PS2

September 18, 2025

1 Problem set 2

1.1 Name: [Yawen Tan]

1.2 Link to your PS2 github repo: [https://github.com/IsabellaTan/Brown-DATA1030-HW2#]

1.3 Problem 0

-2 points for every missing green OK sign. If you don't run the cell below, that's -16 points.

Make sure you are in the DATA1030 environment.

[1]:
```
from __future__ import print_function
from packaging.version import parse as Version
from platform import python_version

OK = '\x1b[42m[ OK ]\x1b[0m'
FAIL = "\x1b[41m[FAIL]\x1b[0m"

try:
    import importlib
except ImportError:
    print(FAIL, "Python version 3.12.10 is required,"
                " but %s is installed." % sys.version)

def import_version(pkg, min_ver, fail_msg=""):
    mod = None
    try:
        mod = importlib.import_module(pkg)
        if pkg in {'PIL'}:
            ver = mod.VERSION
        else:
            ver = mod.__version__
        if Version(ver) == Version(min_ver):
            print(OK, "%s version %s is installed."
                  % (lib, min_ver))
        else:
            print(FAIL, "%s version %s is required, but %s installed."
                  % (lib, min_ver, ver))
```

```
    except ImportError:
        print(FAIL, '%s not installed. %s' % (pkg, fail_msg))
    return mod


# first check the python version
pyversion = Version(python_version())

if pyversion >= Version("3.12.10"):
    print(OK, "Python version is %s" % pyversion)
elif pyversion < Version("3.12.10"):
    print(FAIL, "Python version 3.12.10 is required,"
                " but %s is installed." % pyversion)
else:
    print(FAIL, "Unknown Python version: %s" % pyversion)


print()
requirements = {'numpy': "2.2.5", 'matplotlib': "3.10.1",'sklearn': "1.6.1",
                'pandas': "2.2.3",'xgboost': "3.0.0", 'shap': "0.47.2",
                'polars': "1.27.1", 'seaborn': "0.13.2"}

# now the dependencies
for lib, required_version in list(requirements.items()):
    import_version(lib, required_version)
```

```
[ OK ] Python version is 3.12.10
```

```
[ OK ] numpy version 2.2.5 is installed.
[ OK ] matplotlib version 3.10.1 is installed.
[ OK ] sklearn version 1.6.1 is installed.
[ OK ] pandas version 2.2.3 is installed.
[ OK ] xgboost version 3.0.0 is installed.
[ OK ] shap version 0.47.2 is installed.
[ OK ] polars version 1.27.1 is installed.
[ OK ] seaborn version 0.13.2 is installed.
```

1.4 Problem 1 - data collection

Which Rhode Island school has the largest undergraduate student population? And graduate student population? You will collect and analyze data to answer these two questions using the College Scoreboard API of the U.S. Department of Education. An API (application point interface) is a mechanism which allows two software components to communicate with each other using a set of definitions and protocols. The two software components in this case are your jupyter notebook running python and the College Scoreboard server.

APIs are a popular way to share and modify data in an automated, secure, and cost-efficient way. Read more about APIs here.

The documentation of the College Scoreboard API is available here, read it carefully. It is a REST API which means that we will perform operations using standard HTTP methods. We want to know the name of each RI school, in which city/town it is located, the zipcode, and how many undergrad and gradute students they have based on the most recent data.

You will use python packages like `requests`, `dotenv`, and `pandas` or `polars` to collect and save the data. - The `requests` package is how you will query the API. We will submit only one query to the College Scoreboard API which could in principle be done in a browser. However I want you write python code because often you need to make a large number of API requests which needs to be automated with code. Read more about it here. - One way APIs achieve security is to limit access to authorized users only. Authorized users have an API_KEY which is a secret key specific to each user. This API_KEY needs to be provided when you make a request to the server, it will be part of the HTTP URL. This is how the server knows who makes the request and what level of access the user has. While most users can have read access, only a limited number of users usually have access to modify a dataset. Therefore the API_KEY needs to be kep secret. **Your API_KEY should NEVER be directly copy-pasted into your notebook or pushed to any github repository!** If you do so, that's a security risk even if the repository is private. Also, you will lose points. A pretty popular way to share secrets like the API_KEY with your notebook is done by using the `dotenv` package. Read more about it here. - The API request will be returned in a json format. You will use `pandas` or `polars` to convert the json output to a dataframe, and save it as a csv file.

If some of these terminologies or concepts don't make sense right now, don't worry about it. Read the linked documentations, follow the steps as outlined below, and post on the course forum or come to office hours if you have questions.

[3]:
```
# resolve any error messages you might encounter as you work through the steps

# import pandas/polars, requests, and dotenv packages here
from dotenv import load_dotenv
import pandas as pd
import requests
import os

### Setup Steps (do these first):
# 1. Go to the college scoreboard documentation (linked above) and request an␣
 ↪API_KEY
# 2. Create a `.env` file in the same folder as this notebook
# 3. Add your API key to the `.env` file: API_KEY=your_key_here
# 4. Add `.env` to your `.gitignore` file
# 5. Test that `load_dotenv()` works before proceeding
# If you encounter any issues, check the documentation for most common errors.
load_dotenv()

# read the college scoreboard documentation carefully.
# collect info on all Rhode Island schools. We want to know the name of each␣
 ↪school,
```

```
# in which city/town it is located, the zipcode, and how many undergrad and␣
 ↪gradute students they have based on the most recent data
# collect below the fields necessary to collect the data as a python dictionary.
# add your API_KEY to this dictionary (feel free to look up how to access␣
 ↪environment variables, if you're confused)
API_KEY = os.getenv("API_KEY")
request_params = {
    "api_key": API_KEY,
    "school.state": "RI",
    "fields": "school.name,school.city,school.zip,school.size,latest.student.
 ↪grad_students,latest.student.size",
    "per_page": 200
}
# add the base URL below.
api_url = 'https://api.data.gov/ed/collegescorecard/v1/schools'


# use the request_params and the api_url to make a get request using the␣
 ↪requests package
# save the response
response = requests.get(api_url, params=request_params)


# print the response below while you are developing the code
# comment out the print statement once you are certain the code works as␣
 ↪intended
# this is for debugging purposes only because requests are finicky things.
# one missed character in the URL, one small typo in one of the parameters,
# one small error in your code, and the request will return an error code.
# therefore it is important to carefully read the manuals
# and follow them to the letter

#print("Response code:", response.status_code)
#print("Sample:", response.text[:300])

# write an if-else statement
# if the response code is 200 (successful request), save the result as a json␣
 ↪object
# else print our the response code and the error message and raise a valueError
# note: some fatal errors (e.g. if the base URL is totally wrong) will come up␣
 ↪in the original get call. Here, we only care about calls that return a␣
 ↪unsuccessful request.

# successful request
if response.status_code == 200:
    result = response.json()
```

```
# unsuccessful request
else:
    print("Response code ", response.status_code)
    print("First 300 characters of error message:", response.text[:300])
    raise ValueError("API request failed.")

# print out the json result below.
# what's the total number of schools returned? how many responses were actually␣
 ↪returned?
# check the manuals again to fix the discrepancy.

# Print the total number of schools returned in the metadata
total_schools = result.get("metadata").get("total")
print("Total schools (metadata.total):", total_schools)
# Print the number of schools actually returned in this response
returned_schools = len(result["results"])
print("Number of schools actually returned:", returned_schools)


# save the json results into a pandas dataframe
uni_df = pd.DataFrame(result["results"])

# save uni_df into a csv file, save the file in the same folder where this␣
 ↪notebook is located
uni_df.to_csv("rhode_island_schools.csv", index=False)

# You are done with problem 1!
```

```
Total schools (metadata.total): 23
Number of schools actually returned: 23
```

1.5 Problem 2 - EDA

As mentioned in class, you should approach a new dataset with a healthy set of skepticism. You will study the dataset you collected in problem 1.

1.5.1 Problem 2a

Solve the tasks outlined in the cell below.

[4]:
```
# import the necessary packages
import pandas as pd
# Read in the csv file and print out the columns.
file = pd.read_csv("rhode_island_schools.csv")
# Do you have all the columns we need? If you don't, go back to problem 1 to␣
 ↪resolve the issue.
# Create a list of required columns
```

```
required_columns = ['school.name','school.city','school.zip','latest.student.
 ↪grad_students','latest.student.size']
# Create a empty list to hold missing columns
missing_columns = []
# Check for missing columns
for col in required_columns:
    if col not in file.columns:
        missing_columns.append(col)
# Print missing columns if any
if len(missing_columns)!=0:
    print("Missing columns:", missing_columns)
else:
    print("All required columns are present.")

# Do you have extra columns we don't need? Drop the unnecessary columns.
# keep only the required columns
file = file[required_columns]

# Check for duplicate rows in the dataset. If there are duplicate rows, drop␣
 ↪all but one.
duplicate_count = file.duplicated().sum()
print("Number of duplicate rows:", duplicate_count)
if duplicate_count > 0:
    file = file.drop_duplicates()
```

```
All required columns are present.
Number of duplicate rows: 0
```

1.5.2 Problem 2b

You will study the validity of the dataset, look for typos/errors, study the missing values, and finally answer our two original questions.

[37]:
```
# we have some numerical features like the zipcode and the number of students
# what sort of impossible or incorrect values could you see in these columns?
# use critical thinking skills
# test at least three columns and write at least one test per column to verify␣
 ↪the data validity
# here is the format of the test:
# if condition == True:
#     raise ValueError('error message')
# consider the number of undergraduate/graduate student features.
# what values would be technically possible/plausible but unrealistic?
# write a test


# Check if zipcode is NA
if (file['school.zip'].isnull()).any():
```

```
    raise ValueError("Error: NA zipcode found!")

# Check if 'latest.student.size'(undergraduate student size) is numeric
if not pd.api.types.is_numeric_dtype(file['latest.student.size']):
    raise ValueError("Error: Undergraduate student size column is not numeric!")

# Check if 'latest.student.grad_students'(graduate student size) is numeric
if not pd.api.types.is_numeric_dtype(file['latest.student.grad_students']):
    raise ValueError("Error: Graduate student size column is not numeric!")

# Check if 'latest.student.size'(undergraduate student size) is negative
if (file['latest.student.size'] < 0).any():
    raise ValueError("Error: Invalid data found!")

# Check if 'latest.student.grad_students'(graduate student size) is negarive
if (file['latest.student.grad_students'] < 0).any():
    raise ValueError("Error: Negative graduate student size found!")

# Check if 'latest.student.size'(undergraduate student size) is float if not␣
 ↪null
if ((file['latest.student.size'] % 1 != 0) & (file['latest.student.size'].
 ↪notnull())).any():
    raise ValueError("Error: Undergraduate student size cannot be float!")

# Check if 'latest.student.grad_students'(graduate student size) is float if␣
 ↪not null
if ((file['latest.student.grad_students'] % 1 != 0) & (file['latest.student.
 ↪grad_students'].notnull())).any():
    raise ValueError("Error: Graduate student size cannot be float!")

# are there missing values in the dataset?
# in which columns?
# what fraction of the points contain missing values?
# why could the values be missing?
# write a short description on possible reasons.

# Summarize missing values
missing_counts = file.isnull().sum()
print("Number of missing values per column:\n", missing_counts)
# Summarize missing value fractions
missing_fraction = file.isnull().mean()
print("Fraction of missing values per column:\n", missing_fraction)
# Create a empty list to add columns with missing values
columns_with_missing = []
for col in missing_counts.index:
    if missing_counts[col] > 0:
        columns_with_missing.append(col)
```

```
print("Columns with missing values:", columns_with_missing)

print("The possible reasons for missing values could be: \n Some schools may␣
 ↪not accept undergraduate or graduate students, which results in the number␣
 ↪of undergraduate or graduate students being NA.")


# finally, answer the original questions we set out to investigate
# print out which RI school has the largest undergraduate student population.
# print out which RI school has the largest graduate student population.
# the dataset is small enough that you could answer these questions just by␣
 ↪looking at the csv file but that's not accepted.
# use code to determine the answer

# Find the index of the school with the maximum undergraduate student size
max_undergrad_index = file['latest.student.size'].idxmax()
# Retrieve the school name and undergraduate student size using the index
max_undergrad_school = file.loc[max_undergrad_index, 'school.name']
max_undergrad_count = file.loc[max_undergrad_index, 'latest.student.size']
print(f"The RI school with the largest undergraduate population is␣
 ↪{max_undergrad_school} with {max_undergrad_count} students.")

# Find the index of the school with the maximum graduate student size
max_grad_index = file['latest.student.grad_students'].idxmax()
# Retrieve the school name and graduate student size using the index
max_grad_school = file.loc[max_grad_index, 'school.name']
max_grad_count = file.loc[max_grad_index, 'latest.student.grad_students']
print(f"The RI school with the largest graduate population is {max_grad_school}␣
 ↪with {max_grad_count} students.")
```

```
Number of missing values per column:
 school.name                      0
school.city                       0
school.zip                        0
latest.student.grad_students     11
latest.student.size               3
dtype: int64
Fraction of missing values per column:
 school.name                      0.000000
school.city                      0.000000
school.zip                       0.000000
latest.student.grad_students     0.478261
latest.student.size              0.130435
dtype: float64
Columns with missing values: ['latest.student.grad_students',
'latest.student.size']
The possible reasons for missing values could be:
```

```
 Some schools may not accept undergraduate or graduate students, which results
in the number of undergraduate or graduate students being NA.
The RI school with the largest undergraduate population is University of Rhode
Island with 13822.0 students.
The RI school with the largest graduate population is Brown University with
3775.0 students.
```