Welcome to your 310 Portfolio

A progression of learning for CSC/DSP 310: Programming for Data Science at University of Rhode Island.

About Me

Data Science, to me

Grading in this class

```
def compute_grade(num_level1,num_level2,num_level3):
    '''
    Computes a grade for CSC/DSP310 from numbers of achievements at each level

    Parameters:
    ------------
    num_level1 : int
      number of level 1 achievements earned
    num_level2 : int
      number of level 2 achievements earned
    num_level3 : int
      number of level 3 achievements earned

    Returns:
    --------
    letter_grade : string
      letter grade with modifier (+/-)
    '''
    if num_level1 == 15:
        if num_level2 == 15:
            if num_level3 == 15:
                grade = 'A'
            elif num_level3 >= 10:
                grade = 'A-'
            elif num_level3 >=5:
                grade = 'B+'
            else:
                grade = 'B'
        elif num_level2 >=10:
            grade = 'B-'
        elif num_level2 >=5:
            grade = 'C+'
        else:
            grade = 'C'
    elif num_level1 >= 10:
        grade = 'C-'
    elif num_level1 >= 5:
        grade = 'D+'
    elif num_level1 >=3:
        grade = 'D'
    else:
        grade = 'F'


    return grade
```

```
compute_grade(15,15,15)
```

```
'A'
```

```
compute_grade(14,14,14)
```

```
'C-'
```

```
assert compute_grade(14,14,14) == 'C-'
```

```
assert compute_grade(15,15,15) == 'A'
```

```
assert compute_grade(15,15,11) == 'A-'
```

Portfolio Check 1

I've demonstrated level 2 for process in Podcast Review by explaining the phases and how they related to the topic that episode.

I had trouble with assignment 2, so I redid that to demonstrate level 2 access in Loading Data. For level 3 of access, I added extra analysis comparing different ways of loading data in the Loading data chapter. Also across my 4 chapters, I loaded data in 3 different ways.

I've demonstrated level 3 for construct and prepare in Building a new Dataset by cleaning data from unrelated sources, reorganizing them so that they could be merged and merging the two on the new column that I added to each.

In the Cool Analysis chapter, I demonstrate level 3 for summarize and visualize. I used my built dataset from the previous chapter and found cool insights supported with statistics and plots.

I demonstrate level 3 python throughout all of the chapters because I did `specific things.`

Topic Name Epsisode of Podcast

I listened to the Topic episode of Podcast. In the episode they talked to Expert about their work on Something Fancy. Some more details about something fancy, it's definition and what else they covered.

One interesting thing I learned was something cool.

In this episode it related to the data science process because reasons.

Loading Data

Correcting Assignment 2

On assignment 2 I was confused about a thing and did a thing wrong.

First I had a typo

```
import panda as pd
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-f542dbfa5144> in <module>
----> 1 import panda as pd

ModuleNotFoundError: No module named 'panda'
```

Now I know that the correct name is `pandas`

```
import pandas as pd
```

That cost me a lot of time but I also wrote psuedo code anyway, it wasn't right though since I couldn't see the output

```
#  code that ran but produce the wrong output
```

this should have been this way instead

```
#  code that is fixed
```

I learned that this method works by doing some steps, here is a way we can see about that functionality

```
#  code that demos a feature or inspects a value
```

Other ways of loading data

Another thing we can do loading data is something, that looks like this

```
#  code
```

this would be good for these situations, but a risk of doign that is something, we can see that by

```
#  code that does a bad thing
```

and then we investigate

```
#  code that shows why that's bad
```

This is especially good when something is true beacuse it's faster

```
%%timeit
# one way  code
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-734253614b43> in <module>
----> 1 get_ipython().run_cell_magic('timeit', '', '# one way  code\n')

/opt/hostedtoolcache/Python/3.7.9/x64/lib/python3.7/site-
packages/IPython/core/interactiveshell.py in run_cell_magic(self, magic_name, line,
cell)
   2379             with self.builtin_trap:
   2380                 args = (magic_arg_s, cell)
-> 2381                 result = fn(*args, **kwargs)
   2382             return result
   2383

<decorator-gen-53> in timeit(self, line, cell, local_ns)

/opt/hostedtoolcache/Python/3.7.9/x64/lib/python3.7/site-
packages/IPython/core/magic.py in <lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

/opt/hostedtoolcache/Python/3.7.9/x64/lib/python3.7/site-
packages/IPython/core/magics/execution.py in timeit(self, line, cell, local_ns)
   1144
   1145         t0 = clock()
-> 1146         code = self.shell.compile(timeit_ast, "<magic-timeit>", "exec")
   1147         tc = clock()-t0
   1148

/opt/hostedtoolcache/Python/3.7.9/x64/lib/python3.7/codeop.py in __call__(self,
source, filename, symbol)
    139
    140     def __call__(self, source, filename, symbol):
--> 141         codeob = compile(source, filename, symbol, self.flags, 1)
    142         for feature in _features:
    143             if codeob.co_flags & feature.compiler_flag:

ValueError: empty body on For
```

```
%%timeit
# other way code
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-86aaa9f4f0dd> in <module>
----> 1 get_ipython().run_cell_magic('timeit', '', '# other way code\n')

/opt/hostedtoolcache/Python/3.7.9/x64/lib/python3.7/site-
packages/IPython/core/interactiveshell.py in run_cell_magic(self, magic_name, line,
cell)
   2379             with self.builtin_trap:
   2380                 args = (magic_arg_s, cell)
-> 2381                 result = fn(*args, **kwargs)
   2382             return result
   2383

<decorator-gen-53> in timeit(self, line, cell, local_ns)

/opt/hostedtoolcache/Python/3.7.9/x64/lib/python3.7/site-
packages/IPython/core/magic.py in <lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

/opt/hostedtoolcache/Python/3.7.9/x64/lib/python3.7/site-
packages/IPython/core/magics/execution.py in timeit(self, line, cell, local_ns)
   1144
   1145         t0 = clock()
-> 1146         code = self.shell.compile(timeit_ast, "<magic-timeit>", "exec")
   1147         tc = clock()-t0
   1148

/opt/hostedtoolcache/Python/3.7.9/x64/lib/python3.7/codeop.py in __call__(self,
source, filename, symbol)
    139
    140     def __call__(self, source, filename, symbol):
--> 141         codeob = compile(source, filename, symbol, self.flags, 1)
    142         for feature in _features:
    143             if codeob.co_flags & feature.compiler_flag:

ValueError: empty body on For
```

Building a New Dataset

I would like to study if x and y are related, but I couldn't find a dataset about both. I found a dataset about x that included w and a dataset about y that included z. I can compute v from w and v from z.

Getting the data together

First let's get all fo the data loaded into python

Loading and cleaning X

First let's exmaine the data about x

```
# load data x
```

It has some problems:

- problem 1
- problem 2
- problem 3

First I'll fix problem 2 because that will make fixin 1 & 3 easier

The plan to fix is this high level idea

```
# comment on step 1

# second step comment
```

next

```
# more code
```

loading and cleaning y

what i'm going to do

```
# more code
```

observation.

next plan

```
# more code
```

observation

Making them compatible

computing v from w

computing v from z

Merging

I'll use this type of merge because ...

My Cool Analysis

I am interested in the relationship between x and y and in the previous chapter I built a dataset to study this.

Getting ready

Some steps

```
#  code that load data
```

Keyword

First I'll answer this question related to Keyword

```
#  code that uses summary stats like level 3 describes
```

interpretation. more questions

```
#  more code
```

more interpretation

Another keyword

Next I'll address: another question?

```
#  code that plots using new parameters from seaborn
```

evaluation of plot, it has this weakness.

```
#  code that makes the plot better using matplotlib to customize it
```

interpretation of insights from the plot

By Sarah M Brown