Lesson 2: Exploring and Visualizing Data

Key Ideas:

1. Understanding tabular data: variables and observations
2. Accessing columns and entries of a data frame
3. Arithmetic with lists and columns of a data frame
4. Adding new columns to a data frame
5. Sorting and filtering rows
6. Grouping and aggregating data
7. Visualizing distribution of a variable
8. Visualizing relationships between pairs of numerical variables

In [ ]:
```
# we start by first loading the libraries that we will use in this lesson
import numpy as np
import pandas as pd
import seaborn as sns
```

1. Understanding tabular data: variables and observations

Recall that data frames are simply as the term that pandas uses to refer to data tables (similar to how excel spreadsheets organize data, in rows and columns).

Given a set of data

- An observation refers to an individual or an entity from which data is collected
- A variable is any characteristic / attributes about each individual that is recorded in the dataset

In a well-organized data frame,

- Each row of a data frame corresponds to one observation
- Each column of a data frame corresponds to a variable

Example

In this lesson01 folder, we have a csv file called `berkeley73.csv`. In the code cell below, we load the content of this file as a pandas data frame and name it `berkeleydata`.

How many observations are there in this data frame? How many variables?

In [ ]:
```
berkeleydata = pd.read_csv('berkeley73.csv')
berkeleydata
```

Atrributes of a data frame

Python objects, including data frames, have attributes associated with them.

Attributes of a data frame are values (properties or information) associated to a data frame.

Below are three of the most important attributes of data frames:

- `.shape` : the numbers of rows and columns of the data frame (in this order)
- `.columns` : the list of the column names of the data frame
- `.dtypes` : the list of the data types of the columns of the data frame

Example

Run the code cells below to view the number of rows and columns, the list of column names, and the data types associated to each column of the data frame `berkeleydata` .

In [ ]:
```
berkeleydata.shape
```

In [ ]:
```
berkeleydata.columns
```

In [ ]:
```
berkeleydata.dtypes
```

Method of a data frame

Python objects, including data frames, also have "methods" associated with them. Methods of a data frame are essentially functions that are associated to a data frame (or actions that can be done to a data frame).

Below are three of the most important methods of data frames:

- `.head()` : a function that returns the first few rows of the data frame
- `.tail()` : a function that returns the last few rows of the data frame
- `.sample()` : a function that returns a random selection/sample of rows of the data frame

Example

Run the code cells below to view the first few rows, the last few rows, and a random selection of rows from the data frame `berkeleydata` .

In [ ]:
```
berkeleydata.head()
```

In [ ]:
```
berkeleydata.head(3)
```

In [ ]:
```
berkeleydata.tail()
```

In [ ]:
```
berkeleydata.sample()
```

2. Accessing columns and entries of a data frame

Given a data frame, we might need to extract values from a particular column. There are several ways to do this; here are two of them

1. `DATAFRAMENAME['COLUMNNAME']`
   Gives you a list containing all entries in the column `COLUMNNAME` of the data frame `DATAFRAMENAME`
2. `DATAFRAMENAME.iloc[ : , COLINDEX ]`
   Gives you a list containing all entries in column number `COLINDEX` of the data frame `DATAFRAMENAME`

Example

In the code cell below, we extract only the `Men_Applicants` column from the `berkeleydata` data frame above.

```
In [ ]: berkeleydata['Men_Applicants']
```

You can also extract two or more columns at a time

1. `DATAFRAMENAME[ ['COLUMNNAME1', 'COLUMNNAME2'] ]`
   Gives you a list containing all entries in the columns `COLUMNNAME1` and `COLUMNNAME2` of the data frame `DATAFRAMENAME`
2. `DATAFRAMENAME.iloc[ : , [COLINDEX1, COLINDEX2] ]`
   Gives you a list containing all entries in columns index `COLINDEX1` and `COLINDEX2` of the data frame `DATAFRAMENAME`

Example

In the code cells below, we use the two methods above to extract the `Women_Applicants` and `Men_Applicants` columns from the `berkeleydata` data frame above.

```
In [ ]: berkeleydata[['Women_Applicants', 'Men_Applicants']]
```

```
In [ ]: berkeleydata.iloc[ :, [3, 1]]  # the Women_Applicants column is at column index 3; the Men_Applicants
         column is at column index 1
```

There are also two ways to access an entry in a data frame

1. `DATAFRAMENAME[ 'COLUMNNAME' ][ROWINDEX]`
   To access an entry in row index ROWINDEX and column called `COLUMNNAME`
2. `DATAFRAMENAME.iloc[ ROWINDEX, COLINDEX ]`
   To access an entry in row index `ROWINDEX` and column index `COLINDEX`

Example

In the code cells below, we use the two methods above to extract the number of women applicants in Department C. Department C is at row index 2; the `Women_Applicants` column is at column index 3.

```
In [ ]: berkeleydata['Women_Applicants'][2]
```

```
In [ ]: berkeleydata.iloc[ 2, 3 ]
```

3. Arithmetic operations on columns of a data frame

A column of a data frame is in fact simply a list-like object. When the column of a data frame contains numerical data, we can apply arithmetic operations to it.

Example

Suppose that we would like to compute the total number of applicants (across both of the genders included in this dataset) to each of the six departments in the `berkeleydata` data frame above.

Let us separately extract the `Women_Applicants` and `Men_Applicants` columns, naming the two resulting objects `womenapplicants` and `menapplicants` respectively. Then, we will use them to compute the total number of applicants to each department.

In [ ]:
```
womenapplicants = berkeleydata['Women_Applicants']
menapplicants = berkeleydata['Men_Applicants']

womenapplicants + menapplicants # the total number of applicants to each department
```

Exercise

Please compute the total number of admitted students in each department.

Applying list functions to columns

Because columns are list-like objects, we can apply the following functions to columns:

- `len( LISTNAME )` : to find the "length" of a list (i.e. how many values are stored in a list
- `max( LISTNAME )` : to find the largest value in a list
- `min( LISTNAME )` : to find the smallest value in a list
- `np.sum( LISTNAME )` : to find the sum of numerical values in a list or a numpy array
- `np.mean( LISTNAME )` : to find the mean of numerical values in a list or a numpy array

Example

In the `berkeleydata` data frame above, we can find the total number of men applicants across the six departments using the numpy function `np.sum()` .

In [ ]:
```
num_men_applicants = np.sum( berkeleydata['Men_Applicants'] )

print(num_men_applicants)
```

Exercise

Please find:

1. The total number of women applicants across the six departments
2. The total number of women admitted across the six departments
3. The total number of men admitted across the six departments
4. Use the above information to compute overall acceptance rates (1) for women across all six departments and (2) for men across all six departments. How do these acceptance rates compare?

In [ ]:
```

```

4. Adding new columns to a data frame

To add a new column in a data frame:

```
DATAFRAMENAME[ 'NEWCOLUMNNAME' ] = #insert formula/contents for new column here
```

Example

Suppose that we want to add a new column to the data frame `berkeleydata` above. This column will be called `Total_Applicants` and will consist of the total number of applicants in each department, across both of the genders included in the dataset.

In [ ]:
```
berkeleydata[ 'Total_Applicants' ] = berkeleydata['Women_Applicants'] + berkeleydata['Men_Applicants']

berkeleydata # display the updated dataframe, after the above new column is added
```

Exercise

1. Please add a new column called `Women_Acceptance_Rate` to the `berkeleydata` data frame above, which consists of the acceptance rate of women to each of the six departments
2. Please add a new column called `Men_Acceptance_Rate` to the `berkeleydata` data frame above, which consists of the acceptance rate of men to each of the six departments
3. How do the acceptance rates of men and women to each department compare? How does your observations about these **department-wise acceptance rates** compare to your observation about the **overall acceptance rates** for men and for women when the numbers are aggregated across departments?

The phenomena that you observe in this exercise is known as Simpson's Paradox (https://en.wikipedia.org/wiki/Simpson%27s_paradox).

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

5. Sorting and filtering rows

Sorting rows

Sometimes, we might want to organize the rows based on the values in a particular column.

```
DATAFRAMENAME.sort_values( 'COLUMNNAME' )
```

- Returns the data frame `DATAFRAMENAME` where the rows are sorted based on values in column `COLUMNNAME` , in ascending order (default) `DATAFRAMENAME.sort_values( 'COLUMNNAME', ascending = False )`
- Returns the data frame `DATAFRAMENAME` where the rows are sorted based on values in column `COLUMNNAME` , in descending order

Example

Recall the data frame `berkeleydata` from before. Sort the rows by the number of women applicants in descending order, so that the department with the most number women applicants is listed first.

In [ ]:
```
berkeleydata.sort_values( 'Women_Applicants' , ascending = False )
```

Exercise

Sort the rows of the `berkeleydata` data frame above based on the `Men_Admitted` column, in ascending order.

In [ ]:

Filtering rows based on values in a column

We might want to focus our data analysis only on observations that satisfies particular properties, depending on values in particular columns.

Given a data frame called `DATAFRAMENAME`, to create a new data frame that consists only of rows where the value in a particular column `COLUMNNAME` is **exactly equal to** `VALUE`, we use the following command:

```
DATAFRAMENAME[ DATAFRAMENAME['COLUMNNAME'] == VALUE ]
```

- Returns a data frame containing only rows of DATAFRAMENAME that meets the given criteria.
- In the command above, the portion inside the outer pair of square brackets (namely `DATAFRAMENAME['COLUMNNAME'] == VALUE`) checks whether each row satisfies the given criteria.

Example

In the `berkeleydata` data frame above, suppose that we only want to keep rows where the number of men applicants is exactly 191. Please run the two code cells below.

- The output of the first code cell should be `False, False, False, False, True, False`, which means that only the row (the department) at index 4 have exactly 191 men applicants.
- The output of the second code cell should be a data frame that contains the one row that satisfies this criterion.

In [ ]:
```
berkeleydata['Men_Applicants'] == 191
```

In [ ]:
```
berkeleydata[berkeleydata['Men_Applicants'] == 191]
```

In [ ]:

Other types of filtering conditions

- `DATAFRAMENAME[ DATAFRAMENAME['COLUMNNAME'] < VALUE ]`
  - Returns a data frame with rows where the entries in the `COLUMNNAME` column is less than `VALUE`
- `DATAFRAMENAME[ DATAFRAMENAME['COLUMNNAME'] <= VALUE ]`
  - Returns a data frame with rows where the entries in the `COLUMNNAME` column is less than or equal to `VALUE`
- `DATAFRAMENAME[ DATAFRAMENAME['COLUMNNAME'] > VALUE ]`
  - Returns a data frame with rows where the entries in the `COLUMNNAME` column is greater than `VALUE`
- `DATAFRAMENAME[ DATAFRAMENAME['COLUMNNAME'] >= VALUE ]`
  - Returns a data frame with rows where the entries in the `COLUMNNAME` column is greater than or equal to `VALUE`
- `DATAFRAMENAME[ DATAFRAMENAME['COLUMNNAME'].isin( LISTOFVALUES ) ]`
  - Returns a data frame with rows where the entries in the `COLUMNNAME` column is contained in the list `LISTOFVALUES`

Example

In the code cells below, we keep only rows where

1. The numbers of women applicants are greater than or equal to 340
2. The Department name is A, C, or E

In [ ]:
```
berkeleydata[ berkeleydata['Women_Applicants'] >= 340 ]
```

In [ ]:
```
berkeleydata[ berkeleydata['Department'].isin( [ 'A', 'C', 'E' ] ) ]
```

In [ ]:

6. Grouping and aggregating data

When exploring data, sometimes we want to analyze aggregated information of grouped data.

Example

Below, we have a data frame called `topsongs`, which consists of the top 50 songs in the music streaming app Spotify in 2020. Among the columns in this data frame are the `loudness` column (the song's loudness, in decibels) and the `genre` column.

- We can compute the average loudness of all songs in this dataset using the command `np.mean( topsongs['loudness'] )`.
- However, suppose that we also want to compare the average loudness among songs in different genres. We actually already have the tool to do this:
  - We can first pick a particular genre (for example, 'Hip-Hop/Rap') and filter this data frame, keeping only rows that belong to the genre 'Hip-Hop/Rap'.
  - Then, we can apply `np.mean()` to the `loudness` column of this filtered data frame.
  - Repeat the above steps to each of the other genres

In [ ]:
```
topsongs = pd.read_csv('spotifytoptracks.csv')
topsongs.head()
```

In [ ]:
```
# average loudness (all rows)
overall_average_loudness = np.mean( topsongs['loudness'] )

# keep only songs in the 'Hip-Hop/Rap' genre:
topsongs_hiphoprap = topsongs[ topsongs['genre'] == 'Hip-Hop/Rap' ]
# compute the average of the loudness columns among only the songs whose genre is Hip-Hop/Rap
hiphoprap_average_loudness = np.mean( topsongs_hiphoprap['loudness'] )

print(overall_average_loudness)
print(hiphoprap_average_loudness)

# repeat for the other genres

# EXERCISE: pick one other genre and compute the average loudness
# ...
```

However, if there are a lot of different genres in this dataset, then the above process is tedious.

Fortunately, pandas has another set of tools that could help us "automate" the above two-step process of (1) grouping by genre then (2) getting a summary information (e.g., mean) about songs in each genre.

1. First, **group the rows** by the values in a column, using `DATAFRAMENAME.groupby('COLUMNNAME')`,
2. Then, **get an aggregate/summary information** using `GROUPEDDATAFRAME.agg('mean')`

We can combine the above two steps in one line:

```
DATAFRAMENAME.groupby('COLUMNNAME').agg('mean')
```

In addition to computing mean, we could replace `'mean'` inside `.agg( )` with other summary information, including:

- `'count'` : to count the number of rows that belong to each group
- `'median'` : to find the median value in each group
- `'std'` : to find the standard deviation in each group

We can compute multiple summaries by putting a list of one or more of the above inside `.agg()`.

Grouping and aggregating of values in one column only

The above method computes summaries for each numerical column in the data frame. Suppose that we want to compute the aggregate information only for one particular column, we could use:

```
DATAFRAMENAME.groupby('COLUMNNAME').agg( { 'NUMERICALCOLUMNNAME': ['mean'] })
```

Example

- From the `topsongs` data frame above, find the average and median values of all numerical variables in this data frame, by genre.
- From the `topsongs` data frame above, find the average and median values of just the `loudness` column in this data frame, by genre.

In [ ]:
```
topsongs.groupby('genre').agg( ['mean' , 'median'] )  # computes mean and median of each numerical col
umn
```

In [ ]:
```
topsongs.groupby('genre').agg( {'loudness': ['mean' , 'median']} ) # computes mean and median of just
 the loudness column
```

7. Visualizing distribution of a variable

Bar Charts

We will use a library ("toolbox") called `seaborn` for data visualization

```
import seaborn as sns
```

To visualize the distribution of a **categorical variable**, we create a **bar chart**:

```
sns.displot( data = DATAFRAMENAME, x = 'CATEGORICALCOLUMNNAME' )
```

The above command produces a bar chart with 'count' (frequency) in the vertical axis. To produce a bar chart with ***probability (proportion)*** on the vertical axis, use:

```
sns.displot( data = DATAFRAMENAME, x = 'CATEGORICALCOLUMNNAME' , stat = 'probability')
```

We can create a "horizontal bar chart" by changing `x` to `y` :

```
sns.displot( data = DATAFRAMENAME, y = 'CATEGORICALCOLUMNNAME' )
```

Histograms

To visualize the distribution of a **numerical variable**, we create a **histogram**. The commands below creates the same data visualization:

```
sns.displot( data = DATAFRAMENAME, x = 'NUMERICALCOLUMNNAME' )
```

or

```
sns.histplot( data = DATAFRAMENAME, x = 'NUMERICALCOLUMNNAME' )
```

When creating a histogram, if we want **to specify the number of bins** used (where `N` is any positive integer):

```
sns.displot( data = DATAFRAMENAME, x = 'NUMERICALCOLUMNNAME' , bins = N )
```

or

```
sns.histplot( data = DATAFRAMENAME, x = 'NUMERICALCOLUMNNAME', bins = N )
```

The above commands produce histograms 'count' (frequency) in the vertical axis. To produce a histogram with ***probability density*** on the vertical axis, use:

```
sns.displot( data = DATAFRAMENAME, x = 'NUMERICALCOLUMNNAME' , bins = N , stat = 'density' )
```

or

```
sns.histplot( data = DATAFRAMENAME, x = 'NUMERICALCOLUMNNAME', bins = N , stat = 'density' )
```

Example

Consider the `topsongs` data frame above.

1. Visualize the distribution of the `genre` column. What genres appear more frequently? Least frequently?
2. Visualize the distribution of the `loudness` column. Experiment with different numbers of bins. What does this histogram tell us about the loudness of these top 50 songs in 2020?

In [ ]:
```
sns.displot( data = topsongs, y = 'genre')

# alternatively:
# sns.displot( data = topsongs, x = 'genre')

# Lots of Pop and Hip-Hop/Rap songs; few of the others (e.g., Chamber pop, etc.)
```

In [ ]:
```
sns.displot( data = topsongs, x = 'loudness' , bins = 10)  # try different numbers of bins here

# most songs in this dataset are in the -6 to -4 decibels range; few songs less than -10 decibels
```

8. Visualizing relationships between pairs of numerical variables

Scatterplots

```
sns.relplot(data = DATAFRAMENAME , x = 'COLUMNNAME1', y ='COLUMNNAME2' )
```

Example

Consider the `topsongs` data frame above. Choose any two numerical variables in this dataset (columns); create a scatterplot to investigate whether there is any relationship between the two variables.

In [ ]:
```
# for example: a scatterplot of loudness vs. energy
sns.relplot( data = topsongs, x = 'loudness' , y = 'energy')
```

In [ ]:
```
# choose two other variables and create a scatterplot
#   can you find two variables with a strong linear relationship between them?

# ...
```

Other data visualizations

There are many other types of data visualization out there that the `seaborn` package could help us produce. Please see this Seaborn Gallery (https://seaborn.pydata.org/examples/index.html) for examples of other types of data visualizations.

In [ ]:

In [ ]:

Lesson 2 Exercises

Exercise 1 *(working with data frames)*

In the code cell below, we read a csv file containing top 50 spotify songs and store them as a data frame called `topsongs` . Please use the tools we have learned above to explore `topsongs` and to answer the following questions.

1. Which songs have the 3 highest energy scores and which songs have the 3 lowest energy scores? To answer this question, please sort the rows of the `topsongs` data frame.
2. Create a new data frame that consists only of rows corresponding to songs whose beats per minute is 120 or greater; name this data frame `high_bpm` .
   How many songs have beats per minute of 120 or greater? Do not count by hand; instead, use a data frame attribute to find this information.
3. Summarizing a categorical variable

   - How many different genres are there in the `topsongs` data frame? For each genre, how many songs in this dataset belong to that genre? Please use a python function to obtain an answer to this question.
   - Use a function from the seaborn library to create a bar plot that visualizes the distribution of the Genre variable. Then, in the provided markdown cell, please briefly (1-2 sentences) interpret this data visualization.

In [ ]:
```
topsongs = pd.read_csv('spotifytoptracks.csv')

# your work for exercise 1 here

# ...
```

...

Exercise 2 *(finding data, working with data frames)*

1. Find out if your city/state government maintains an open data portal. For example, New York City government agencies and partners publish public data on NYC Open Data (https://opendata.cityofnewyork.us/). Find a city/state open data portal and explore the available datasets. If there is a dataset that you find interesting, see if the dataset is available in csv form, and use the `pd.read_csv(URL)` function to load it as a dataframe in the code cell below.

   For example, among the many datasets that can be found on NYC Open Data is this fun 2015 NYC Tree Census (https://data.cityofnewyork.us/Environment/2015-Street-Tree-Census-Tree-Data/pi5s-9p35) dataset. We can obtain the URL to the dataset in csv format by clicking **Export**, then right-clicking **CSV** and selecting **copy link address**. The URL should be something like `https://data.cityofnewyork.us/api/views/5rq2-4hqu/rows.csv?accessType=DOWNLOAD` .
   In the code cell below, we load the tree census dataset as a dataframe called `nyc_trees` using the obtained URL.

In [ ]:
```
# nyctrees example
nyctrees = pd.read_csv('https://data.cityofnewyork.us/api/views/5rq2-4hqu/rows.csv?accessType=DOWNLOAD')
nyctrees.head() # preview the content

# datasets that you found yourself
# ...
```

1. Then, please answer the following questions for either the `nyctrees` dataset or another dataset that you found.

   - How many observations (rows) and variables (columns) are there in this dataframe?
   - How many variables are numeric? Which variables are categorical? Are there other variable types?
   - There is a categorical column called `boroname` . This column tells us in which borough (which part of the city) the tree is located.
     - Please use `groupby()` and `agg()` to group the rows based on values in this column and count how many rows belong to each group in this categorical variable. What does the result tell us?
     - Please visualize the distribution of the values in this categorical distribution.
   - There is a numerical column called `tree_dbh` ; this column contains the diameter of each tree in this dataset. How are the values in this column distributed? You can compute summary statistics or create an appropriate data visualization to answer this question.
   - Please find the average tree diameter, grouped by borough. Are the average values very different? Which borough has the highest average tree diameter? Lowest?

In [ ]:
```
# your work for exercise 2 here
# ...
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: