Welcome to your 310 Portfolio

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

About Me

Hello, my name is Brianna MacDonald and I am a Senior at URI with a double major in Computer Engineering and Chinese with a double minor in Cyber Security and Data Science. I've been studying Chinese for about 4 years and I recently had the oppertunity to go to Shanghai and Beijing last Winter right before COVID-19.

Data Science, to me

Data Science is the intersection between computer science, statistics, and domain knowledge. Data Science has many different uses, such as in medical sciences or in machine learning.

There are many different components of Data Science. The four main components of Data Science are Data Strategy, Data Engineering, Data Analysis and Modeling, and Data Visualization/Operationalization. Data Strategy is about determing what data you want to gather and why. It makes a connection between the data you want to gather and the goals for that data. Data Engineering is about the systems and technology that are used to leverage, access, organize, and use the data. Data Analysis/Modeling is about describing or predicting data, creating analysis and assumptions about data, and mathematically modeling the data. Data Visualization/Operationalization is about visualizing data and understanding how different visuals describe the data, as well as making the data operational by making a machine/person make a decision or action based on the computing of the data.

Compute the Grade for CSC/DSP 310

- To run by entering values into function, please run **compute_grade()** with desired values.

```python
def compute_grade(num_level1, num_level2, num_level3):
    """
    Computes a grade for CSC/DSP 310 from numbers of achievements earned at each level
    :param num_level1: int, number of level 1 achievements earned
    :param num_level2: int, number of level 2 achievements earned
    :param num_level3: int, number of level 3 achievements earned
    :return: letter_grade: string, letter grade with possible modifier (+/-)
    """
    # Initializing Variables
    letter_grade = ""
    total_grade = num_level1 + num_level2 + num_level3

    # Error Handling
    if total_grade > 45:
        print("Invalid total. Please re-enter values.")

    # Definitions of Grades
    else:
        if 3 <= num_level1 < 5:
            letter_grade = 'D'
        elif 5 <= num_level1 < 10:
            letter_grade = 'D+'
        elif 10 <= num_level1 < 15:
            letter_grade = 'C-'
        elif num_level1 == 15 and 0 <= num_level2 < 5:
            letter_grade = 'C'
        elif num_level1 == 15 and 5 <= num_level2 < 10:
            letter_grade = 'C+'
        elif num_level1 == 15 and 10 <= num_level2 < 15:
            letter_grade = 'B-'
        elif num_level1 == 15 and num_level2 == 15 and 0 <= num_level3 < 5:
            letter_grade = 'B'
        elif num_level1 == 15 and num_level2 == 15 and 5 <= num_level3 < 10:
            letter_grade = 'B+'
        elif num_level1 == 15 and num_level2 == 15 and 10 <= num_level3 < 15:
            letter_grade = 'A-'
        elif num_level1 == 15 and num_level2 == 15 and num_level3 == 15:
            letter_grade = 'A'
        else:
            print("Does not translate to letter grade.")
    print(f'Your grade is {letter_grade}.')
```

The example below will give a grade of a C.

```python
# Example 1
compute_grade(15, 2, 0)
```

The example below will give a grade of a B.

```python
# Example 2
compute_grade(15, 15, 2)
```

The example below will give a grade of an A-.

```python
# Example 3
compute_grade(15, 15, 12)
```

Portfolio Check 1

I've demonstrated skills *python*, *access*, *construct*, and *prepare* in chapter 1. This file can be found here.

I've demonstrated skill *process* in chapter 2. The file can be found here.

I've demonstrated skills *summarize*, *construct*, and *visualize* in chapter 3. The file can be found here.

I've demonstrated skills *prepare*, *summarize*, and *visualize* in chapter 4. The file can be found here.

All the files linked above are **.md** files, however you can also view the **.ipynb** files by clicking through the chapters.

Submission 1 - Chapter 1

In this section of my portfolio I will be demonstrating skills for **python level 3, prepare level 2, construct level 3, and access level 3**. For more demonstration on **prepare**, please look here.

Access Level 3

Access data from both common and uncommon formats and identify best practices for formats in different contexts.

There are many different ways to access data in Pandas, whether you want to read into a more common format such as a CSV file, or if you to access data stored in an HDF5 table or in SAS format.

When considering which format to use, you should also note the different types of separators that are the defaults of each method. For example, pd.read_csv defaults separation to a comma, while pd.read_table defaults to '\t'. Different file types also come with different parameters (whether optional or required) according to their designated Pandas function.

For a contextual example, consider pd.read_sql_table, pd.read_sql_query, and pd.read_sql. Although these functions seem very similar to each other, there are actually quite a few key differences. The list below describes each method and how you can use them.

pd.read_sql_table

- Reads an SQL database table into a DataFrame
- Takes in parse_dates, columns, chunksize
- Does not support DBAPI connections

pd.read_sql_query

- Reads an SQL query into a DataFrame
- Takes in parse_dates, chunksize
- May support DBAPI connections depending on type

pd.read_sql

- Read SQL query or database table into a DataFrame
- More for convenience, compatibility for previous methods
- Takes in parse_dates, columns, chunksize

Construct Level 3

Now, I will be using SQLite to access the database for a dataset called **Simple Folk**.

Link to database: http://2016.padjo.org/files/data/starterpack/simplefolks.sqlite

```python
import sqlite3
import pandas as pd
con = sqlite3.connect('simplefolks.sqlite')
cur = con.cursor()
rows = cur.fetchall()
```

```python
# DataFrame with the age, sex, and name of individuals
people_df = pd.read_sql_query("SELECT age, sex, name FROM people", con)
# DataFrame with pets, pet owners, and pet names
pet_df = pd.read_sql_query("SELECT name, owner_name FROM pets ORDER BY name", con)
```

```python
# Viewing DataFrames
people_df.head()
```

```python
# Viewing DataFrames
pet_df.head()
```

Now, let's say we wanted to list all of the 30 year old and older men in the people table. We can do this by implementing the query below.

```python
query_1 = pd.read_sql_query("SELECT * FROM people WHERE sex = 'M' AND age >= 30", con)
query_1
```

Or, with the pet table we can find the pets name and type that are not dogs or cats. Personally, the bird named Harambe is my favorite.

```python
query_2 = pd.read_sql_query("SELECT name, type FROM pets WHERE type != 'cat' and TYPE != 'dog'",
con)
query_2
```

Luckily for us, SQLite databases and their respective tables usually come in a very easy-to-read format. In other cases where the data is not as easy to read in, we need to be able to clean and prepare it.

Prepare Level 2

Apply data reshaping, cleaning, and filtering as directed.

```python
# First, I'll load the xls file in and take a look at it uncleaned.
unclean_table = pd.read_excel("tabn039.xls")
unclean_table.head()
```

Obviously, this table needs some major cleaning. Although I am only showing the head of the table above, there are actually plenty of rows towards the end (rows 42-45) that are filled with NaN values. Our first task will be to **drop and rename** columns and rows.

```python
clean_table = pd.read_excel("tabn039.xls", header = 1, index_col=0)
clean_table.head()
```

```python
# This will get rid of leadings rows, as well as NaN rows towards the bottom
clean_table = clean_table.iloc[2:41]
clean_table
```

```python
# Changing row and column names to appropriate names
clean_table.rename(str.lower, axis='index', inplace=True)
clean_table.rename(str.lower, axis='columns', inplace=True)
# Deleting whitespaces and trailing periods
clean_table.index = clean_table.index.str.replace('.','')
clean_table.index = clean_table.index.str.replace(' ','_')
clean_table.index = clean_table.index.str.lstrip('_')
clean_table.index = clean_table.index.str.rstrip('_')
clean_table.columns = clean_table.columns.str.replace(' ','_')
```

```python
clean_table
```

Now, our rows and columns look pretty good. However, we have a couple of different tables combined into one table (cleaned_table). Below, I will attempt to split these into **2 different tables**.

This table shows the **Enrollment in Schools, in Thousands**.

```python
# First table
df1 = clean_table.iloc[:19]
df1.style.set_caption("Enrollment in Schools, in Thousands")
df1.index.name = None
df1
```

This table shows the **Percent Distribution in Enrollment in Schools**.

```python
# Second table
df2 = clean_table.iloc[20:]
df2.index.name = None
df2
```

Submission 1 - Chapter 2

Process Level 2

This blog post is about the podcast **Scientific Reasoning for Practical Data Science with Andrew Gelmen** to earn **process level 2**. The link to the podcast, if you would like to list, is here.

Today I will be discussing the **Philospohy of Data Science Podcast**. This episode focuses on the practical uses of data science and scientific reasoning, with the overall goal of making key ideas of these topics easier to understand. This episode also has a special guest, **Andrew Gelmen**, who is a Professor of Statistics and Political Science at Columbia University in New York. He has worked on numeruous amounts of projects in applied data science as well as other applications in statistics. Recently, he has several books for linear regression and classification as well

In fact, Andrew Gelmen actually ran his own blog with another researcher in order for them to accurately convey their ideas and findings of their research to one another. This blog also featured a *wiki* section in which other people could get involved and make comments or their own findings on the research. Sadly, the wiki portion of the blog got compromised so they shut down the wiki section altogether. However, the original blog is still live and is located at the following [link](#).

The podcast then goes on to give the listeners/viewers an idea of different classification models, but in simpilier terms. They talk about different modeling tools/classifers such as bayesian modeling, deductive falsification, and so on.

However, a key takeaway from this episode is **why should data scientists care about the philosophy of science?** An important thing about both statistics and data science is that there is no *single way* to do things. The podcast attributes this to the likes of different artisits painting the same model, or different musicians playing the same piece. Although the core concepts are the same, many people have different ways of interpreting data, or what to model or not model.

There are many different kinds of **statistical ideologies**, as the podcast goes into. These kinds of ideologies implant different ideas into people's heads. One such idea could be that *if someone has a randomized controlled experiment with statistical significance, one person could view it differently than the other*. Such as, someone very keen on null hypothesis testing who overvalues results from noisy experiements would view it differently than someone who justifies overally noisy results with outliers.

When considering youir own statistical ideology, there are multiple things to consider:

- You must understand the assumptions you want to make about the data, and why you are making those assumptions
- You must be able to make sense of your own philosophy
- You must be able to connect all of your findings to your larger goals
- You must be able to connect these statements to the world at large to avoid making mistakes by having a narrow mindset or tunnel vision

A lot of these points will help statisticians and data scientists to absolve bias from their data (for the most part) and to also see how their own philosophies and ideologies shape how they interpret their data. Doing this will allow data scientists of all different applied fields (statistics, machine learning, artficial intelligence) to have a greater understanding of data anomalies, and their own **statistical reasoning**.

Submission 1 - Chapter 3

In this submission, I will be attempting to earn **summarize level 3** and **visualize level 3**.

```
### Some code here
```

Submission 1 - Chapter 4

For this section of my portfolio, I will be attempting to earn **summarize level 3, visualize level 3, and prepare level 2**. In this first section, I will be correcting **Assignment 4** as well as adding additional graphs and plots to summarize and visualize the data.

Prepare Level 2

For Assignment 4, I was a bit wary of cleaning and preparing data as it was a new concept for me. As I showed briefly in *submission_1*, I have gained some knowledge when dealing with cleaning and preparing data. Similarly, I will be using data from the same database as Assignment 4, located [here](#).

This table shows the different percentages of college students (16-24 years old) who were employed, how many hours they worked a week, and their level of institution.

```python
# Imports
import pandas as pd
```

```python
# First, let's take a look at what we are dealing with
enroll_uc_table = pd.read_excel("tabn503.20.xls")
enroll_uc_table.head(10)
```

```python
enroll_uc_table.tail(10)
```

As we can see from just the table above, the data is quite messy. There are multiply NaN values, headers in the wrong place, as well as trailing characters. Through this submission, I will correct these errors.

```
enroll_clean = pd.read_excel('tabn503.20.xls', skipfooter=7, header=list(range(3)), skiprows=2)
enroll_clean.set_index('Control and level of institution and year',inplace=True)
# enroll_clean = enroll_clean.iloc[2:,]
enroll_clean.dropna('index',inplace=True)
```

```
enroll_clean.head(5)
```

```
# Map years by removing trailing periods
year_mapper = {yr: yr[0].replace('.','').strip() for yr in enroll_clean.index}
year_mapper
```

```
enroll_clean.rename(year_mapper, inplace=True)
enroll_clean.head(10)
```

In this table, we also have to take into account the different levels of students. In this table, we have **4 levels of institutions**. The first is the **total**, the second is **public 4-year institutions**, **private 4-year institutions**, and finally **public 2-year institutions**.

To deal with all of these, I will separate these four different levels into four different tables, as well as having different tables for full-time and part-time students.

```
# Total, all institutions table
total_df = enroll_clean.iloc[:32]
```

```
# Public 4-year institutions table
public_four_df = enroll_clean.iloc[32:43]
```

```
# Private 4-year institutions
priv_4_df = enroll_clean.iloc[43:49]
```

```
# Public 2-year institutions
public_2_df = enroll_clean.iloc[49:]
```

Now, I'm going to split the **total** table into two different dataframes. The first one will be for **full-time students**, while the second one will be for **part-time students**. This will make the overall table much cleaner.

```
# Making full-time dataframe
full_cols = [col for col in total_df.columns if ('Unnamed' in col[2]) or ('Part-time students' in col[0])
             or ('Less than 20 hours .1' in col[2]) or ('Less than 20 hours .2' in col[2])
             or ('20 to 34 hours .1' in col[2]) or ('20 to 34 hours .2' in col[2])
             or ('35 or more hours .1' in col[2]) or ('35 or more hours .2' in col[2])]
full_time = total_df.drop(full_cols, axis=1)
```

```
# Making part-time dataframe
part_cols = [col for col in total_df.columns if ('Unnamed' in col[2]) or ('Full-time students' in col[0])
             or ('Less than   20 hours .1' in col[2]) or ('Less than   20 hours .2' in col[2])
             or ('20 to 34 hours .1' in col[2]) or ('20 to 34 hours .2' in col[2])
             or ('35 or more hours .1' in col[2]) or ('35 or more hours .2' in col[2])]
part_time = total_df.drop(part_cols, axis=1)
```

```
# Viewing dataframes
full_time.head()
```

```
part_time.head()
```

Now, we will apply more formatting to the above dataframes to make them cleaner and more readable, also following column name guidelines.

```
# Full-time students dataframe
full_time = full_time.unstack().reset_index().drop('level_1',axis=1)
full_time.rename(columns = {'level_0':'enrollment_status',
                            'level_2':'hours_worked',
                            'Control and level of institution and year':'year',
                            0:'total'}, inplace=True)

full_time.head()
```

```
# Part-time students dataframe
part_time = part_time.unstack().reset_index().drop('level_1', axis=1)
part_time.rename(columns = {'level_0':'enrollment_status',
                            'level_2':'hours_worked',
                            'Control and level of institution and year':'year',
                            0:'total'}, inplace=True)

part_time.head()
```

Now the above tables are much cleaner. To visualize them better, I will develop some plots that will visualize the hours worked in total across full-time and part-time students.

```
# First, concat the two tables together to form the total table
enroll_df = pd.concat([full_time, part_time])
enroll_df
```

Visualize Level 3

```
# Plotting full-time students from total amount
import seaborn as sns
full_g = sns.relplot(data=full_time, x='year', y='total', col='enrollment_status',
hue='hours_worked')
full_g.set_xticklabels(rotation=50)
```

```
# Plotting part-time students from total amount
part_g = sns.relplot(data=part_time, x='year', y='total', col='enrollment_status',
hue='hours_worked')
part_g.set_xticklabels(rotation=50)
```

Summarize Level 3

Now, I'll compute some brief summary statistics regarding the data used in this submission.

```
# Calculating brief summary statistics for full-time students, a subset of the total amount of
students
full_time.describe()
```

```
# Calculating various statistics for part-time students, a subset from the total amount of students
part_time.agg({'total' : ['sum', 'min', 'max']})
```

The above summary statistic shows the **sum of the total hours** worked for part-time students, the **minimum number of hours** worked by part-time students, and the **maximum number of hours** worked by a part-time student.