_ylabel("Sales")`	

Task 7:

Code	Output
`ax1.set_xticks(df['Day'])`	

Task 8:

Code	Output
`plt.show()`	

Name: Date:

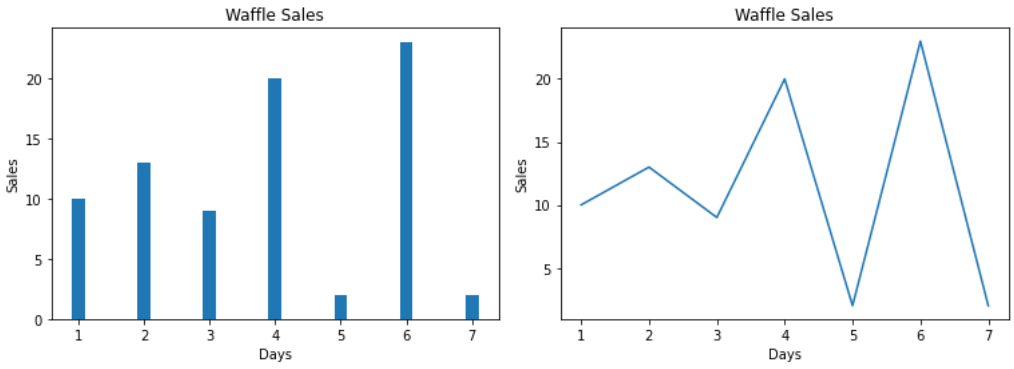
Lesson 2.3

Full:

Code	Output

```
import matplotlib.pyplot as plt

fig1,ax1= plt.subplots(1,1)

ax1.plot(df['Day'], df['Waffle'])

ax1.set_title("Waffle Sales")
ax1.set_xlabel("Days")
ax1.set_ylabel("Sales")

ax1.set_xticks(df['Day'])

plt.show()
```

Expected Graphs

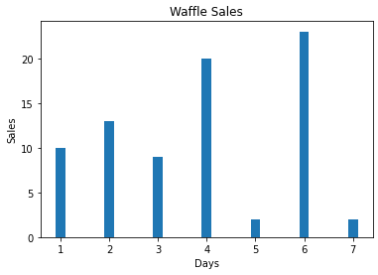
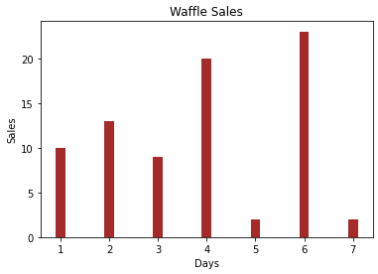
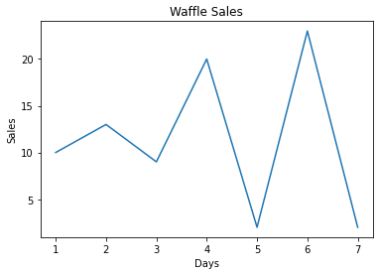


Name: Date:

Lesson 2.3

Decorating the graphs – uses examples from the previous section

Setting Colour

`ax1.bar(x, df['Waffle'], width = w)`	
`ax1.bar(x, df['Waffle'], width = w,` **`color='Brown'`**`)`	
`ax1.plot(df['Day'], df['Waffle'])`	
`ax1.plot(df['Day'], df['Waffle'],` `color="Brown")`	

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(df['Day'], df['Waffle'], color="Brown", marker='H')`	
`ax1.plot(df['Day'], df['Waffle'], color="Brown", 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	Day 6	Day 7
Hello Panda	Snack	20	21	16	15	18	12	18
Waffle	Snack	10	13	9	20	2	23	2
Coke	Drink	15	11	19	21	22	26	12
Sprite	Drink	12	23	14	6	13	16	27
Fanta	Drink	19	20	5	17	16	17	16

Write a Python Code to

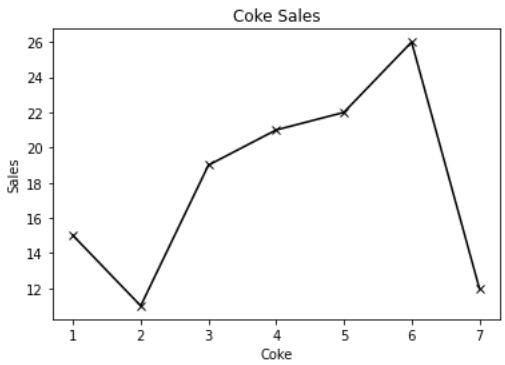
1. Create a line plot for **Coke –** Black in Color, with x as Marker
2. Create a line plot for **Sprite –** Green in Color, with '--' as Linestyle
3. Create a bar plot for **Fanta** – Orange in Color

Name: Date:

Lesson 2.3

Task 1:

```
store =  {
    'Day': [1,2,3,4,5,6,7],
    'Coke': [15,11,19,21,22,26,12],
    'Sprite': [12,23,14,6,13,16,27],
    'Fanta': [19,20,5,17,16,17,16]
  }

import pandas as pd
df = pd.DataFrame(store)

import matplotlib.pyplot as plt

fig1,ax1= plt.subplots(1,1)

ax1.plot(df['Day'], df['Coke'], color="Black", marker='x')

ax1.set_title("Coke Sales")
ax1.set_xlabel("Coke")
ax1.set_ylabel("Sales")

ax1.set_xticks(df['Day'])

plt.show()
```

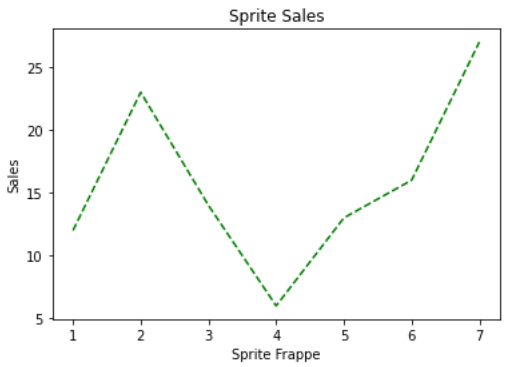


Name: Date:

Lesson 2.3

Task 2:

```
store =  {
    'Day': [1,2,3,4,5,6,7],
    'Coke': [15,11,19,21,22,26,12],
    'Sprite': [12,23,14,6,13,16,27],
    'Fanta': [19,20,5,17,16,17,16]
  }

import pandas as pd
df = pd.DataFrame(store)

import matplotlib.pyplot as plt

fig1,ax1= plt.subplots(1,1)

ax1.plot(df['Day'], df['Sprite'], color="Green", linestyle='--')

ax1.set_title("Sprite Sales")
ax1.set_xlabel("Sprite Frappe")
ax1.set_ylabel("Sales")

ax1.set_xticks(df['Day'])

plt.show()
```

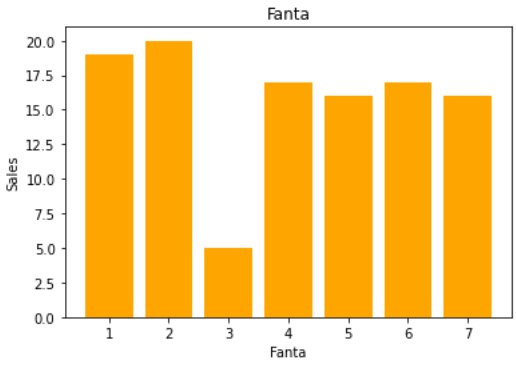


Name: Date:

Lesson 2.3

Task 3:

```
store =  {
    'Day': [1,2,3,4,5,6,7],
    'Coke': [15,11,19,21,22,26,12],
    'Sprite': [12,23,14,6,13,16,27],
    'Fanta': [19,20,5,17,16,17,16]
  }

import pandas as pd
df = pd.DataFrame(store)

import matplotlib.pyplot as plt

fig1,ax1= plt.subplots(1,1)

ax1.bar(df['Day'], df['Fanta'], color="Orange")

ax1.set_title("Fanta")
ax1.set_xlabel("Fanta")
ax1.set_ylabel("Sales")

ax1.set_xticks(df['Day'])

plt.show()
```



Name: Date:

Day 3

	Lesson Plan
3	- Exploration of data analytics - Bar Graph, Line Graph - 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/SeoulBikeData.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: Understanding the DataFrame

```
df.info()
```

Since it is a new DataFrame, we can always see the various columns and Dtype using info()

Name: Date:

Lesson 3.1

Let get information to get the total number of Rented Bike Count on the 3 days.

Date	Rented Bike Count
2017/01/12	
2017/02/12	
2017/03/12	

Task 3: Filter out the Columns

```
df_1 = df_1[['Date', 'Hour', 'Rented Bike Count']]
df_1
```

You can filter out the columns in the DataFrame.

Task 4: DateTime Object

```
import pandas as pd
df_1['Date'] = pd.to_datetime(df_1['Date'])
df_1
```

Since we are working with time series date, we should convert the Date column to a DateTime Object

Name: Date:

Lesson 3.1

Task 5: Select specific Dates

```
df_1 = df_1[ (df_1['Date'] == '2017-01-12') | (df_1['Date'] == '2017-02-12' ) |
 (df_1['Date'] == '2017-03-12' )]
df_1
```

| means OR so helps to combine the different logics

(df_1['Date'] == '2017-01-12') | (df_1['Date'] == '2017-02-12') looks for any entries that has a Date Value of 2017-01-12 and 2017-02-12

Task 6: Grouping

```
df_1 = df_1.groupby(by='Date').sum()
df_1 = df_1[['Rented Bike Count']]
df_1
```

Eventually you will get the table below in df_1

Date	Rented Bike Count
2017/01/12	9539
2017/02/12	8523
2017/03/12	7222

Name: Date:

Lesson 3.1

3.1 Practice – Handling Online Data with Pandas

Date	Rented Bike Count	Temperature	Humidity
2017/01/12	9539		
2017/02/12	8523		
2017/03/12	7222		
2017/04/12			
2017/05/12			
2017/06/12			
2017/07/12			

Using Pandas Library, find the values for **max** Temperature, average Humidity for the respective dates

Solution found in the next few pages

Name: Date:

Lesson 3.1

For Rented Bike Count

Task 1: Import online dataset

```
import pandas as pd

url =" https://raw.githubusercontent.com/tlc-
datascience/datasets/main/SeoulBikeData.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_1 = df_1[['Date', 'Hour', 'Rented Bike Count']]
df_1
```

You can filter out the columns in the DataFrame.

Task 3: DateTime Object

```
import pandas as pd
df_1['Date'] = pd.to_datetime(df_1['Date'])
df_1
```

Since we are working with time series date, we should convert the Date column to a DateTime Object

Name: Date:

Lesson 3.1

Task 4: Select specific Dates

```
df_1 = df_1[ (df_1['Date'] == '2017-01-12') | (df_1['Date'] == '2017-02-12' ) |
 (df_1['Date'] == '2017-03-12' ) | (df_1['Date'] == '2017-04-12') | (df_1['Date']
== '2017-05-12' ) | (df_1['Date'] == '2017-06-12' )]
df_1
```

| means OR so helps to combine the different logics

(df_1['Date'] == '2017-01-12') | (df_1['Date'] == '2017-02-12') looks for any entries that has a Date Value of 2017-01-12 and 2017-02-12

Task 5: Grouping

```
df_1 = df_1.groupby(by='Date').sum()
df_1 = df_1[['Rented Bike Count']]
df_1
```

Name: Date:

Lesson 3.1

For Temperature & Humidity

Task 1: Import online dataset

```
import pandas as pd

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

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

Task 2: Filter out the Columns

```
df_2 = df.copy()
df_2 = df_2[['Date','Hour', 'Temperature(C)', 'Humidity ']]
df_2
```

Copy is used to make sure that we do not modify the original DataFrame

Task 3: Pandas DateTime

```
import pandas as pd
df_2['Date'] = pd.to_datetime(df_2['Date'])
```

Convert Date Column to DateTime to filter better

Name: Date:

Lesson 3.1

Task 4: Filter Dates

```
df_2 = df_2[ (df_2['Date'] == '2017-01-12') | (df_2['Date'] == '2017-02-12' ) |
 (df_2['Date'] == '2017-03-12' ) | (df_2['Date'] == '2017-04-12') | (df_2['Date']
== '2017-05-12' ) |
 (df_2['Date'] == '2017-06-12' )]
df_2
```

| means OR so helps to combine the different logics

(df_1['Date'] == '2017-01-12') | (df_1['Date'] == '2017-02-12') looks for any entries that has a Date Value of 2017-01-12 and 2017-02-12

Task 5: Groupby Dates & get the max values

```
df_2 = round(df_2.groupby("Date").max())
df_2
```

Task 6: Display

```
df_2 = df_2[['Temperature(C)', 'Humidity ']]
df_2
```

Name: Date:

Lesson 3.2

3.2 Learning – Bar Graph, Line Graph

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

1. Create a line plot for **Rented Bike Count, Hourly on 2017-01-12 –** Red in Color, with 'o' as Marker, 'dotted' as Linestyle
2. Create a bar plot for **Rented Bike Count, Hourly on 2017-01-12 –** Blue in Color

Task: Getting the data

```
import pandas as pd
url = "https://raw.githubusercontent.com/tlc-
datascience/datasets/main/SeoulBikeData.csv"
df = pd.read_csv(url)

df = df[['Date','Hour','Rented Bike Count']]
df['Date'] = pd.to_datetime(df['Date'])
df = df[(df['Date'] == '2017-01-12')]
df
```

Name: Date:

Lesson 3.2

Task 1: Create a line plot for **Rented Bike Count, Hourly on 2017-01-12**

Step 1: Import the relevant libraries

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

Step 2: Create the figure and axes

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

Step 3: Plot the line graphs

```
ax1.plot(df['Hour'], df['Rented Bike Count'], color="Red", marker='o',
linestyle='dotted')
```

This is where you do the styling of the graph

Step 4: Set the title, x-label & y-label

```
ax1.set_title("2017-01-12")
ax1.set_xlabel("Hour")
ax1.set_ylabel("Rented Bike Count")
```

Step 5: Display the plot

```
plt.show()
```

Name: Date:

Lesson 3.2

Task 2: Create a bar plot for **Rented Bike Count, Hourly on 2017-01-12**

Step 1: Import the relevant libraries

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

Step 2: Create the figure and axes

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

Step 3: Plot the bar graphs

```
w = 0.2
x = np.asarray(df['Hour'])
```

Step 4: Plot the line graphs

```
ax1.bar(df['Hour'], df['Rented Bike Count'], color="Blue")
```

This is where you do the styling of the graph

Name: Date:

Lesson 3.2

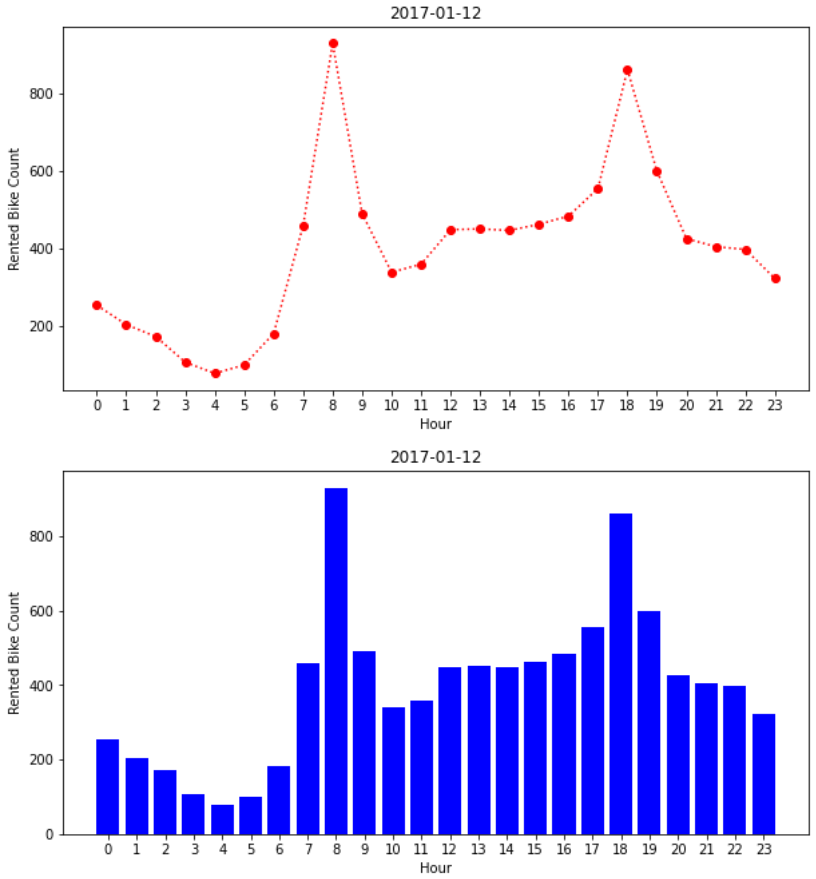
Step 5: Set the title, x-label & y-label

```
ax1.set_title("2017-01-12")
ax1.set_xlabel("Hour")
ax1.set_ylabel("Rented Bike Count")
```

Step 6: Display the plot

```
plt.show()
```

Expected Graphs



Name: Date:

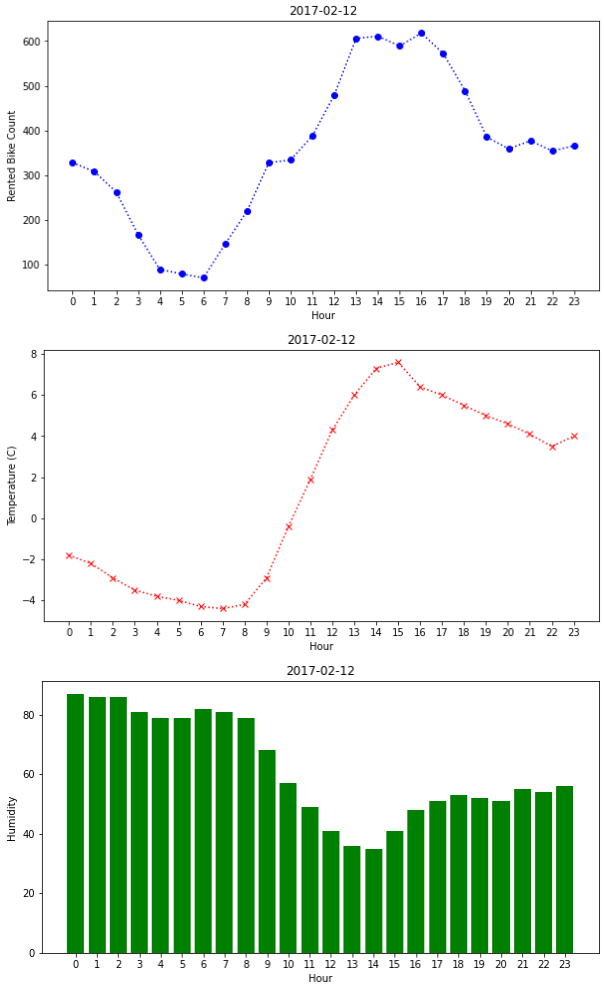
Lesson 3.2

3.2 Practice – Bar Graph, Line Graph

Write a Python Code using Matplotlib to

1. Create a line plot for **Rented Bike Count, Hourly on 2017-02-12 –** Blue in Color, with 'o' as Marker, 'dotted' as Linestyle
2. Create a line plot for **Temperature, Hourly on 2017-02-12 –** Red in Color, with 'x' as Marker, 'dotted' as Linestyle
3. Create a bar plot for **Humidity, Hourly on 2017-02-12 –** Green in Color

Expected Graphs



No Guided Solution Provided

Name: Date:

Lesson 3.3

3.3 Learning – Multiplot

In this example, we will write a Python Code using Matplotlib to create a multiplot for 'Rented Bike Count', 'Temperature', 'Humidity' for 2017-01-12.

Task: Load in DataFrame

```
import pandas as pd
url = "https://raw.githubusercontent.com/tlc-
datascience/datasets/main/SeoulBikeData.csv"
df = pd.read_csv(url)

df = df[['Date','Hour','Rented Bike Count', 'Temperature(C)', 'Humidity ']]
df['Date'] = pd.to_datetime(df['Date'])
df = df[(df['Date'] == '2017-01-12')]
df
```

Name: Date:

Lesson 3.3

Line graph

Task 2: Import the libraries

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

Task 3: Create the figure and axes

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

Task 4: Plot the line graph

```
ax1.plot(df['Hour'], df['Rented Bike Count'], label='Rented Bike Count')
ax1.plot(df['Hour'], df['Temperature(C)'], label='Temperature')
ax1.plot(df['Hour'], df['Humidity '], label='Humidity')
```

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

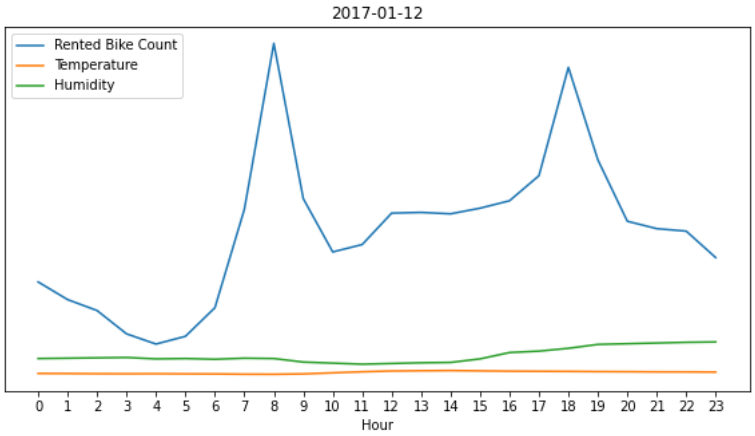
```
ax1.set_title("2017-01-12")
ax1.set_xlabel("Hour")
# ax1.set_ylabel("Temperature")
```

Task 6: Set x ticks & clear y ticks

```
ax1.set_xticks(df['Hour'])
ax1.set_yticks([])
```

Task 7: Display the plot & legend

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



Name: Date:

Lesson 3.3

Bar graph

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))
```

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

```
w = 0.2
x = np.asarray(df['Hour'])
```

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, df['Rented Bike Count'], width = w, label='Rented Bike Count')
ax1.bar(x, df['Temperature(C)'], width = w, label='Temperature')
ax1.bar(x+w, df['Humidity '], width = w, label='Humidity')
```

x-w & x+w 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 and -w helps to shift the bar to the left or right of each other

Name: Date:

Lesson 3.3

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

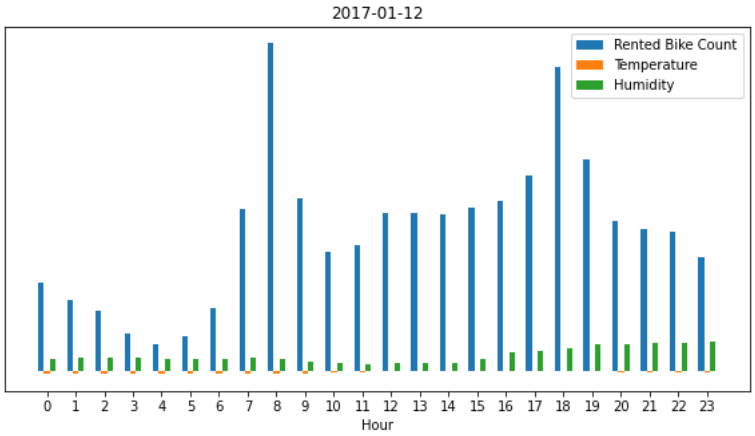
```
ax1.set_title("2017-01-12")
ax1.set_xlabel("Hour")
# ax1.set_ylabel("Temperature")
```

Task 7: Set x-ticks & clear y-ticks

```
ax1.set_xticks(df['Hour'])
ax1.set_yticks([])
```

Task 8: Display the plot

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



Name: Date:

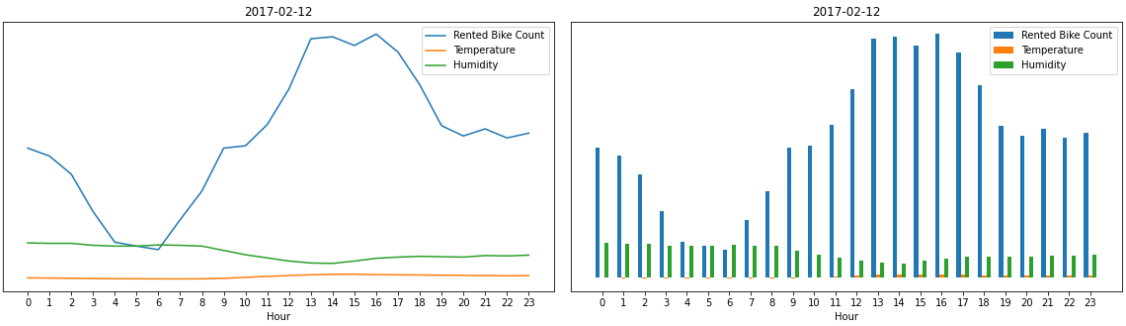
Lesson 3.3

3.3 Practice – Multiplot

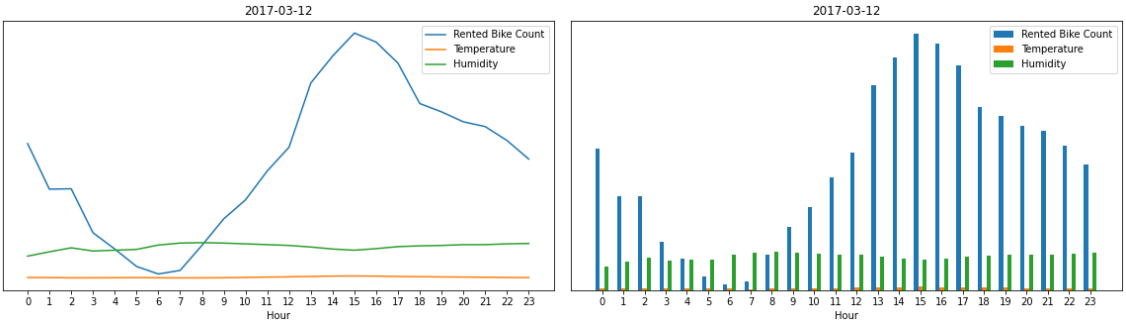
In this example, we will write a Python Code using Matplotlib to create a multiplot for 'Rented Bike Count', 'Temperature', 'Humidity' for 2017-02-12 & 2017-03-12 for both line & bar graph.

Expected Graph

2017-02-12



2017-03-12



No Guided Solution Provided

Name: Date:

Day 4

	Lesson Plan
4	- Data Analysis: Graduate Employment Survey - NTU, NUS, SIT, SMU, SUSS & SUTD

**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

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

Exercise 1:

Objective- To see which year was the highest/lowest rate of employment for NTU Bioengineering.

Plot a line graph to show the trend of employment rate ("employment_rate_overall") from 2013 to 2019 for the Nanyang Technological University, College of Engineering – Bioengineering.

https://data.gov.sg/dataset/graduate-employment-survey-ntu-nus-sit-smu-suss-sutd

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 graphs do we use?

Name: Date:

Lesson 4

Recall your Step 1 to Step 9
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	

Step 3:
#Remove NA and "-"for value columns

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

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	

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

1 line graph of year (x-axis) vs employment rate (y-axis) for Nanyang Technological University, College of Engineering – Bioengineering.

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

We have
a. University - Nanyang Technological University
b. School - College of Engineering
c. Degree - Bioengineering
So What do we need to do?
Filter and Extract to get to our goal for the above and for each of the year 2008 till 2020

Step 5: Filter and extract the year

5	df_newname = df_oldname[(df_oldname["columnname"] == criteria & (df_oldname["columnname"]==criteria)]
Ans	`df_copy = df_copy[ (df_copy['year'] >= 2013) & (df_copy['year'] <= 2019)]` `df_copy = df_copy[(df_copy['university'] == 'Nanyang Technological University') & (df_copy['school'] == 'College of Engineering') &(df_copy['degree'] == 'Bioengineering']`

Name: Date:

Lesson 4

Step 6: Sum the sub-categories of the industries – Applicable?

6	df_newname=df_newname.groupby([“columnname”]).sum()
Ans	

Step 7: Remove duplicates

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

Step 8: Plot line graph

9	fig1.ax1=plt.subplot(1,1) ax1.plot(df.”columnnmane” , df.”columnname”, label= “labelname”)
Ans	

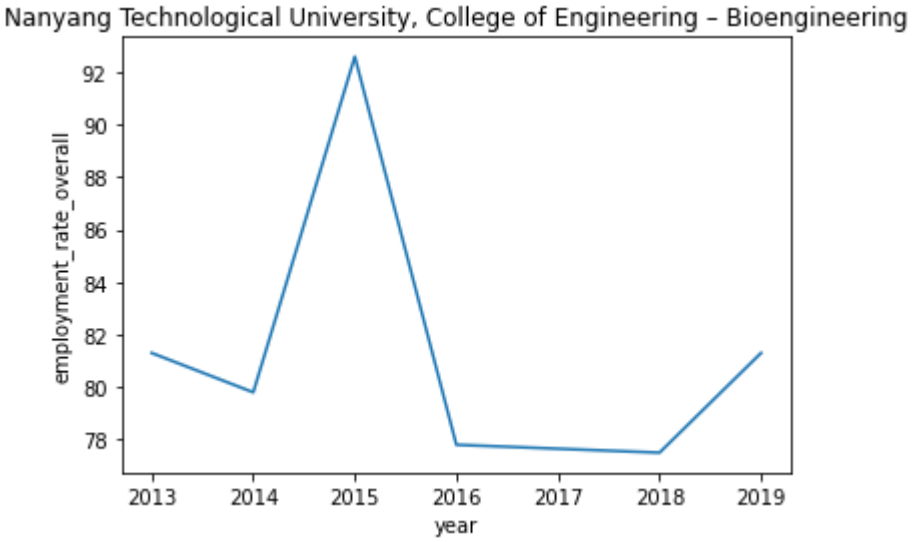
Step 9: Display plot with labels

8	ax1.set_xlabel(“columnname”) ax1.set_ylabel(“columnname”) ax1.set_title(“titlename”) plt.show()
Ans	

Name: Date:

Lesson 4

Output:



Show the trend of employment rate ("employment_rate_overall") from 2013 to 2019 for the Nanyang Technological University, College of Engineering – Bioengineering.

Conclusion: Which year was the highest rate of employment? Which year was the lowest rate of employment? Why period had the greatest increase in employment? Which period had the greatest decrease in employment?

Name: Date:

Lesson 4

Exercise 2: *(Similar to previous exercise, to be done independently)*

Objective- To see which year was the highest/lowest rate of employment for NUS – computing.

Plot a line graph to show the trend of employment rate ("employment_rate_overall") from 2013 to 2019 for the National University of Singapore, School of Computing - Bachelor of Computing (Information Systems)

Write your code here – Recap steps 1 to 9

Conclusion:

Name: Date:

Lesson 4

Exercise 3:

Objective- To compare the employment rate between the same course in 2 different university over the years

Plot 2 line graphs in 1 plot to show the trend of employment rate ("employment_rate_overall") from 2013 to 2019 to compare between the 2 courses

Singapore Management University, School of Accountancy, (4-years programme) *

Nanyang Technological University, College of Business, (Nanyang Business School)

Recall your Step 1 to Step 9 – Do it for Singapore Management University, School of Accountancy, (4-years programme) * first, then repeat steps 1 to 9 for Nanyang Technological University, College of Business, (Nanyang Business School)

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	

Step 3:
#Remove NA and "-"for value columns

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

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	

*Step 5a: Create a bag of labels
Note: As there are many different types of degree names of in each school, we need to make sure as we filter, we include all, so we need to create a "bag" to "put" these different degree names inside.

5	df= df[df['columnname'].isin(bags_1)]

Ans	`bags_1 = [`
	`'Accountancy (4-years programme) Cum Laude and above',`
	`'Accountancy (4-year programme) Cum Laude and above',`
	`'Accountancy Cum Laude and above',`
	`'Accountancy (Cum Laude and above)'`
	`]`

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

Plot line graph in 1 plot to show the trend of employment rate ("employment_rate_overall") from 2013 to 2019 to compare between the 2 courses

Singapore Management University, School of Accountancy, (4-years programme) *

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

We have
a. University – Singapore Management University
b. School - School of Accountancy
c. Degree – "In the bag" above
So What do we need to do?
Filter and Extract to get to our goal for the above and for each of the year 2013 till 2019

Name: Date:

Lesson 4

Step 5b: Filter and extract the year

5	df_newname = df_oldname[(df_oldname[“columnname”] == criteria & (df_oldname[“columnname”]==criteria)]

Ans	`df_copy_1 = df_copy.copy()`
	`df_copy_1 = df_copy_1[ (df_copy_1['year'] >= 2013) & (df_copy_1['year'] <= 2019) ]`
	`df_copy_1 = df_copy_1[(df_copy_1['university'] == 'Singapore Management University')]`
	`df_copy_1 = df_copy_1[df_copy_1['degree'].isin(bags_1)]`

Step 6: Sum the sub-categories of the industries – Applicable?

6	df_newname=df_newname.groupby([“columnname”]).sum()
Ans	

Step 7: Remove duplicates

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

Step 8: Plot line graph

9	fig1.ax1=plt.subplots(1,1) ax1.plot(df.”columnnmane” , df.”columnname”, label= “labelname”) plt.legend()
Ans	

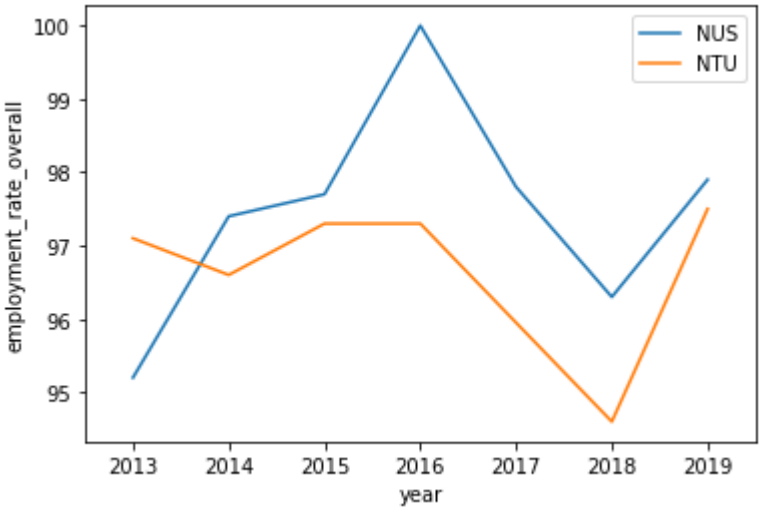
Name: Date:

Lesson 4

Step 9: Display plot with labels

8	ax1.set_xlabel("columnname") ax1.set_ylabel("columnname") ax1.set_title("titlename") plt.show()
Ans	

Output:



Show the trend of employment rate ("employment_rate_overall") from 2013 to 2019 to compare between the 2 courses

Singapore Management University, School of Accountancy, (4-years programme) *

Nanyang Technological University, College of Business, (Nanyang Business School)

Conclusion: How are the trends for employment rate different between the 2 schools? Are they in tandem or in opposites?

Name: Date:

Lesson 4

Exercise 4: *(Similar to previous exercise, to be done independently)*

Objective- To compare the remuneration between the same course in 2 different university over the years

Plot 2 bar graphs in 1 plot to show the trend of remuneration ("basic_monthly_mean") from 2013 to 2019 to compare between the 2 courses

Singapore Management University, School of Law (4-years programme) * - Law (4-years programme) #

National University of Singapore, Faculty of Law - Bachelor of Laws (L.L.B)

Write your code here – Recap steps 1 to 9

Recap step 6:

```
df_newname=df_newname.groupby(["columnname"]).sum()
```

To find the mean:

```
df_newname=round(df_newname.groupby(["columnname"]).mean(),2)
```

Conclusion:

Name: Date:

Lesson 4

Exercise 5:

Objective- To compare which college of study in NTU has the highest/lowest average average remuneration ("basic_monthly_mean")

Plot 1 vertical bar graph in 1 plot to show the difference in average remuneration ("basic_monthly_mean") of the different schools in the Nanyang Technological University for the year 2019

Task 1b:

Recall your Step 1 to Step 9

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	

Step 3:
#Remove NA and "-"for value columns

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

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	

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

Plot 1 bar graph in 1 plot to show the difference in average remuneration ("basic_monthly_mean") of the different schools in the Nanyang Technological University for the year 2019

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

We have the different schools:
College of Business (Nanyang Business School
College of Engineering
College of Humanities, Arts & Social Sciences
College of Sciences
National Institute of Education (NIE)
Sports Science and Management
Lee Kong Chian School of Medicine
Filter and Extract to get to our goal for the above and for each of the year 2019

Step 5b: Filter and extract the year

5	df_newname = df_oldname[(df_oldname["columnname"] == criteria & (df_oldname["columnname"]==criteria)]
Ans	`df_copy = df_copy[df_copy['university'] == 'Nanyang Technological University']` `df_copy = df_copy[df_copy['year'] == '2019']`

Name: Date:

Lesson 4

Step 6: Sum the sub-categories of the industries – Applicable? Or find the mean?

6	df_newname=df_newname.groupby([“columnname”]).sum()
6	df_newname=round(df_newname.groupby([“columnname”]).mean(),2)
Ans	

Step 7: Remove duplicates

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

Step 8: Plot vertical bar graph

9	fig1,ax1=plt.subplots(1,1) ax1.bar(df.”columnnmane” , df.”columnname”, label= “labelname”) plt.legend()
Ans	

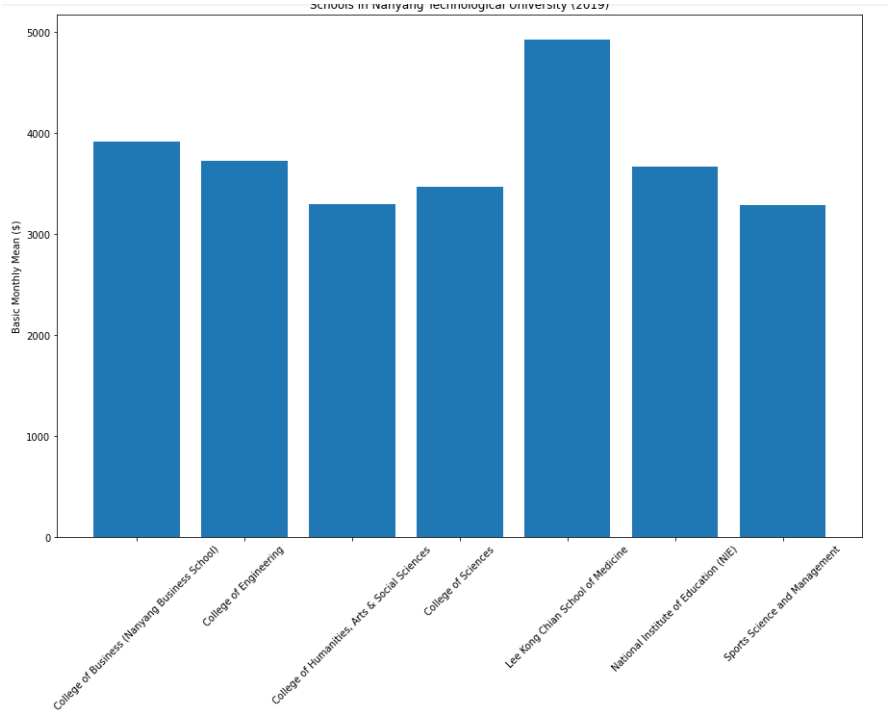
Step 9: Display plot with labels

8	ax1.set_xlabel(“columnname”) ax1.set_ylabel(“columnname”) ax1.set_title(“titlename”) plt.show()
Ans	

Name: Date:

Lesson 4

Output:



Plot 1 vertical bar graph in 1 plot to show the difference in average remuneration ("basic_monthly_mean") of the different schools in the Nanyang Technological University for the year 2019

Conclusion: Which school has the highest basic monthly mean in 2019? Which school has the lowest basic monthly mean in 2019?

Name: Date:

Lesson 4

Exercise 6: *(Similar to previous exercise, to be done independently)*

Objective- To compare which college of study in NUS has the highest/lowest average average remuneration ("basic_monthly_mean")

Plot 1 horizontal bar graph in 1 plot to show the difference in average remuneration ("basic_monthly_mean") of the different schools in the National University of Singapore for the year 2019

Write your code here – Recap steps 1 to 9

Conclusion:

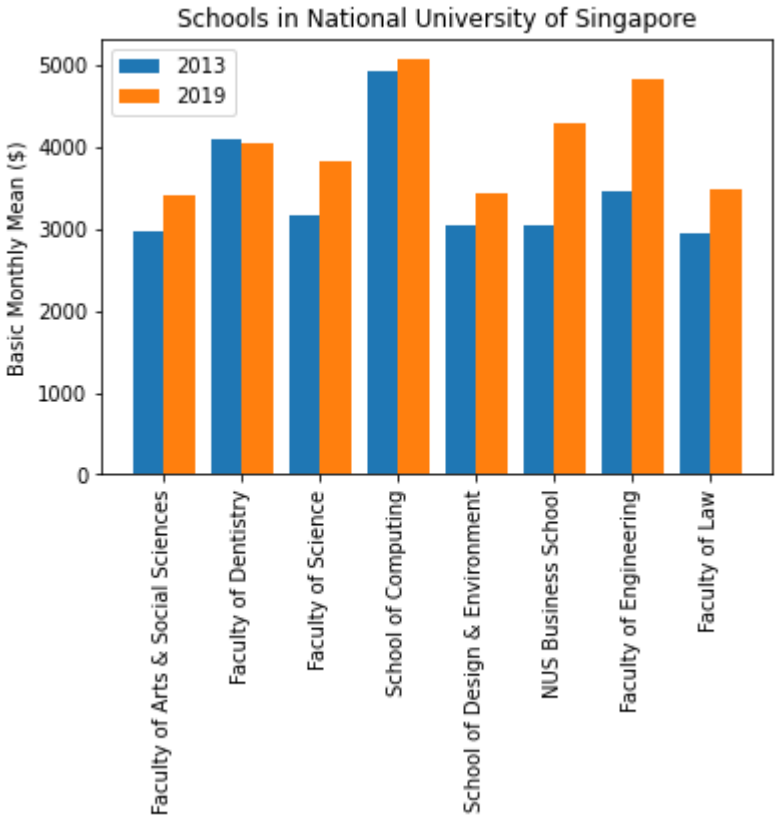
Name: Date:

Lesson 4

Exercise 7: *(For the extra challenge)*

Objective- To compare average remuneration ("basic_monthly_mean") between 2 years 2013 and 2019 for all college of study in NUS

Plot 2 bar graph side by side in 1 plot to show the difference in average remuneration ("basic_monthly_mean") between 2013 and 2019 of the different schools in the National University of Singapore



Name: Date:

Lesson 4

```
### Step 1
df_copy = df.copy()

### Step 2
df_copy = df_copy[df_copy['university'] == 'National University of Singapore']

### Step 3
df_copy = df_copy[df_copy['basic_monthly_mean'] != 'na']

### Step 4
df_copy['basic_monthly_mean'] = pd.to_numeric(df_copy['basic_monthly_mean'])

### Step 5
df_copy_2013 = df_copy[df_copy['year'] == '2013'].copy()
df_copy_2013 = df_copy_2013[['school', 'basic_monthly_mean']]
df_copy_2019 = df_copy[df_copy['year'] == '2019'].copy()
df_copy_2019 = df_copy_2019[['school', 'basic_monthly_mean']]

### Step 6 - Find Intersection
list1 = df_copy_2013['school'].unique()
list2 = df_copy_2019['school'].unique()
bag = list(set(list1).intersection(list2))

### Step 7
df_copy_2013 = df_copy_2013[df_copy_2013['school'].isin(bag)]
df_copy_2013 = round(df_copy_2013.groupby(['school']).mean(),2)

df_copy_2019 = df_copy_2019[df_copy_2019['school'].isin(bag)]
df_copy_2019 = round(df_copy_2019.groupby(['school']).mean(),2)
```