Mud card

- **Can we do more examples like the spam baseline question? I was confused with the baseline answer. I want to be able to understand this "neither" way of thinking for more scenarios.**
  - yep, we will have more examples
  - generally speaking, you need to check how frequently each class label occurs in your dataset
  - the fraction of points that belong to the most frequent class is your baseline
  - e.g., if 90% of your email are fine, and 10% are spam, your baseline accuracy is 90%
  - if you were to predict 'no spam' to each email, you would be correct 90% of the time
  - so a classifier is good if it's accuracy is better than the baseline
- **I was wondering what are some of the best practices that should be applied to Exploratory Data Analysis (especially for large datasets)?**
  - we will cover a couple of things during the next two lectures
  - summary stats for each column is a great way to start
  - figures for each column
  - create a correlation coefficient matrix
  - ceheck how strongly each feature correlates with the target variable
- **Is the model the same as the parameter?**
  - nope, models have parameters
- **Will you further explain the different packages that we will be importing and using to create these models?**
  - yes, we cover pandas today and next Tuesday for example
- **The muddiest part of this lecture was how to go about splitting a dataset and how to choose an evaluation metric.**
  - we will spend a week on each of those topics
- **"The sampling, what difference between curly X and X is, and what is S?**
  - curly X is the domain set so all possible values of your features. If a data point is described by let's say d features, it could be a set of d dimensional real numbers
  - X is sampled from curly X
  - S is your training set which contains feature vectors (x_i) and and a true target variable (y_i)
  - Check out chapter 2 of this book for more info
- **How do we define a "large" dataset? How many observations?**
  - there is no clear answer to these types of questions
  - some people would consider more than 10k points large, others might consider a dataset with more than 10m points to be large
- **How does a model determine the Y target variable value on the decision boundary?**
  - it could be random choice, one could return the most frequently occuring label in the training set, there might be other reasonable approaches too
- **Were we expected to understand the pipeline or was this just an overview that we will dive into later?**
  - Yep, it was just an overview and we will dive into each step later
- **If you could explain more on the bias-variance tradeoff that would be great!**
  - we will revisit it a couple of times during the term
- **How did you determine the Cs in the example?**
  - we will cover this later too
  - some hyperparameters need to be uniformly spaced in log, other hyperparameters are linearly spaced
  - we will learn how to decide
- **I didn't quite fully follow the process of how we get the best C score. Do we just feed the C into the classifier, which is part of the packages I assume, and then use that classifier to predict and then check the output?**
  - yes but we tried 13 different values for C
  - we need to decide which of them gave the best model
- **I'm a bit confused about whether the generalization error is calculated from the train set or the test set?**
  - test set, always the test set
  - the only purpose of the test set is to calculate the generalization error
- **The muddiest part was hyper parameters. Could you please elaborate on what they are?**
  - we will spend two weeks on them around mid-October

Exploratory data analysis in python, part 1

The steps

1. Exploratory Data Analysis (EDA): you need to understand your data and verify that it doesn't contain errors

- do as much EDA as you can!

2. Split the data into different sets: most often the sets are train, validation, and test (or holdout)

- practitioners often make errors in this step!
- you can split the data randomly, based on groups, based on time, or any other non-standard way if necessary to answer your ML question

3. Preprocess the data: ML models only work if X and Y are numbers! Some ML models additionally require each feature to have 0 mean and 1 standard deviation (standardized features)

- often the original features you get contain strings (for example a gender feature would contain 'male', 'female', 'non-binary', 'unknown') which needs to transformed into numbers
- often the features are not standardized (e.g., age is between 0 and 100) but it needs to be standardized

4. Choose an evaluation metric: depends on the priorities of the stakeholders

- often requires quite a bit of thinking and ethical considerations

5. Choose one or more ML techniques: it is highly recommended that you try multiple models

- start with simple models like linear or logistic regression
- try also more complex models like nearest neighbors, support vector machines, random forest, etc.

6. Tune the hyperparameters of your ML models (aka cross-validation)

- ML techniques have hyperparameters that you need to optimize to achieve best performance
- for each ML model, decide which parameters to tune and what values to try
- loop through each parameter combination
  - train one model for each parameter combination
  - evaluate how well the model performs on the validation set
- take the parameter combo that gives the best validation score
- evaluate that model on the test set to report how well the model is expected to perform on previously unseen data

7. Interpret your model: black boxes are often not useful

- check if your model uses features that make sense (excellent tool for debugging)
- often model predictions are not enough, you need to be able to explain how the model arrived to a particular prediction (e.g., in health care)

Pandas

- data are often distributed over multiple files/databases (e.g., csv and excel files, sql databases)
- each file/database is read into a pandas dataframe
- you often need to filter dataframes (select specific rows/columns based on index or condition)
- pandas dataframes can be merged and appended

Some notes and advice

- **ALWAYS READ THE HELP OF THE METHODS/FUNCTIONS YOU USE!**
- stackoverflow is your friend, use it! https://stackoverflow.com/
- you can also use generative AI (like github copilot, bard, or chatGPT's code interpreter) to help you fix bugs

Data transformations: pandas data frames

By the end of this lecture, you will be able to

- read in csv, excel, and sql data into a pandas data frame
- filter rows in various ways
- select columns

- merge and append data frames

Data transformations: pandas data frames

By the end of this lecture, you will be able to

- **read in csv, excel, and sql data into a pandas data frame**
- filter rows in various ways
- select columns
- merge and append data frames

In [ ]:
```
# how to read in a database into a dataframe and basic dataframe structure
import pandas as pd

# load data from a csv file
df = pd.read_csv('data/adult_data.csv') # there are also pd.read_excel(), and pd.read_sql()

print(df)
#help(df.head)
#print(df.head(10)) # by default, shows the first five rows but check help(df.head) to specify the number of rows
#print(df.shape) # the shape of your dataframe (number of rows, number of columns)
#print(df.shape[0]) # number of rows
#print(df.shape[1]) # number of columns
```

Packages

A package is a collection of classes and functions.

- a dataframe (pd.DataFrame()) is a pandas class
  - a class is the blueprint of how the data should be organized
  - classes have methods which can perform operations on the data (e.g., .head(), .shape)
- df is an object, an instance of the class.
  - we put data into the class
  - methods are attached to objects
    - you cannot call pd.head(), you can only call df.head()
- read_csv is a function
  - functions are called from the package
  - you cannot call df.read_csv, you can only call pd.read_csv()

DataFrame structure: both rows and columns are indexed!

- index column, no name
  - contains the row names
  - by default, index is a range object from 0 to number of rows - 1
  - any column can be turned into an index, so indices can be non-number, and also non-unique. more on this later.
- columns with column names on top

Always print your dataframe to check if it looks ok!

Most common reasons it might not look ok:

- the first row is not the column name
  - there are rows above the column names that need to be skipped
  - there is no column name but by default, pandas assumes the first row is the column name. as a result, the values of the first row end up as column names.
- character encoding is off
- separator is not comma but some other charachter

In [ ]:
```
# check the help to find the solution
help(pd.read_csv)
```

Exercise 1

How should we read in adult_test.csv properly? Identify and fix the problem.

```
In [ ]:
```

Data transformations: pandas data frames

By the end of this lecture, you will be able to

- read in csv, excel, and sql data into a pandas data frame
- **filter rows in various ways**
- select columns
- merge and append data frames

How to select rows?

1) Integer-based indexing, numpy arrays are indexed the same way.

2) Select rows based on the value of the index column

3) select rows based on column condition

1) Integer-based indexing, numpy arrays are indexed the same way.

```
In [ ]: # df.iloc[] - for more info, see https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#indexing-i
        # iloc is how numpy arrays are indexed (non-standard python indexing)

        # [start:stop:step] -  general indexing format

        # start stop step are optional
        #print(df.iloc[:])
        #print(df.iloc[::])
        #print(df.iloc[::1])

        # select one row - 0-based indexing
        #print(df.iloc[3])

        # indexing from the end of the data frame
        #print(df.iloc[-1])
```

```
In [ ]: # select a slice - stop index not included
        #print(df.iloc[3:7])

        # select every second element of the slice - stop index not included
        #print(df.iloc[3:7:2])

        #print(df.iloc[3:7:-2]) # return empty dataframe
        #print(df.iloc[7:3:-2])#  return rows with indices 7 and 5. 3 is the stop so it is not included

        # can be used to reverse rows
        #print(df.iloc[::-1])

        # here is where indexing gets non-standard python
        # select the 2nd, 5th, and 10th rows
        #print(df.iloc[[1,4,9]]) # such indexing doesn't work with lists but it works with numpy arrays
```

2) Select rows based on the value of the index column

```
In [ ]: # df.loc[] - for more info, see https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#indexing-la

        print(df.index) # the default index when reading in a file is a range index. In this case,
                        # .loc and .iloc works ALMOST the same.
        # one difference:
        #print(df.loc[3:9:2]) # this selects the 4th, 6th, 8th, 10th rows - the stop element is included!

        #help(df.set_index)
```

```
In [ ]: df_index_age = df.set_index('age',drop=False)

        #print(df_index_age.index)
        #print(df_index_age.head())
```

```
print(df_index_age.loc[30].head()) # collect everyone with age 30 - the index is non-unique
```

3) select rows based on column condition

```
In [ ]: # one condition
        #print(df[df['age']==30].head())
        # here is the condition: it's a boolean series - series is basically a dataframe with one column
        #print(df['age']==30)

        # multiple conditions can be combined with & (and) | (or)
        #print(df[(df['age']>30)&(df['age']<35)].head())
        print(df[(df['age']==90)|(df['native-country']==' Hungary')])
```

Exercise 2

How many people in adult_data.csv work at least 60 hours a week and have a doctorate?

```
In [ ]:
```

Data transformations: pandas data frames

By the end of this lecture, you will be able to

- read in csv, excel, and sql data into a pandas data frame
- filter rows in various ways
- **select columns**
- merge and append data frames

```
In [ ]: columns =  df.columns
        #print(columns)

        # select columns by column name
        #print(df[['age','hours-per-week']])
        #print(columns[[1,5,7]])
        #print(df[columns[[1,5,7]]])

        # select columns by index using iloc
        #print(df.iloc[:,3])

        # select columns by index - not standard python indexing
        #print(df.iloc[:,[3,5,6]])

        # select columns by index -  standard python indexing
        print(df.iloc[:,::2])
```

Data transformations: pandas data frames

By the end of this lecture, you will be able to

- read in csv, excel, and sql data into a pandas data frame
- filter rows in various ways
- select columns
- **merge and append data frames**

How to merge dataframes?

Merge - info on data points are distributed in multiple files

```
In [ ]: # We have two datasets from two hospitals

        hospital1 = {'ID':['ID1','ID2','ID3','ID4','ID5','ID6','ID7'],'col1':[5,8,2,6,0,2,5],'col2':['y','j','w','b','a',
        df1 = pd.DataFrame(data=hospital1)
        print(df1)

        hospital2 = {'ID':['ID2','ID5','ID6','ID10','ID11'],'col3':[12,76,34,98,65],'col2':['q','u','e','l','p']}
```

```
df2 = pd.DataFrame(data=hospital2)
print(df2)
```

In [ ]:
```
# we are interested in only patients from hospital1
df_left = df1.merge(df2,how='left',on='ID') # IDs from the left dataframe (df1) are kept
print(df_left)

# we are interested in only patients from hospital2
#df_right = df1.merge(df2,how='right',on='ID') # IDs from the right dataframe (df2) are kept
#print(df_right)

# we are interested in patiens who were in both hospitals
#df_inner = df1.merge(df2,how='inner',on='ID') # merging on IDs present in both dataframes
#print(df_inner)

# we are interested in all patients who visited at least one of the hospitals
#df_outer = df1.merge(df2,how='outer',on='ID')  # merging on IDs present in any dataframe
#print(df_outer)
```

How to append dataframes?

Append - new data comes in over a period of time. E.g., one file per month/quarter/fiscal year etc.

You want to combine these files into one data frame.

In [ ]:
```
df_append = pd.concat([df1,df2]) # note that rows with ID2, ID5, and ID6  are duplicated! Indices are duplicated
print(df_append)

# df_append = pd.concat([df1,df2],ignore_index=True) # note that rows with ID2, ID5, and ID6  are duplicated!
# print(df_append)

# d3 = {'ID':['ID23','ID94','ID56','ID17'],'col1':['rt','h','st','ne'],'col2':[23,86,23,78]}
# df3 = pd.DataFrame(data=d3)
# print(df3)

# df_append = pd.concat([df1,df2,df3],ignore_index=True) # multiple dataframes can be appended
# print(df_append)
```

Exercise 3

In [ ]:
```
raw_data_1 = {
        'subject_id': ['1', '2', '3', '4', '5'],
        'first_name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
        'last_name': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']}

raw_data_2 = {
        'subject_id': ['6', '7', '8', '9', '10'],
        'first_name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
        'last_name': ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']}

raw_data_3 = {
        'subject_id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
        'test_id': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]}

# Create three data frames from raw_data_1, 2, and 3.
# Append the first two data frames and assign it to df_append.
# Merge the third data frame with df_append such that only subject_ids from df_append are present.
# Assign the new data frame to df_merge.
# How many rows and columns do we have in df_merge?
```

Always check that the resulting dataframe is what you wanted to end up with!

- small toy datasets are ideal to test your code.

If you need to do a more complicated dataframe operation, check out pd.concat()!

We will learn how to add/delete/modify columns later when we learn about feature engineering.

By now, you are able to

- read in csv, excel, and sql data into a pandas data frame
- filter rows in various ways

- select columns
- merge and append data frames

Mud card

```
In [ ]:
```