Mudcard

- **loc and iloc :)**
  - we will go through this again today
- **should we just use pandas in this class or would using both/polars benefit us more in and out of the classroom?**
  - I'd say learn both, that's the most benefitial for you out of classroom.
  - As far as problem sets and quizzes are concerned, it's fine to use one of the packages.
  - I'll mostly use pandas going forward.
- **What are some ways to get more familiar with working with python?**
  - easy leetcode problems are a good start to learn standard python (no packages)
  - you will practice how to work with popular data science packages well enough during class and the problem sets I think
- **need to review basic python functions, like skiprows**
  - skiprows is an argument of the pd.read_csv function :)

Lecture 3

Working with data (step 0) continued

Learning objectives

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

- filter columns
- merge and append data frames
- modify a dataframe

Pandas and Polars

- 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 - last lecture
- you often need to filter dataframes by selecting specific rows - last lecture
- you often need to filter dataframes by selecting specific columns - today
- multiple dataframes need to be merged and appended - today
- dataframes sometimes need to be modified - today

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 to help you fix bugs
  - Gemini is supported by Brown OIT (see here https://go.brown.edu/gemini)
- here is an excellet review of the syntax differences between pandas and polars

Learning objectives

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

- **filter columns**
- merge and append data frames
- modify a dataframe

In [1]:

```
import pandas as pd
df_pd = pd.read_csv('../data/adult_data.csv')

columns =  df_pd.columns
print(columns)

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

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

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

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

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'gross-income'],
      dtype='object')
```

In [2]:

```
import polars as pl
df_pl = pl.read_csv('../data/adult_data.csv')

columns =  df_pl.columns
print(columns)

# select columns by column name
#print(df_pl['age','hours-per-week'])
```

```
#print(df_pl.select(['age','hours-per-week'])) # use .select if you know the
#print(columns[1:4]) # indices must be integers or slices
#print(df_pl[columns[1:4]])

# select columns by index, polars has no .iloc
#print(df_pl[:,3])

# select columns by index - not standard python indexing but it works
#print(df_pl[:,[3,5,6]])

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

```
['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-statu
s', 'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-lo
ss', 'hours-per-week', 'native-country', 'gross-income']
```

Learning objectives

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

- filter columns
- **merge and append data frames**
- modify a dataframe

How to merge dataframes in Pandas?

Merge - info on data points are distributed in multiple files

In [3]:

```
# We have two datasets from two hospitals

hospital1 = {'ID':['ID1','ID2','ID3','ID4','ID5','ID6','ID7'],'col1':[5,8,2,
df1 = pd.DataFrame(data=hospital1)
print(df1)

hospital2 = {'ID':['ID2','ID5','ID6','ID10','ID11'],'col3':[12,76,34,98,65],
df2 = pd.DataFrame(data=hospital2)
print(df2)
```

```
    ID  col1 col2
0  ID1     5    y
1  ID2     8    j
2  ID3     2    w
3  ID4     6    b
4  ID5     0    a
5  ID6     2    b
6  ID7     5    t
     ID  col3 col2
0   ID2    12    q
1   ID5    76    u
2   ID6    34    e
3  ID10    98    l
4  ID11    65    p
```

In [4]:

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

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

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

How to append dataframes in pandas?

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 [5]:

```
df_append = pd.concat([df1,df2]) # note that rows with ID2, ID5, and ID6  ar
print(df_append)

#df_append = pd.concat([df1,df2],ignore_index=True) # note that rows with ID
#print(df_append)

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

# df_append = pd.concat([df1,df2,df3],ignore_index=True) # multiple datafram
# print(df_append)
```

```
     ID  col1 col2  col3
0   ID1   5.0    y   NaN
1   ID2   8.0    j   NaN
2   ID3   2.0    w   NaN
3   ID4   6.0    b   NaN
4   ID5   0.0    a   NaN
5   ID6   2.0    b   NaN
6   ID7   5.0    t   NaN
0   ID2   NaN    q  12.0
1   ID5   NaN    u  76.0
2   ID6   NaN    e  34.0
3  ID10   NaN    l  98.0
4  ID11   NaN    p  65.0
```

How to merge/join dataframes in Polars?

In [6]:

```
hospital1 = {'ID':['ID1','ID2','ID3','ID4','ID5','ID6','ID7'],'col1':[5,8,2,
df1 = pl.DataFrame(data=hospital1)
#print(df1)

hospital2 = {'ID':['ID2','ID5','ID6','ID10','ID11'],'col3':[12,76,34,98,65],
df2 = pl.DataFrame(data=hospital2)
#print(df2)


# left join
#df_left = df1.join(df2,how='left',on='ID') # IDs from the left dataframe (c
#print(df_left)

# right join
#df_right = df1.join(df2,how='right',on='ID') # IDs from the right dataframe
#df_right = df2.join(df1,how='left',on='ID')
#print(df_right)

# inner join
#df_inner = df1.join(df2,how='inner',on='ID') # merging on IDs present in bc
#print(df_inner)

# outer join is called a full join
#df_outer = df1.join(df2,how='full',on='ID')  # merging on IDs present in ar
#print(df_outer)
```

Quiz

In [7]:

```
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
# 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 operations, check out pd.concat() and pl.concat()!

Learning objectives

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

- filter columns
- merge and append data frames
- **modify a dataframe**

Do not EVER overwrite the original data files!

Always save the modified dataframe to a new file!

Why do we need to modify the dataset?

- feature engineering
    - generating new features (adding new informative columns) can improve the performance of the ML model
- fix dataset issues
    - there might be data quality issues that need to be manually fixed
    - typos, missing values, etc

In [8]:

```python
import pandas as pd
df_pd = pd.read_csv('../data/adult_data.csv')
print(df_pd.head())

# let's generate a new feature
# is immigrant? False (0) if the person's home country is the USA, True (1)
# such a feature has only two categories but it might be pretty informative
df_pd['is immigrant'] = df_pd['native-country'] != ' United-States'

#print(df_pd.head())
```

```
   age          workclass  fnlwgt   education  education-num  \
0   39          State-gov   77516   Bachelors             13   
1   50   Self-emp-not-inc   83311   Bachelors             13   
2   38            Private  215646     HS-grad              9   
3   53            Private  234721        11th              7   
4   28            Private  338409   Bachelors             13   

        marital-status          occupation    relationship    race      sex  \
0        Never-married        Adm-clerical   Not-in-family   White     Male   
1   Married-civ-spouse     Exec-managerial         Husband   White     Male   
2             Divorced   Handlers-cleaners   Not-in-family   White     Male   
3   Married-civ-spouse   Handlers-cleaners         Husband   Black     Male   
4   Married-civ-spouse      Prof-specialty            Wife   Black   Female   

   capital-gain  capital-loss  hours-per-week  native-country gross-income  
0          2174             0              40   United-States        <=50K  
1             0             0              13   United-States        <=50K  
2             0             0              40   United-States        <=50K  
3             0             0              40   United-States        <=50K  
4             0             0              40            Cuba        <=50K  
```

Mud card

```
In [ ]: 
```