CS506 Programming for Computing
HOS04 – NumPy & pandas
03/08/2023 Reviewed by Maram Elwakeel
09/28/2024 Reviewed by Naveena Moddu
03/15/2025 Reviewed by Clark Ngo
School of Technology & Computing (STC)
City University of Seattle (CityU)

Before You Start

- Screenshots may be different from your environment.
- The directory path shown in screenshots may be different from yours.
- There might be subtle discrepancies along with the steps. Please use your best judgment while going through this cookbook-style tutorial to complete each step.
- Some steps may not be explained in detail. If you are not sure what to do:

1. Consult the resources from the course.
2. If you cannot solve the problem after a few tries (usually 15 -30 minutes), ask a TA for help.

Resources

- Data from Kaggle: https://www.kaggle.com/datasets
- NumPy Documentation: https://numpy.org/doc/stable/
- McIntire, G., Martin, B., & Washington, L. Pandas A Complete Introduction. Retrieved from https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/
- Pandas User Guide: https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html
- Pandas Documentation: https://pandas.pydata.org/docs/user_guide/io.html?highlight=json#io-json-writer
- Pandas Documentation: https://pandas.pydata.org/docs/user_guide/index.html
- Pandas.qcut(): https://pandas.pydata.org/docs/reference/api/pandas.qcut.html
- Pandas.cut(): https://pandas.pydata.org/docs/reference/api/pandas.cut.html
- Pandas.tranform(): https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.transform.html?highlight=transform#pandas.DataFrame.transform
- Stanford University. (2020). CS231n: Convolutional Neural Networks for Visual Recognition: NumPy.

Learning Outcomes

- **Section 1: Jupyter Notebook in Codespaces**
- **Section 2: NumPy**
- **Section 3: What is pandas?**
- **Section 4: Read and write with pandas**
- **Section 5: Data transformation**
- **Section 6: Data Discretization**

Section 1: Jupyter Notebook in Codespaces
Follow instructions on how to install Jupyter Notebook on GitHub Codespaces here:
https://cityuseattle.github.io/docs/environment/codespaces_jupyter/

Learn and try Jupyter Notebook from JupyterLab: https://jupyter.org/try-jupyter/lab/

Section 2: NumPy
Which stands for Numerical Python, is an open-source and the core library for scientific computing in Python. It provides a high-performance multidimensional array object and tools for working with these arrays.

NumPy is memory efficient, meaning it can handle a vast amount of data more accessible than any other library. We'll cover the following in this module:

i. Datatypes
ii. Arrays

Datatypes - The object defined in NumPy is an N-dimensional array type called ndarray, describing the collection of items of the same type. An element in ndarray is an object of data-type object called dtype.

i. A **dtype** described in the following aspects:
   - Type of data (integer, float, Python object, etc.)
   - Size of the data (how many bytes is in e.g., the integer)
   - Byte order of the data (little-endian or big-endian)
   - Structured data type (*e.g.*, describing an array item consisting of an integer and a float)

2) Create a new file called datatypes.ipynb and simply click on the file to open notebook.
3) Type the following to see how NumPy says about data types. Run selected cell to see result for that cell.

```
import numpy as np
```

```
dt = np.array([1, 2])
print(dt.dtype)                 # Output > "int64"

dt = np.array([1.0, 2.0])
print(dt.dtype)                  # Output > "float64"

dt = np.array([1, 2], dtype=np.float64)   # Force a particular datatype
print(dt)
print(dt.dtype)
```

```
int64
float64
[1. 2.]
float64
```

We also can define a structured data type. For example, we will define a data type named student with the 3 fields: name as string, age as integer, points as float.

```python
student = np.dtype([('name', np.str_, 10), ('age', np.int64), ('marks', np.float64)])
print(student)
```

[4] ✓ 0.0s Python

```
[('name', '<U10'), ('age', '<i8'), ('marks', '<f8')]
```

Let's apply to ndarray object.

[4]

```python
ms_student = np.array([('student1', 20, 100), ('student2withlongname', 22, 89)]
, dtype=student)
print(ms_student)
print(ms_student.dtype)
```

```
[('student1', 20, 100.) ('student2wi', 22,  89.)]
[('name', '<U10'), ('age', '<i8'), ('marks', '<f8')]
```

As you can see here, *"student2withlongname"* has exceeded the string length, only 10 characters acceptance was defined.

Arrays - A numpy array is a grid of values like Python lists, but all the same type, and is indexed by a tuple of nonnegative integers. The rank and shape are the number of dimensions and a tuple of integers of the size of the array, respectively.

i. Create a new file called arrays.ipynb and simply click on the file to open notebook.
ii. Type the following to create numpy arrays. Accessing items in array is similar to Python list.

```python
import numpy as np
```

```python
# create rank 1 dim array
arr = np.array([1, 2, 3])
print(type(arr))                          # outputs <class 'numpy.ndarray'>
print(arr.dtype)                          # outputs int64
print(arr.shape)                          # outputs (3,)
print(arr[0], arr[1], arr[2])             # outputs 1 2 3
arr[2] = 4                                # change last element to 4
print(arr)                                # outputs [1 2 4]

# create rank 2 array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2.shape)                         # outputs (2, 3)
print(arr2[0, 0], arr2[0, 1], arr2[1, 0]) # outputs 1 2 4
print(arr2[-1])                           # outputs last element [4 5 6]
```

iii. Stay in same file, type to following functions from numpy to generate arrays in a different way. You will see how it can make your life easier with array.

```python
# create an array of zeros with 2 dim and size of 2 each
zero = np.zeros((2,2))
print(zero)                      # outputs "[[0. 0.]
                                 #          [0. 0.]]"

# create an array of ones with 1 dim and size of 2 each
one = np.ones((1,2))
print(one)                       # outputs "[[ 1.  1.]]"

# create a new array with given shape, type, and filled with value
constant = np.full((2,2), 5)     # defaule type is float
print(constant)                  # outputs "[[5 5]
                                 #           [5 5]]"

# create a 2-D array with 1 on diagonal and 0 on others
diag = np.eye(3)                 # number of rows
print(diag)                      # outputs [[1. 0. 0.]
                                 #          [0. 1. 0.]
                                 #          [0. 0. 1.]]
# diag starts at index 1
diag = np.eye(4, k=1)
print(diag)                      # outputs [[0. 1. 0. 0.]
                                 #          [0. 0. 1. 0.]
                                 #          [0. 0. 0. 1.]
                                 #          [0. 0. 0. 0.]]

# Create an array with random values
rand = np.random.random((2,2))
print(rand)
```

iv. Indexing and slicing in NumPy array is similar to Lists in Python. Try the following into a new cell.

```python
# Array indexing

arr = np.arange(10)          # genarate array [0 1 2 3 4 5 6 7 8 9]
print(arr[0:8:2])            # every two number from 0 up to but not include 8

# create rank 2 with shape (3, 4) array
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(arr[:2, :2])           # outputs [[1 2]
                             #          [5 6]]
```

v. In NumPy, you can do integer indexing which helps in selecting any arbitrary item in an array using the data from another array like the example below. Add this to new cell.

```python
# Integer indexing

arr = np.array([[1,2], [3, 4], [5, 6]])
print(arr[[0, 1, 2], [0, 1, 1]])                # outputs [1 4 6]

# arr[[0, 1, 2], [0, 1, 1]] is equivalent to [arr[0, 0], arr[1, 1, arr[2, 1]]]
```

Section 3: What is pandas?

pandas (derived from the word Panel Data – an Econometrics from Multidimensional data – Tutorialspoint) is a powerful, open-source Python library providing high-performance, easy-to-use data structures for data analysis, manipulation, and visualization.

Features of Pandas

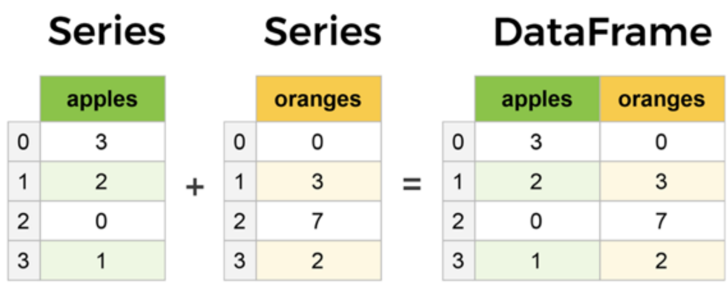
- Fast and efficient DataFrame object
- Tools for loading data into in-memory data objects from different file formats.
- Label-based slicing, indexing and subsetting of large data sets.
- Columns from a data structure can be deleted or inserted.
- Group by data for aggregation and transformations.
- High performance merging and joining of data.

Core Components - The two primary data structures of pandas, Series and DataFrame.

ii. **Series** – 1D labeled homogeneously-typed array

iii. **DataFrame** – General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column

A Series is essentially a column, and a DataFrame is a multi-dimensional table made up of a collection of Series.



1) Create a new file called pandas_object.ipynb and simply click on the file to open notebook.
2) Type the following into the file just created. Run selected cell to see each result.

Pandas object creation

```python
import pandas as pd
```

```python
series = pd.Series([1, 3, 5, 6, 8])          # create Series data structure
series
```

```
0    1
1    3
2    5
3    6
4    8
dtype: int64
```

3) We just created Series object with pandas, next we will create DataFrame, which is a collection of Series.

DataFrame

By default, pandas creates indexes for us, but we can specify them with parameter "index" See the different table from DataFrame below.

```python
data = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2],
    'peaches': [2, 4, 6, 8]
}
fruits = pd.DataFrame(data)                    # create DataFrame with 3 columns
fruits
```

	apples	oranges	peaches
0	3	0	2
1	2	3	4
2	0	7	6
3	1	2	8

Each (key, value) item in data corresponds to a column in the resulting DataFrame.

4) The previous DataFrame, index was given at the creation by default, but we can create our own labels as the following example.

```
buyer = pd.DataFrame(data, index=['Tom', 'Isabelle', 'Daisy', 'Blathers'])
buyer
```

	apples	oranges	peaches
Tom	3	0	2
Isabelle	2	3	4
Daisy	0	7	6
Blathers	1	2	8

5) We gave string names as index replacement, so it is more convenient to locate the data by name. For example:

   buyer.loc['Tom']

   Output:

```
apples      3
oranges     0
peaches     2
Name: Tom, dtype: int64
```

6) That's the basic pandas, there is a lot more from pandas that you can do. Learn more here: https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html

Section 4: Read and write with pandas

Pandas is a very powerful and popular framework for data analysis and manipulation. One of the most useful features of Pandas is the ability to read and write various types of files including CSV.

The Pandas I/O API is a set of top-level reader functions accessed like pd.read_csv() that generally return a Pandas object. More available readers and writers can be found here https://pandas.pydata.org/docs/user_guide/io.html

Common functionalities:

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
	MS Excel	read_excel	to_excel

1) Verify that you have the file sales_data_sample.csv
2) Create a new file called reading_writing.ipynb and simply click on the file to open notebook.
3) Type the following into the file just created. Run selected cell to see each result.

```
import pandas as pd
```

```
sales = pd.read_csv('sales_data_sample.csv', encoding='latin')
```

```
sales
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSLINE1	ADDRESSLINE2	CITY	STATE	POS
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	897 Long Airport Avenue	NaN	NYC	NY	
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	5	2003	...	59 rue de l'Abbaye	NaN	Reims	NaN	
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003	...	27 rue du Colonel Pierre Avia	NaN	Paris	NaN	
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	78934 Hillside Dr.	NaN	Pasadena	CA	
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	7734 Strong St.	NaN	San Francisco	CA	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2818	10350	20	100.00	15	2244.40	12/2/2004 0:00	Shipped	4	12	2004	...	C/ Moralzarzal, 86	NaN	Madrid	NaN	
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	Shipped	1	1	2005	...	Torikatu 38	NaN	Oulu	NaN	
2820	10386	43	100.00	4	5417.57	3/1/2005 0:00	Resolved	1	3	2005	...	C/ Moralzarzal, 86	NaN	Madrid	NaN	
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	Shipped	1	3	2005	...	1 rue Alsace-Lorraine	NaN	Toulouse	NaN	
2822	10414	47	65.52	9	3079.44	5/6/2005 0:00	On Hold	2	5	2005	...	8616 Spinnaker Dr.	NaN	Boston	MA	

2823 rows × 25 columns

'latin-1' or 'iso-8859-1' is the simplest text encoding maps the code points 0–255 to the bytes 0x0–0xff, which means that a string object that contains code points above U+00FF can't be encoded with this codec. Doing so will raise a **UnicodeEncodeError.**

4) This data has 25 columns, you might not see the whole thing, but you still can see what they are with the columns DataFrame like below.

```
sales.columns
```

```
Index(['ORDERNUMBER', 'QUANTITYORDERED', 'PRICEEACH', 'ORDERLINENUMBER',
       'SALES', 'ORDERDATE', 'STATUS', 'QTR_ID', 'MONTH_ID', 'YEAR_ID',
       'PRODUCTLINE', 'MSRP', 'PRODUCTCODE', 'CUSTOMERNAME', 'PHONE',
       'ADDRESSLINE1', 'ADDRESSLINE2', 'CITY', 'STATE', 'POSTALCODE',
       'COUNTRY', 'TERRITORY', 'CONTACTLASTNAME', 'CONTACTFIRSTNAME',
       'DEALSIZE'],
      dtype='object')
```

5) Now we will write a new CSV file with multiple columns selected by the indexing operator by passing a list of column names.

```
data = pd.DataFrame(sales.loc[:50,['ORDERNUMBER', 'CUSTOMERNAME', 'ADDRESSLINE1', 'POSTALCODE', 'STATUS']])

data.to_csv('new_sales.csv')
```

6) The Series and DataFrame objects have an instance method to_csv which allows storing the contents of the object as a comma-separated-values file.
   i. There are a couple common exceptions that arise when doing selections with just the indexing operator.
   ii. If you misspell a word, you will get a KeyError
   iii. If you forgot to use a list to contain multiple columns, you would also get a KeyError
   iv. You just created a new CSV file named new_sales.csv with to_csv method of DataFrame.
   v. Check the contents of your new_sales.csv

Section 5: Data transformation

The process of changing the format, structure, or values of data so that the result may be more efficient and easier to understand. This involves converting data from one structure to another so you can integrate it with different applications.

With Pandas, we can perform transformation. For this section, we will use the transform function.

1) Create a new file named data_transformation.ipynb.
2) Create a DataFrame with the following rows and columns.

```
import numpy as np
import pandas as pd
```

```
trans = pd.DataFrame({"A":[12, 4, 5, None, 1],
                      "B":[7, 2, 54, 3, None],
                      "C":[20, 16, 11, 3, 8],
                      "D":[14, 3, None, 2, 6]},
                      index=['Row_1', 'Row_2', 'Row_3', 'Row_4', 'Row_5'])
trans
```

	A	B	C	D
Row_1	12.0	7.0	20	14.0
Row_2	4.0	2.0	16	3.0
Row_3	5.0	54.0	11	NaN
Row_4	NaN	3.0	3	2.0
Row_5	1.0	NaN	8	6.0

3) Now we will use transform() function to add 10 to each element of the data frame.
   i. DataFrame.transform call func on self-producing a DataFrame with transformed values. Produced DataFrame will have same axis length as self.

```
result = trans.transform(lambda x : x + 10)
result
```

	A	B	C	D
Row_1	22.0	17.0	30	24.0
Row_2	14.0	12.0	26	13.0
Row_3	15.0	64.0	21	NaN
Row_4	NaN	13.0	13	12.0
Row_5	11.0	NaN	18	16.0

Notice that we used an anonymous function (lambda) to add 10 to each value. So, all non-empty values have been added successfully.

Section 6: Data Discretization

Defined as a process of converting continuous data attribute values into a finite set of intervals with minimal loss of information. For example, you have height data and want to discretize it to 0 and 1 interval depending on if the height is below or above a certain value of height.

1) Within the same file, create a numpy array with 10 random integers between 10 and 200.

```
# 10 random numbers from 10-200
x = np.random.randint(10, 200, size=10)
x

array([144,  88, 180, 157,  96,  29, 123, 128,  40, 133])
```

2) Numpy provides some of the functions you can perform discretize with, one is using digitize() for doing so.

3) In this step, we will discretize where 50 is the threshold to divide the data into two categories: one less than 50 and ones above 50.

```
np.digitize(x,bins=[50])

array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1])
```

- The bins argument is a list and so we can specify multiple binning or discretizing conditions.
- As you can see, there are only two numbers lower than 50 represented as 0. There are more than 50 represented as 1.

4) Let's say if we want to have 3 categories, how many values should we specify in the bins array? You got it, 2 values.

```
np.digitize(x, bins=[50, 100])

array([2, 1, 2, 2, 1, 0, 2, 2, 0, 2])
```

- We have what we wanted. There are 3 categories here: 0, 1, and 2. 0 category has the values of less than 50, the 1 category has the value of less than 100 and category 3 has the value of more than 100.

5) Now let’s use a function from Pandas called cut() to discretize our data.
6) It does the same thing as what Numpy’s digitize() does, but the way it works is a bit different. Let’s first create a DataFrame with random number from previous.

```python
df = pd.DataFrame({"height": x})
df
```

	height
0	144
1	88
2	180

- We created 10 values in height columns with index 0-9.

7) Categorize the height variable into four categories using Pandas cut function.

```python
# create new column 'binned' with interval
df['binned'] = pd.cut(x=df['height'], bins=[0, 25, 50, 100, 200])
df
```

	height	binned
0	144	(100, 200]
1	88	(50, 100]
2	180	(100, 200]
3	157	(100, 200]
4	96	(50, 100]
5	29	(25, 50]
6	123	(100, 200]
7	128	(100, 200]
8	40	(25, 50]
9	133	(100, 200]

- The height values between 0 and 25 are in one category, height between 25 and 50 are in the second category, 50-100 in third category, and 100-200 are in the fourth category.

8) We also can label them as follows.

```
df['bin_label'] = pd.cut(x = df['height'],
                         bins = [0, 25, 50, 100, 200],
                         labels = [1, 2, 3, 4])
df
```

	height	binned	bin_label
0	144	(100, 200]	4
1	88	(50, 100]	3
2	180	(100, 200]	4
3	157	(100, 200]	4
4	96	(50, 100]	3
5	29	(25, 50]	2
6	123	(100, 200]	4
7	128	(100, 200]	4
8	40	(25, 50]	2
9	133	(100, 200]	4

- As my random data does not have values below 25 so there is no category 1 in this example. Your data may be different.
- You can change labels to string instead of number.

9) Pandas also have another function called qcut() which discretize variable into equal-sized buckets. You only need to tell the function the number of quantiles, pandas will figure out how to bin that data.

```
pd.qcut(df['height'], q=5)
```

```
0    (130.0, 146.6]
1     (78.4, 112.2]
2    (146.6, 180.0]
3    (146.6, 180.0]
4     (78.4, 112.2]
5    (28.999, 78.4]
6    (112.2, 130.0]
7    (112.2, 130.0]
8    (28.999, 78.4]
9    (130.0, 146.6]
Name: height, dtype: category
Categories (5, interval[float64]): [(28.999, 78.4] < (78.4, 112.2] < (112.2, 130.0] < (130.0, 146.6] < (146.6, 180.0]]
```

- Notice that qcut discretized the data into equal size but the interval is not equal. Binned into 5 groups with 2 members each. Unlike what cut function does which is no guarantee about the distribution of items in each bin, but the interval pretty much the same for all bin.

	Spacing between 2 Bins	Frequency of Samples in Each Bins
cut	Equal Spacing	Different
qcut	Unequal Spacing	Same

10) Let's look at a familiar example below of how to use discretize with data.

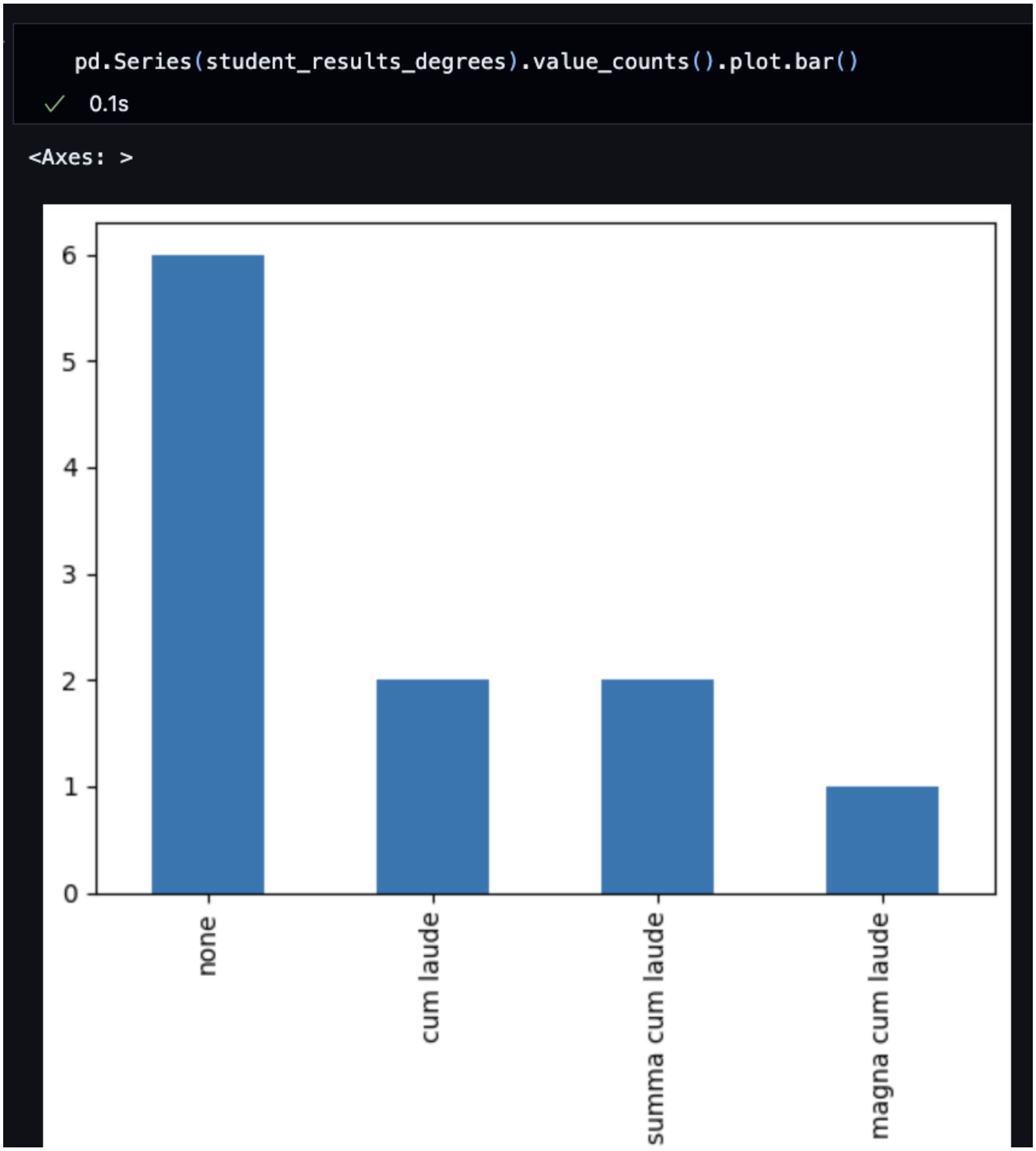
```python
degrees = ["none", "cum laude", "magna cum laude", "summa cum laude"]
student_results = [3.93, 3.24, 2.80, 2.83, 3.91, 3.698, 3.731, 3.25, 3.24, 3.82, 3.22]
student_results.sort(reverse=True)

student_results_degrees = pd.cut(student_results, [0, 3.6, 3.8, 3.9, 4.0], labels=degrees)
honor = pd.DataFrame({'grades': student_results,
                      'honors': student_results_degrees})
honor
```

	grades	honors
0	3.930	summa cum laude
1	3.910	summa cum laude
2	3.820	magna cum laude
3	3.731	cum laude
4	3.698	cum laude
5	3.250	none
6	3.240	none
7	3.240	none
8	3.220	none
9	2.830	none
10	2.800	none

- Line number 3, we sorted for easier to see in each category. We categorized what grades belong to which categories. Then, we put the data into DataFrame.

11) If you are a visual person, you need to see the chart. You can do that with a basic plotting provided by pandas like the below image.
12) We will do a bar plot based on the number of students in each group.



13) If you want to see what value_counts() does, remove ".plot.bar()"

Push your work to GitHub
Follow instructions here: https://cityuseattle.github.io/docs/git/codespaces_submission/