Assignment-1

Notes from videos

Xiaoyi Cui

Positron

Getting Started with Positron: A Quick Tour

Folder from Git

`File-New Folder from Git`-enter the http (e.g.: https://github.com/posit-dev/posit-conf-2025-positron-assistant-demo.git)

Underlined packages are not installed

Install Packages

1. In the terminal, enter `uv venv` to create virtual environment
2. Enter `.venv\Scripts\activate`
3. e.g. `uv pip install -r requirements.txt` or `uv pip install jupyter`

For Help `help()`

e.g.: In the console, enter `help(p9.theme_minimal)`

Your First Python Project in Positron

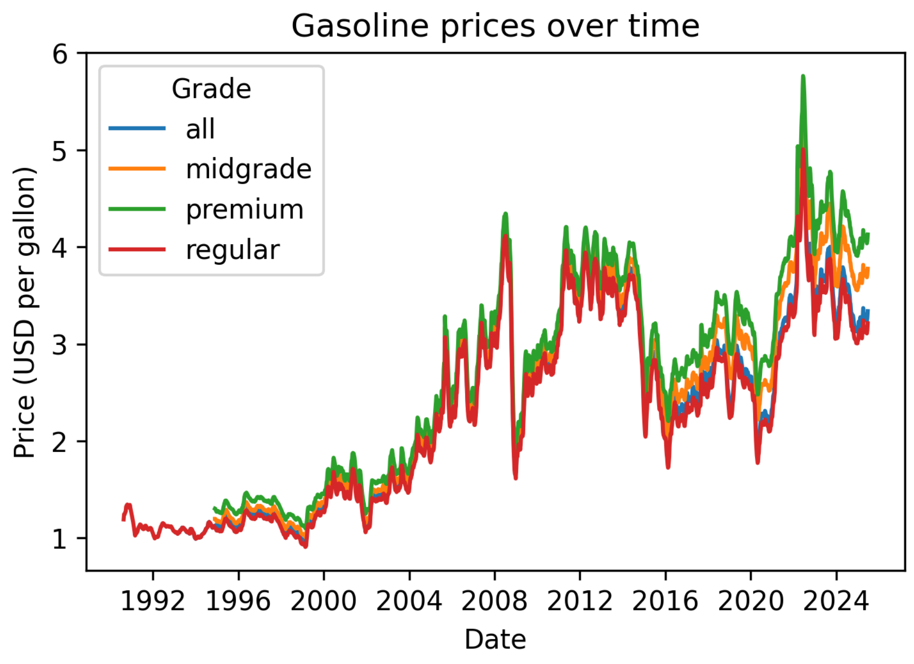
```
import pandas as pd
import matplotlib.pyplot as plt

data_url = 'https://raw.githubusercontent.com/rfordatascience/tidytuesday/main/data/2025/2025
df = pd.read_csv(data_url, parse_dates=["date"])
```

```
filtered = df[
    (df["fuel"] == "gasoline") &
    (df["formulation"] == "all")
]
filtered = filtered.sort_values("date")

for grade, group in filtered.groupby("grade"):
    plt.plot(group["date"], group["price"], label=grade)

plt.title("Gasoline prices over time")
plt.xlabel("Date")
plt.ylabel("Price (USD per gallon)")
plt.legend(title="Grade")
```



Push to Github

(1) `Source Control`

- `Source Control` on the leftside
- Enter a message and then commit
- click the clould button to upload a branch

(2) `Posit Publisher`

- `Posit Publisher` on the leftside
- Can enter url to deploy on my own server

Python

Strings - Working with Textual Data

```python
# print hello world
print('Hello World')

my_message1 = 'Crispy\'s World'
print(my_message1)

my_message = "Crispy's World"
print(my_message)
```

```
Hello World
Crispy's World
Crispy's World
```

```python
print(len(my_message1))
print(len(my_message))
```

```
14
14
```

```python
print(my_message[0])  # the 1st
print(my_message[6])  # the 7th, which is "'"
# print(my_message[15]) string index out of range
print(my_message[0:6])  # include [0] but not include [6]
print(my_message[:6])
print(my_message[6:])
```

```
C
'
Crispy
Crispy
's World
```

```python
print(my_message.lower())
print(my_message.upper())
print(my_message.count("s"))
print(my_message.find("s"))  # it only returns the first 's', which is [3]
```

```
crispy's world
CRISPY'S WORLD
2
3
```

```python
my_message.replace("World", "Universe")
print(my_message)
new_message = my_message.replace("World", "Universe")
print(new_message)
```

```
Crispy's World
Crispy's Universe
```

```python
greeting = "Hello"
name = "Crispy"
message = greeting + name
print(message)
message = greeting + ", " + name
print(message)
message = greeting + ", " + name + ". Welcome!"
print(message)

message = "{}, {}. Welcome!".format(greeting, name)
print(message)
```

```
HelloCrispy
Hello, Crispy
Hello, Crispy. Welcome!
Hello, Crispy. Welcome!
```

```python
message = f'{greeting}, {name}. Welcome!'
print(message)
message = f'{greeting}, {name.upper()}. Welcome!'
print(message)
```

```
Hello, Crispy. Welcome!
Hello, CRISPY. Welcome!
```

```
print(dir(name))  # Show functions that can be applied to it
# print(help(str))
print(help(str.lower))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__fo
Help on method descriptor lower:

lower(self, /) unbound builtins.str method
    Return a copy of the string converted to lowercase.

None
```

Integers and Floats - Working with Numeric Data

```
num=3
print(type(num))
```

```
<class 'int'>
```

```
num=3.14
print(type(num))
```

```
<class 'float'>
```

```
# Arithmetic Operators:
# Addition:       3 + 2
# Subtraction:    3 - 2
# Multiplication: 3 * 2
# Division:       3 / 2
# Floor Division: 3 // 2
# Exponent:       3 ** 2
# Modulus:        3 % 2

print(3 + 2)
print(3 - 2)
```

```python
print(3 * 2)
print(3 / 2)
print(3 // 2)
print(3**2)
print(3 % 2)

print(2 % 2)
print(3 % 2)
print(4 % 2)
print(5 % 2)

print(3 * 2 + 1)
```

```
5
1
6
1.5
1
9
1
0
1
0
1
7
```

```python
num = 1
num = num+1
print(num)

num = 1
num *= 10
print(num)
```

```
2
10
```

```python
print(abs(-3))
print(round(3.75))
print(round(3.75, 1))
```

```
3
4
3.8
```

```python
# Comparisons:
# Equal:            3 == 2
# Not Equal:        3 != 2
# Greater Than:     3 > 2
# Less Than:        3 < 2
# Greater or Equal: 3 >= 2
# Less or Equal:    3 <= 2

num_1 = 3
num_2 = 2
print(num_1 == num_2)
print(num_1 != num_2)
print(num_1 > num_2)
print(num_1 < num_2)
print(num_1 >= num_2)
print(num_1 <= num_2)
```

```
False
True
True
False
True
False
```

```python
num_1 = "100"
num_2 = "200"  # as str
print(num_1 + num_2)

num_1 = int(num_1)
num_2 = int(num_2)
print(num_1 + num_2)
```

```
100200
300
```

Lists, Tuples, and Sets

Lists

```python
courses = ["History", "Math", "Physics", "CompSci"]
print(courses)
print(courses[0])
print(courses[3])
print(courses[-1])  # convenient to get the last one
print(courses[0:2])  # include [0] but not include [2]
print(courses[2:])
```

```
['History', 'Math', 'Physics', 'CompSci']
History
CompSci
CompSci
['History', 'Math']
['Physics', 'CompSci']
```

```python
courses = ["History", "Math", "Physics", "CompSci"]
courses.append("Art")
print(courses)

courses = ["History", "Math", "Physics", "CompSci"]
courses.insert(0, "Art")
print(courses)

courses = ["History", "Math", "Physics", "CompSci"]
courses_2 = ["Art", "Education"]
courses.insert(0, courses_2)
print(courses)

courses = ["History", "Math", "Physics", "CompSci"]
courses_2 = ["Art", "Education"]
courses.extend(courses_2)  # or courses_2.extend(courses)
print(courses)

courses = ["History", "Math", "Physics", "CompSci"]
courses.remove("Math")
print(courses)
```

```
['History', 'Math', 'Physics', 'CompSci', 'Art']
['Art', 'History', 'Math', 'Physics', 'CompSci']
[['Art', 'Education'], 'History', 'Math', 'Physics', 'CompSci']
['History', 'Math', 'Physics', 'CompSci', 'Art', 'Education']
['History', 'Physics', 'CompSci']
```

```python
courses = ["History", "Math", "Physics", "CompSci"]
popped = courses.pop()
print(popped)
print(courses)
```

```
CompSci
['History', 'Math', 'Physics']
```

```python
courses = ["History", "Math", "Physics", "CompSci"]
courses.reverse()
print(courses)

courses = ["History", "Math", "Physics", "CompSci"]
courses.sort()  # Alphabet
print(courses)

nums = [1, 4, 3, 5, 2]
nums.sort() # assending
print(nums)

courses = ["History", "Math", "Physics", "CompSci"]
courses.sort(reverse=True)
print(courses)

nums = [1, 4, 3, 5, 2]
nums.sort(reverse=True)
print(nums)
```

```
['CompSci', 'Physics', 'Math', 'History']
['CompSci', 'History', 'Math', 'Physics']
[1, 2, 3, 4, 5]
['Physics', 'Math', 'History', 'CompSci']
[5, 4, 3, 2, 1]
```

```python
# These methods above will change the original item
# To avoid changing:

courses = ["History", "Math", "Physics", "CompSci"]
sorted_courses = sorted(courses)
print(courses)
```

```
['History', 'Math', 'Physics', 'CompSci']
```

```python
nums = [1, 4, 3, 5, 2]
print(min(nums))
print(max(nums))
print(sum(nums))

courses = ["History", "Math", "Physics", "CompSci"]
print(courses.index("CompSci")) # courses[3]
print("Math" in courses) # True or False
```

```
1
5
15
3
True
```

```python
for item in courses:  # we can name it by any names we want, not just 'item'
    print(item)


for index, course in enumerate(courses, start=1):
    print(index, course)
```

```
History
Math
Physics
CompSci
1 History
2 Math
3 Physics
4 CompSci
```

```python
course_str = "-".join(courses)
print(course_str)
```

```
History-Math-Physics-CompSci
```

```python
new_list = course_str.split("-")
print(new_list)
```

```
['History', 'Math', 'Physics', 'CompSci']
```

Tuples

```python
# Mutable
list_1 = ["History", "Math", "Physics", "CompSci"]
list_2 = list_1

print(list_1)
print(list_2)

list_1[0] = "Art"

print(list_1)
print(list_2) # list_2 equals to list_1 so it would change as list_1 change
```

```
['History', 'Math', 'Physics', 'CompSci']
['History', 'Math', 'Physics', 'CompSci']
['Art', 'Math', 'Physics', 'CompSci']
['Art', 'Math', 'Physics', 'CompSci']
```

```python
# Immutable
tuple_1 = ("History", "Math", "Physics", "CompSci")
tuple_2 = tuple_1

print(tuple_1)
print(tuple_2)

# tuple_1[0] = "Art"
# TypeError: 'tuple' object does not support item assignment
```

```
('History', 'Math', 'Physics', 'CompSci')
('History', 'Math', 'Physics', 'CompSci')
```

Sets

```python
# Sets
cs_courses = {'History', 'Math', 'Physics', 'CompSci'}
print(cs_courses)

cs_courses = {'History', 'Math', 'Physics', 'CompSci', 'Math'}
print(cs_courses)

print('Math' in cs_courses)
```

```
{'History', 'Math', 'Physics', 'CompSci'}
{'History', 'Math', 'Physics', 'CompSci'}
True
```

```python
cs_courses = {'History', 'Math', 'Physics', 'CompSci'}
art_courses = {'History', 'Math', 'Art', 'Design'}

print(cs_courses.intersection(art_courses))
print(cs_courses.difference(art_courses))
print(cs_courses.union(art_courses))
```

```
{'History', 'Math'}
{'Physics', 'CompSci'}
{'Art', 'Design', 'Math', 'CompSci', 'Physics', 'History'}
```

```python
# Empty Lists
empty_list = []
empty_list = list()

# Empty Tuples
empty_tuple = ()
empty_tuple = tuple()

# Empty Sets
#empty_set = {} # This isn't right! It's a dict
empty_set = set()
```

Dictionaries - Working with Key-Value Pairs

```python
student = {'name':'Crispy', 'age':25, 'courses':['Math','CompSci']}
print(student)
print(student['courses'])
# print(student['phone']) # KeyError: 'phone'
print(student.get('phone'))
print(student.get('phone','Not Found')) # 'Not Found' is a desired return for None

student['phone']='555-5555-5555'
print(student.get('phone','Not Found'))

student.update({'name':'Jane', 'age': 26, 'phone':'666-6666-6666'})
print(student)

student = {'name':'Crispy', 'age':25, 'courses':['Math','CompSci']}
del student['age']
print(student)

student = {'name':'Crispy', 'age':25, 'courses':['Math','CompSci']}
age = student.pop('age')
print(student)
print(age)
```

```
{'name': 'Crispy', 'age': 25, 'courses': ['Math', 'CompSci']}
['Math', 'CompSci']
None
Not Found
555-5555-5555
{'name': 'Jane', 'age': 26, 'courses': ['Math', 'CompSci'], 'phone': '666-6666-6666'}
{'name': 'Crispy', 'courses': ['Math', 'CompSci']}
{'name': 'Crispy', 'courses': ['Math', 'CompSci']}
25
```

```python
student = {'name':'Crispy', 'age':25, 'courses':['Math','CompSci']}
print(len(student))
print(student.keys())
print(student.items())
```

```
3
```

```
dict_keys(['name', 'age', 'courses'])
dict_items([('name', 'Crispy'), ('age', 25), ('courses', ['Math', 'CompSci'])])
```

```python
for key in student:
    print(key)
```

```
name
age
courses
```

```python
for key,value in student.items():
    print(key,value)
```

```
name Crispy
age 25
courses ['Math', 'CompSci']
```

Conditionals and Booleans - If, Else, and Elif Statements

```python
if True:
    print('Contitional was True')
```

```
Contitional was True
```

```python
if False:
    print('Contitional was True')

# nothing will be return if False
```

```python
language = "python"

if language == "python":
    print("Contitional was True")

#  `language == 'python'` equals to `True`
```

```
Contitional was True
```

```
# Comparisons:
# Equal:            ==
# Not Equal:        !=
# Greater Than:     >
# Less Than:        <
# Greater or Equal: >=
# Less or Equal:    <=
# Object Identity:  is
```

```
language = "python"
if language == "python":
    print("Language is Python")
else:
    print("No match")

language = "JAVA"
if language == "python":
    print("Language is Python")
elif language == "JAVA":
    print("Language is JAVA")
else:
    print("No match")

language = "R"
if language == "python":
    print("Language is Python")
elif language == "JAVA":
    print("Language is JAVA")
else:
    print("No match")
```

```
Language is Python
Language is JAVA
No match
```

```
# and
# or
# not

user = "Admin"
logged_in = True
```

```python
if user == "Admin" and logged_in:
    print("Admin Page")
else:
    print("Bad Creds")

logged_in = False
if not logged_in:
    print("Please log in")
else:
    print("Welcome")
```

```
Admin Page
Please log in
```

```python
# Difference between `==` and `is`
a = [1, 2, 3]
b = [1, 2, 3]
print(a == b)
print(a is b)

print(id(a))
print(id(b))
print(id(a) == id(b))
```

```
True
False
2387091937984
2387521558784
False
```

```python
# False Values:
# False
# None
# Zero of any numeric type
# Any empty sequence. For example, '', (), [].
# Any empty mapping. For example, {}.

condition = False  # same as `None`, `0`, `''`, `{}`, ...
if condition:
    print("Evaluated to True")
else:
    print("Evaluated to False")

# False will lead to else
```

```
Evaluated to False
```

```python
condition = "Test"  # not False means True
if condition:
    print("Evaluated to True")
else:
    print("Evaluated to False")
```

```
Evaluated to True
```

Loops and Iterations - For/While Loops

```python
nums = [1, 2, 3, 4, 5]

for num in nums:
    print(num)
```

```
1
2
3
4
5
```

```python
nums = [1, 2, 3, 4, 5]

for num in nums:
    if num == 3:
        print('Found!')
        break # break statement: to stop the loops
    print(num)
```

```
1
2
Found!
```

```python
nums = [1, 2, 3, 4, 5]

for num in nums:
    if num == 3:
        print('Found!')
        continue # continue statement: continue to the next iteration
    print(num)
```

```
1
2
Found!
4
5
```

```python
# Nested list
nums = [1, 2, 3, 4, 5]

for num in nums:
    for letter in "abc":
        print(num, letter)

# Give all the combinations
```

```
1 a
1 b
1 c
2 a
2 b
2 c
3 a
3 b
3 c
4 a
4 b
4 c
5 a
5 b
5 c
```

```python
for i in range(10): # from 0 to 9 (10 not included)
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```python
for i in range(1,11): # include 1 but not 11
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

loop will stop only until a certen condition is met or we hit a break

```python
x = 0

while x < 10:  # a certen condition is met
    print(x)
    x += 1
```

```
0
1
2
3
```

```
4
5
6
7
8
9
```

```python
x = 0

while x < 10:
    if x == 5:
        break  # hit a break
    print(x)
    x += 1
```

```
0
1
2
3
4
```

```python
x = 0

while True: # create an infinite loop
    if x == 5:
        break  # we must have a break statement otherwise it won't stop
    # In those cases, use `Ctrl+C` to stop
    print(x)
    x += 1
```

```
0
1
2
3
4
```