Source

Welcome to Week 2 Lecture 1!

Data Science in Python & Machine Learning

Learning Goals

By the end of this lesson you will be able to:

1. Recognize Week 2 Core Assignments
2. Make your assignment notebooks awesome - Assignment Tips
3. Explain the date and requirements for taking the Belt Exam
4. Set the index of a dataframe
5. Use .loc and .iloc to select specific data from a data frame
6. Filter rows to return a subset of data that matches a pattern
7. Use Pandas string methods to create filters based on string patterns.

Week 2 CORE Assignments

These MUST be submitted by Sunday December 12th:

1) Loading & Filtering Data
2) Titanic Data
3) Project 1 - Part 2

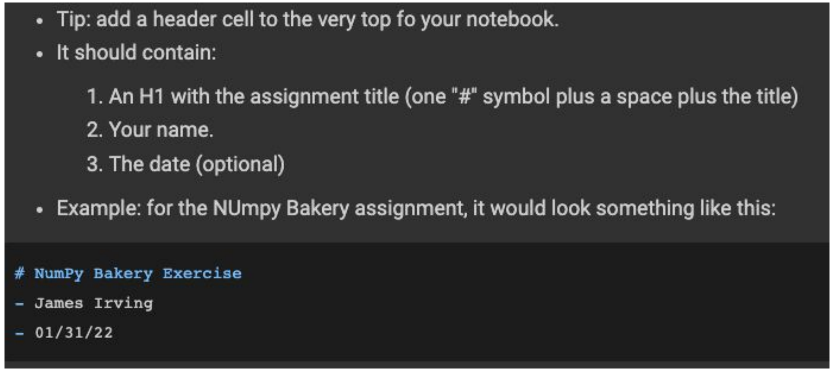
If a core assignment is missing, you will be placed on academic probation

- even if you have done other assignments and
- even if you have spent a lot of time learning and working

Note: If you accumulate 3 academic probations at any time during the program, you will be recommended for dismissal. Probations are cumulative and irreversible except in the case of documented mitigating circumstances.

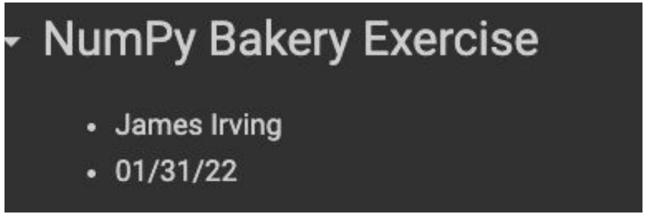
Assignment/Notebook Tips Colab Notebook

- Assignment Header



- Tip: add a header cell to the very top fo your notebook.
- It should contain:
    1. An H1 with the assignment title (one "#" symbol plus a space plus the title)
    2. Your name.
    3. The date (optional)
- Example: for the NUmpy Bakery assignment, it would look something like this:

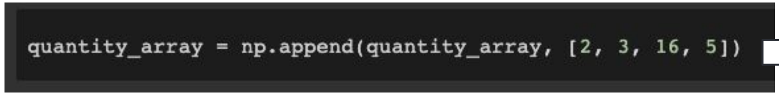
```
# NumPy Bakery Exercise
- James Irving
- 01/31/22
```



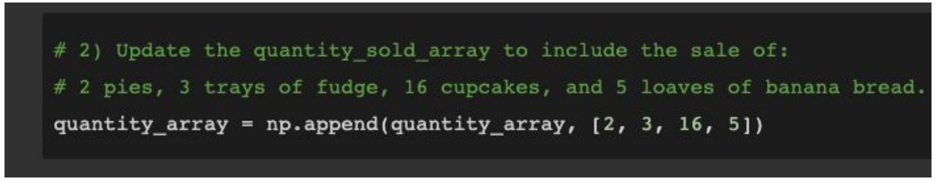
NumPy Bakery Exercise

- James Irving
- 01/31/22

- Code Comments:



```
quantity_array = np.append(quantity_array, [2, 3, 16, 5])
```



```
# 2) Update the quantity_sold_array to include the sale of:
# 2 pies, 3 trays of fudge, 16 cupcakes, and 5 loaves of banana bread.
quantity_array = np.append(quantity_array, [2, 3, 16, 5])
```

- **More tips/suggestions in the Colab Notebook!**

BELT EXAM - STACK 1

End of Week 3 (Stack 1) Belt Exam:

- Fri - Sun: 02/11 - 02/13
    - Must have submitted by 11:59 PM PST on Sunday 02/13

Eligibility:

- 80% Class Attendance
- 90% Successful Completion of Assignments
- REQUIRED: complete all week 1 and week 2 assignments

Belt Exam Process

- If you are eligible, you receive a **belt exam unlock code.**
- Once unlocked, you have **24 hours** to complete the exam.
- RECOMMENDED: complete week 3 before starting exam.
- NO help from TAs/other students/instructors
    - But you CAN use your notes and google.
- 1 chance to retake if you do not pass.

🥊 SLIDESHOWS vs NOTEBOOKS

- Today's Lecture:
  - 📓 Notebook Version
  - 🎥 Slides Version (this slide deck)

- We will walk through the **notebook** version during today's lecture.
  - The same information is also in this slide deck.

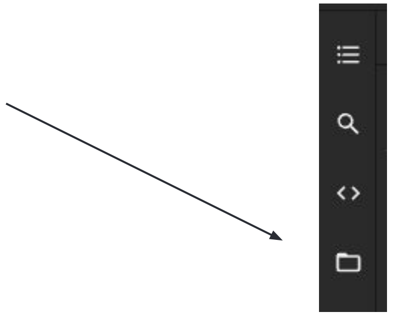
Loading and Exploring Data with Pandas 🐼!

Getting Data into Colab

Two Options:

1. Save file in Drive
    a. Mount drive
    b. Open file folder in Colab on the left:
    c. Navigate to file in Drive
    d. Right click file and ‘copy filepath’

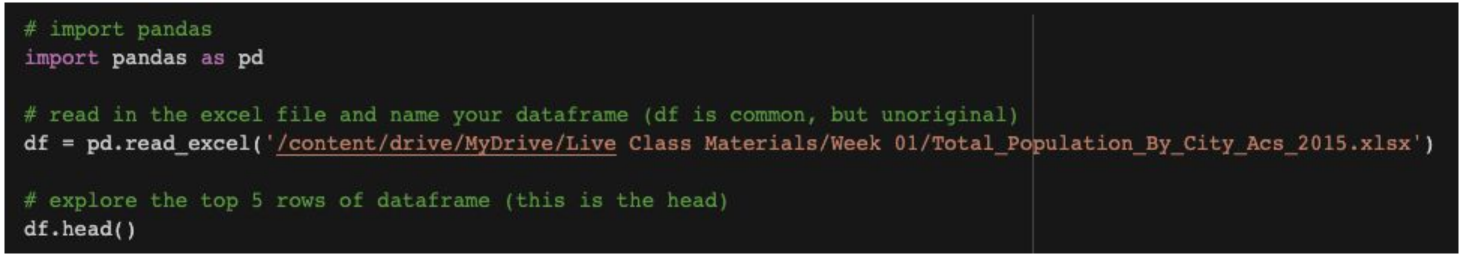
2. Download data locally and upload to temporary Colab environment
    a. Download data
    b. Open file folder on the left
    c. Drag and drop the file from your computer to upload it
    d. Right click file and ‘copy filepath’



pd.read_csv() and pd.read_excel()

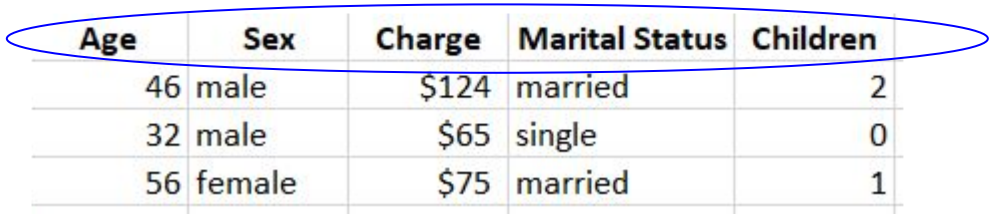
Different file types need different functions to load them:

- .csv files → pd.read_csv(‘filepath’)
- .xlsx files → pd.read_excel(‘filepath’)
- ‘filepath’ can be a remote URL as long as it points directly to a .csv or .xlsx file.
- Other functions exist to load other file types, but these are the ones we will use.



```
# import pandas
import pandas as pd

# read in the excel file and name your dataframe (df is common, but unoriginal)
df = pd.read_excel('/content/drive/MyDrive/Live Class Materials/Week 01/Total_Population_By_City_Acs_2015.xlsx')

# explore the top 5 rows of dataframe (this is the head)
df.head()
```

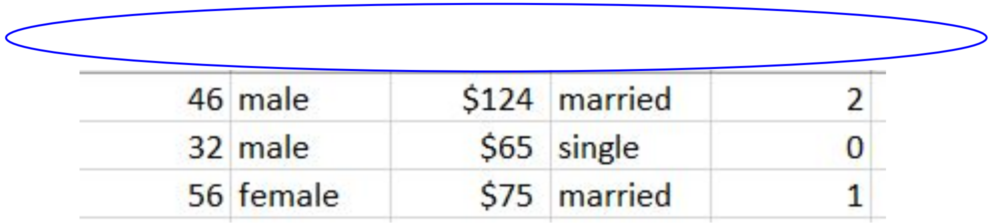
Headers



Age	Sex	Charge	Marital Status	Children
46	male	$124	married	2
32	male	$65	single	0
56	female	$75	married	1

Header is top row. (This is default assumption)

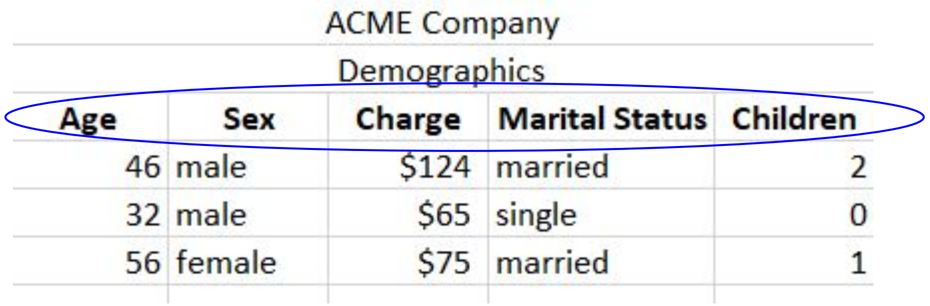
```
pd.read_excel(‘path’)

pd.read_excel(‘path’, header = 0)
```



46	male	$124	married	2
32	male	$65	single	0
56	female	$75	married	1

No header

```
pd.read_excel(‘path’, header = None)
```



		ACME Company		
		Demographics		
Age	Sex	Charge	Marital Status	Children
46	male	$124	married	2
32	male	$65	single	0
56	female	$75	married	1

Header present, but not at the top

```
pd.read_excel(‘path’, header = 2)
```

Quick Look at top and bottom of dataframe

This will give you the top 5 rows:	`df.head()`
To see a different number of rows such as the first 8 rows:	`df.head(8)`
To see the bottom 5 rows	`df.tail()`
To see a different number of rows such as the last 11 rows	`df.tail(11)`

```
To see random rows
df.sample(5)
```

Checking data types

To get lots of info including data type, index, and Column Names	`df.info()`
To get JUST the data type for all columns:	`df.dtypes`
To get the data type for just one column	`df[‘name’].dtypes`

Slicing Dataframes:

Selecting specific column(s)

To select 1 column as a pandas **series**	`df[‘name’]`
To select 1 column as a pandas **dataframe**	`df[[‘name’]]`
To select multiple columns as a **dataframe**	`df[[‘name’, ‘Manufacturer’]]`

Notice double square brackets for dataframe

How many rows and columns?

To get lots of info including data type, index, and Column Names	`df.info()`
To get the shape (rows, columns)	`df.shape`
To get the number of rows	`len(df)`
To get the number of columns	`len(df.columns)`

Accessing and Changing Data

Index

- When you read in a dataset, pandas will assign an index value to each row in the dataset

- Remember Python uses 0 indexing, so the first row of data will be assigned 0.

- You can leave this alone, but sometimes we may want to change our index to correspond with a column of our data

- For example, if our data contains a column such as Employee_ID, that might be a logical column to use as the index

- In order to be used as an index, every value in the column must be unique!

Setting the index

- To set the index, you can use the following code:

  ```
  df.set_index([‘Employee_ID’], drop = True,  inplace = True)
  ```

- Note that the drop = True means that the original column will be dropped and you will only have the data in the index column

- The inplace = True means that you want to permanently change the df

Locators:
.loc and .iloc

Data in dataframes can be accessed directly using the powerful indexing tools, .iloc and .loc. These are used like slicing, with square brackets, not like methods with parentheses.

Can be thought of as latitude and longitude

They both take arguments in the same format. df.loc[row, column] and df.iloc[row, column]

A **row** is one observation and a **column** is a variable or feature.

However, they differ slightly in how you reference the row and column.

Index Locator (.iloc)

- If you know the **index value** or position, you can use the **i**ndex **loc**ator (.iloc)

- .iloc only takes integers

- Return the entire 8th row as a series `df.iloc[7]` or `df.iloc[7, :]`

- Return the entire 8th through 10th rows as a dataframe `df.iloc[7:10]` or `df.iloc[7:10, :]`
  - `Starts at index 7 and ends at index 9 (the last number is exclusive)`

- Return the 2nd column of the of the 8th row `df.iloc[7, 1]`
  As just the value in that cell

- Return the 2nd through 5th columns of the 8th row `df.iloc[7, 1:5]`
  As a series.

Note:
If you select 1 row, it returns the data as a series.

If you select more than one row, it returns the data as a data frame

General Locator (.loc)

- Can take strings and integers
- 6th row of the column ‘name’ `df.loc[5, ‘name’]`

In Pandas, indices can be strings

- Index ‘Jerry’ of column ‘height’ `df.loc[‘Jerry’, ‘height’]`
- `Indices ‘Jerry’ through ‘Hubert’ Of column ‘height’` `df.loc[‘Jerry’:‘Hubert’, ‘height’]`

General Locator Can Also be Used to Change Values

- Change Jerry’s height to 6.5

```
df.loc[‘Jerry’, ‘height’] = 6.5
```

- Change ‘Jerry’ through ‘Hubert’s Heights to the mean of all heights

```
df.loc[‘Jerry’:‘Hubert’, ‘height’] = df[‘height’].mean()
```

Filters

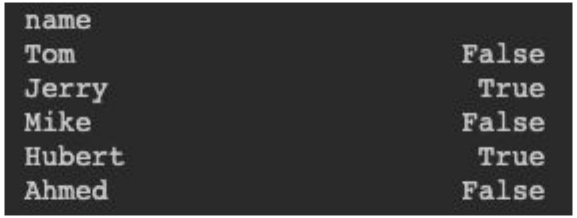
- Filtering is the process of selecting data based on particular criteria
- We often want to focus on a subset of our data.

  You can use boolean expressions to filter rows by the value in a column.
- You can select using >, <, ==, or any other function or method that returns a boolean
- To select all columns for whose age is 15: `df.loc[df[age] == 15, :]`
- To select the names of people who are 15: `df.loc[df[age] == 15, ‘name’]`

Filtering (cont.)

Using a series with a boolean expression returns a **series of booleans:**

df['age'] > 15 returns:



```
name
Tom                      False
Jerry                     True
Mike                     False
Hubert                    True
Ahmed                    False
```

If we use this boolean series as an index value with .loc, it will only return the rows with 'True'.

To return only the rows for people over the age of 15, we can use:

```
filter =  df['age'] > 15
df.loc[filter]
```

We could also change only their height!

```
df.loc[filter, 'height'] = 5.5
```

Boolean String Methods

- There are a lot of string methods and many return booleans!!
- The previous example showed how to filter for a string that starts with ‘B’

`df[‘City’].str.startswith(‘B’)`

- There are similar methods for ends with or contains

`df[‘City’].str.endswith(‘i’)` outputs cities that end in ‘i’ such as Miami

`df[‘City’].str.contains(‘or’)` outputs cities that have ‘or’ in them such as New York

Combining locator and min or max

df.min() gives us the minimum value, but how do we find information about the data with this minimum value?

Option 1: Assign a variable and use locator

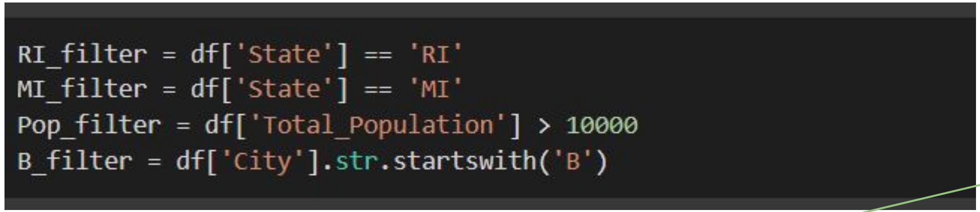
```
min_value = df[‘column’].min()

df.loc[df[‘column’] == min_value, :]
```

Option 2: Put it all into one line of code

```
df.loc[df[‘column’] == df[‘column’].min(), :]
```

Multiple Filters

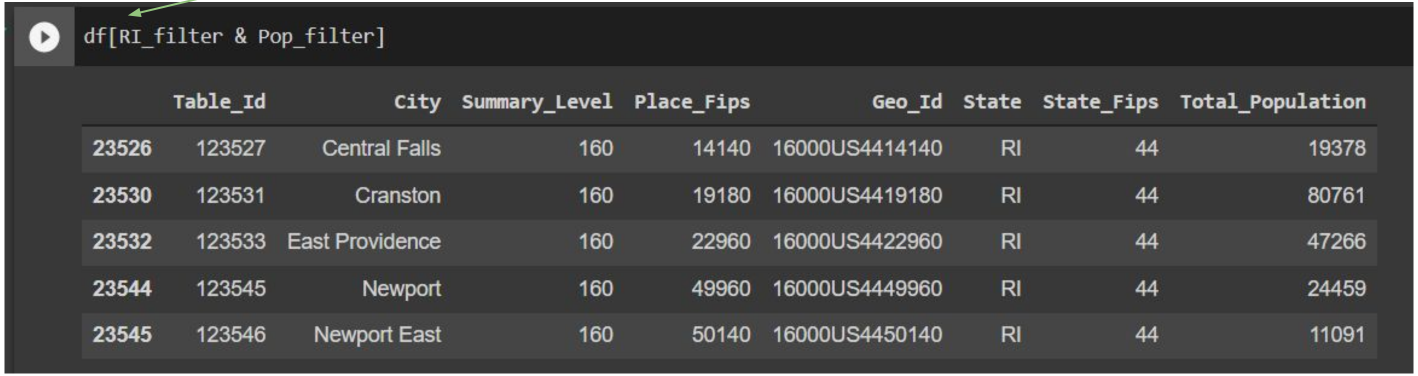
To combine filters use: [&, I, ~]
These translate to [and, or, not].



```
RI_filter = df['State'] == 'RI'
MI_filter = df['State'] == 'MI'
Pop_filter = df['Total_Population'] > 10000
B_filter = df['City'].str.startswith('B')
```

You can also filter without using .loc

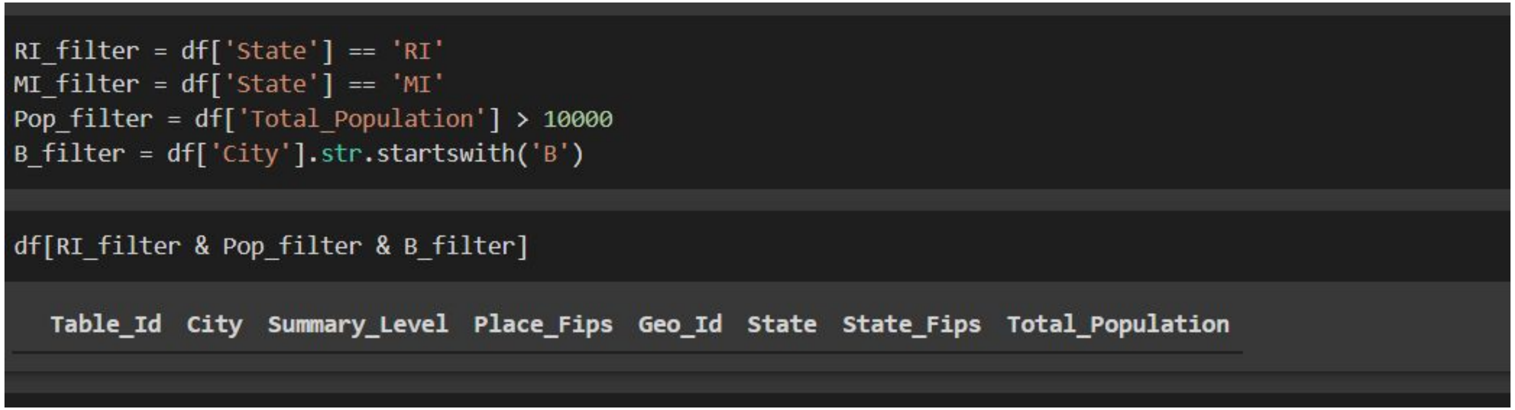
Now you can combine! For example all cities in RI that have a population greater than 10000



```
df[RI_filter & Pop_filter]
```

	Table_Id	City	Summary_Level	Place_Fips	Geo_Id	State	State_Fips	Total_Population
23526	123527	Central Falls	160	14140	16000US4414140	RI	44	19378
23530	123531	Cranston	160	19180	16000US4419180	RI	44	80761
23532	123533	East Providence	160	22960	16000US4422960	RI	44	47266
23544	123545	Newport	160	49960	16000US4449960	RI	44	24459
23545	123546	Newport East	160	50140	16000US4450140	RI	44	11091

No output? No problem!



```
RI_filter = df['State'] == 'RI'
MI_filter = df['State'] == 'MI'
Pop_filter = df['Total_Population'] > 10000
B_filter = df['City'].str.startswith('B')
```

```
df[RI_filter & Pop_filter & B_filter]
```

```
  Table_Id  City  Summary_Level  Place_Fips  Geo_Id  State  State_Fips  Total_Population
```

- We tried combining three filters, and we did not get any output.
- This is not an error!
- This just means that there are NO data points that meet our criteria.
- In this case, there are no cities in RI with a population greater than 10000 that also start with letter B

City Population Challenge - Revisited:

1. Make a copy of this Colab Notebook

2. Continue from header "Exploring City Populations - Part 2 [NEW]"

📔 Today's Notebooks:

- Lecture Notebook
- Assignment Tips notebook
- Class Activity Notebook (did not get to during class)