General guidelines

- This assignment is tentatively due on **Tuesday of week 10,** with peer grades due during week 11. Exact deadlines are **as posted on gradescope.** Note the specific deadline since everyone will need adequate time to perform peer grading during week 11.
- **Given that assignments must be distributed to peer graders, late submissions will not be accepted (or accepted for at most partial credit); there will be a late window though this is for students with an applicable OSD accommodation and not generally available to others.**
- Peer assignments will be available once the above late window closes (plus 12-24 hours for me to compile submissions and run various peer assignment scripts)
- Assignments can be completed in groups of up to 4, though may also be completed individually
- The assignment will be peer-graded. You are welcome to "blind" your submission (to the extent possible) though you are not required to. *As such, there will be no difference in grading based on the group size.*
- You are required to upload:
    1. A **jupyter notebook**, exported as html. It should be "clean", documented, and readable as-is; peer graders should not be expected to execute your code
    2. A **~20-minute presentation**, uploaded as a video file via a google drive link
    3. A peer grading report (due ~1 week later)
- **The above components are worth 29 marks.** The notebook and presentation aren't graded separately, rather, *peer graders should be expected to watch your presentation while following along with your code.* The exact format is up to you, though I recommend a combination of slides and code walkthrough. The specific graded components are described below.
- Examples of datasets and projects that may be of interest in this assignment will be discussed in the lectures, though you may use any dataset you wish (including the ones we used for Assignment 1). For a selection of datasets that I frequently use, see https://cseweb.ucsd.edu/~jmcauley/datasets.html

Presentation

Your presentation should focus on the following five sections:

1. Identify the predictive task you will study.
    - Describe how you will evaluate your model at this predictive task
    - What relevant baselines can be used for comparison
    - How you will assess the validity of your model's predictions?

   Make sure to select a task and models that are relevant to the course content; if you want to try out models you've seen in other classes that's fine, but you should still implement models from this class as baselines / comparison points.

2. Exploratory analysis, data collection, pre-processing, and discussion:
    - **Context:** Where does your dataset come from? What is it for, how was it collected, etc.?
    - **Discussion:** Report how you processed the data (or how it was already processed);
    - **Code:** Support your analysis with tables, plots, statistics, etc.
3. Modeling:
    - **Context:** How do you formulate your task as an ML problem, e.g. what are the inputs, outputs, and what is being optimized? What models are appropriate for the task?
    - **Discussion:** Discuss the advantages and disadvantages of different modeling approaches (complexity, efficiency, challenges in implementation, etc.)
    - **Code:** Walk through your code, explaining architectural choices and any implementation details.
4. Evaluation:
    - **Context:** How should your task be evaluated? Can you justify why your particular metrics are more appropriate than others?
    - **Discussion:** What are some baselines (trivial or otherwise) for your task? How do you demonstrate that your method is better than these methods?
    - **Code:** Walk through the implementation of your evaluation protocol, and support your evaluation with tables, plots, statistics, etc.
5. Discussion of related work:
    - How has this dataset (or similar datasets) been used before?
    - How has prior work approached the same (or similar) tasks?
    - How do your results match or differ from what has been reported in related work?
    - (you can put this section at the beginning if you'd prefer)

You can reorder sections ***as long as your organization is very obvious to graders.*** Try not to go overlength by more than ~10%: I would suggest that graders allocate 20-30 minutes per assignment for grading, so if you go too long you risk having graders watch your presentation at 1.5x speed or missing out on the end.

Rubric and grading

The assignment is worth 29 marks, broken down as 5 marks per section plus 4 marks for peer grading.

Graders should grade each of the five parts *roughly* according to the following rubric:

0: This component was not covered
1: This component was covered, but appears to contain errors, or cannot be easily understood, or the presentation does not seem to match the code, ***or the presentation seems to be automatically generated***

2: This component was covered, but only superficially, i.e., it has some of the right components, and appears correct, but is missing key elements or comparisons that would be expected from a complete analysis
3: This submission is more-or-less minimally acceptable: e.g. a minimal set of essential elements are covered, though maybe not discussed in detail
4: The code and presentation seems feature-complete: all the results "make sense", or any negative results are adequately explained; the presentation explains the tradeoffs between different metrics and goals, is clear and easy to follow, etc. ***This should probably be the most common grade.***
5: The code and presentation go above-and-beyond in some way, e.g. it might be particularly thorough, try an interesting or new approach, or shows excellence in some other way

Submission instructions

Unfortunately, (a) gradescope isn't set up for peer grading; (b) gradescope doesn't accept large files; and (c) google drive files can't be shared anonymously. So submission is a little complicated... It will work as follows:

Step 1: Submit the following files

- **workbook.html:** this should be a jupyter notebook, exported as html. *You should be able to open it in a browser.*
- **video_url.txt:** should contain a single line, which is a *path to a shared mp4 file on google drive* **or** a *youtube link*. It should probably look like:
  https://drive.google.com/file/d/FILE_ID/view?usp=drive_link
  or https://www.youtube.com/watch?v=_QgIj7eNYAM

Your video url should be generated on google drive by enabling "anyone with the link" sharing and copying the link. If you use some other format, please convert to mp4 (which can be generated by recording a zoom session or similar). Please do your best to make sure your video file is watchable on "normal" hardware (e.g. in the past some people have used a video codec that is not generally available).

The autograder will check that (a) your file is downloadable; (b) greater than 1mb; (c) is of type video/mp4; (d) that your workbook.html file is a html file; and (e) that you have submitted files for two tasks.

Note that your google drive link ***won't be shared:*** it is not possible to fully anonymize shared files, so I will download your file and put it in a shared folder. If you use a youtube link, it will be shared as-is.

Step 2: 1-2 days later, I will compile all submissions, make peer assignments, and distribute peer assignments via another gradescope assignment (details to follow). Please be patient as it is quite a process!

How your submission is verified

In case you'd like to check at home, your submission is verified using code similar to the following:

```
import unittest
import os
import magic
import requests

def test_workbook(path):
    f = open("workbook.html", 'r')
    t = f.read()
    if not t.startswith('<!DOCTYPE html>'):
        print("Your workbook submission is not a html file (does not start with
'<!DOCTYPE html>'")
        print("Please save your work as html (e.g. do not just change the extension)")
        raise Exception
    f.close()

def test_video(url):
    if 'youtube' in url or 'youtu.be' in url:
        print("Looks like your video is a youtube link (" + url + ")")
        print("No further tests will be run on your file: please make sure it is
publicly viewable")
        return
    FILE_ID = None
    if "id=" in url:
        FILE_ID = url.split("id=")[1].split('/')[0]
    elif '/d/' in url:
        FILE_ID = url.split("/d/")[1].split('/')[0]
    if FILE_ID == None:
        print("Couldn't determine file ID; expected a url of the form
https://drive.google.com/file/d/FILE_ID/view?usp=drive_link or
https://drive.google.com/uc?export=download&id=FILE_ID")
        print('(got """' + url + '""")')
        raise Exception
    print("Looks like your google drive file ID is " + FILE_ID)
    url_fixed = "https://drive.google.com/uc?export=download&id=" + FILE_ID
    r = requests.get(url_fixed, stream=True)
    print("Trying to download from " + url_fixed)
    chunk = next(r.iter_content(chunk_size=1024*1024))
    if len(chunk) < 1024*1024:
        chunk = chunk.decode('utf8')
        if "Virus scan warning" in chunk:
            print("Looks like your file is too large for Google to scan for viruses")
            uuid = chunk.split('uuid" value="')[1].split('"')[0]
            url_fixed = "https://drive.usercontent.google.com/download?id=" + FILE_ID
+ "&export=download&confirm=t&uuid=" + uuid
            print("Trying to download from " + url_fixed)
            r = requests.get(url_fixed, stream=True)
            chunk = next(r.iter_content(chunk_size=1024*1024))
        else:
            print("Looks like your video file is less than 1mb; it is probably not a
video")
            raise Exception
    print("Confirmed that your video file is greater than 1mb; checking file type")
    mime = magic.Magic(mime=True)
    typename = mime.from_buffer(chunk)
    if not "video/mp4" in typename:
        print("Your file is not an mp4 video file (type=" + typename + ")")
        raise Exception
    print("Everything looks okay! On your own, please verify that you can download a
working video using the assignment script in the spec")
    #print("wget -O output.mp4 '" + url_fixed + "'")

class TestFiles(unittest.TestCase):
    @weight(0)
    def test_submitted_files(self):
        """Check submitted files"""
        missing_files = check_submitted_files(['workbook.html', 'video_url.txt'])
        for path in missing_files:
            print('Missing {0}'.format(path))
        self.assertEqual(len(missing_files), 0, 'Missing some required files!')
        print('All required files submitted!')
        test_workbook("workbook.html")
        f = open("video_url.txt", 'r')
        url = f.read().strip()
        test_video(url)
        f.close()
```