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.

```
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.

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

The example below will give a grade of a B.

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

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

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

Submission 1

For Python Level 3, Prepare Level 2, and Access Level 3

In this section of my portfolio I will be demonstrating skills for the above list. For more demonstration on **Prepare**, please look at *submission_4*.

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

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

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

```
# 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)
```

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

```
# 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.

```
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.

```
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.

```
# 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.

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

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

```
# 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(' ','_')
```

```
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**.

```
# 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**.

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

Submission 2

Process Level 2

In this section of my portfolio, I will be writing a blog-style post about this podcast.

Blog Post

Submission 3

For Summarize Level 3 and Visualize Level 3

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