Introduction to Data Analysis II

THE KNOWLEDGE HOUSE

Celebrating 10 Years of Diversifying Tech

Agenda - Schedule

1. Pandas Basics
2. Data Loading & Basic Manipulations
3. Cleaning your Data
4. Indexing your Data
5. Break
6. Lab

Pandas (styled as pandas) is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.
https://en.wikipedia.org/wiki/Pandas_(software)

Agenda - Goals

- Load a dataset into a pandas DataFrame
- Use basic methods to explore and describe your dataset
- Drop rows with missing values
- Select specific columns using .loc
- Apply simple boolean conditions to filter your data

Announcement(s)

- **Week 6 Pre-Class Quiz** due 4/15 (*2 attempts*)
- **Career Class on 4/17** (*this time for real*)
- **TLAB #2** due 4/21

Spreadsheets - Limitations

Limitations to Spreadsheets

As we established, Google Sheets should not be your goto solution for data manipulation for a few reasons:

- **Manual** data transformations
- Running data transformations on a **schedule** is clunky
- Will **not** prevent disastrous errors
- ~~Telling people you use Google Sheets is not as cool as telling people you use Python~~

Therefore, we will instead learn about the **wonderful world of data manipulation packages**.

code	iso	country	PMD.raw.2015	PMD.raw.2016	PMD.raw.2017	PMD.raw.2018	PMD.raw.2019
826	GBR	United Kingdom	294.8980096	289.6564515	286.4196291	285.8509262	280.5127785
840	USA	United States of	283.5238203	267.2755296	252.9543121	250.8516978	249.4296641
32	ARG	Argentina	624.5758182	620.0178524	596.8603743	585.5216743	580.4380542
156	CHN	China	1921.885946	1854.62846	1782.888749	1749.996404	1743.3549
566	NGA	Nigeria	1929.854375	1904.940118	1849.284647	1838.035469	1869.209248
818	EGY	Egypt	4388.481759	4248.828354	4089.565822	4034.412844	3993.224209
76	BRA	Brazil	568.7506785	568.0071932	552.8455853	546.0554702	542.7443866
392	JPN	Japan	262.7909262	261.7102781	257.2665629	256.6441396	256.0192158
643	RUS	Russia	975.8511373	871.8861424	763.8980703	754.5885114	764.4997137
276	DEU	Germany	375.4855158	353.7205444	338.2790574	335.3876598	334.522656

For example, let's say you work for the last non defunded environmental agency in the US. Your task would entail performing our stated data operations from yesterday **multiple times** (and perhaps on a **schedule**). Google Sheets is not the answer!

Pandas

Pandas (*portmanteau of panel-data*) was built by a **researcher/open-source programmer** working at AQR Capital.

However the ease-of-use and power of this package has slowly made it a **dominant tool** in the world of programmatic data manipulation.

While this is currently a funded project, open source is a **great idea**, and has given us some truly useful tools.

There are some notable contenders out there for programmatic data manipulation tools (for example **polars**). However pandas is not going away anytime soon.

Always always consider the **why** of what you are doing.

A Reminder - Data Analysis

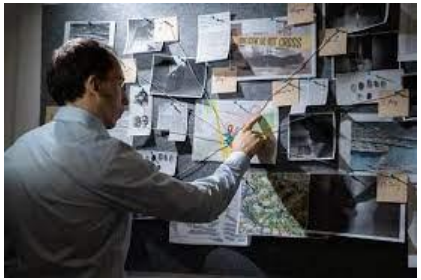
An important thing to note is that it is **not important to memorize pandas syntax**. This will come naturally as you continue exploring this package.

Always remember the **why** of the data analysis that you are performing.

Companies much rather hire someone that **understands** what **good analytics & statistics** looks like, but has a looser grasp on pandas…

…as opposed to a walking pandas documentation that doesn't **fundamentally understand the analytics** that they are doing.

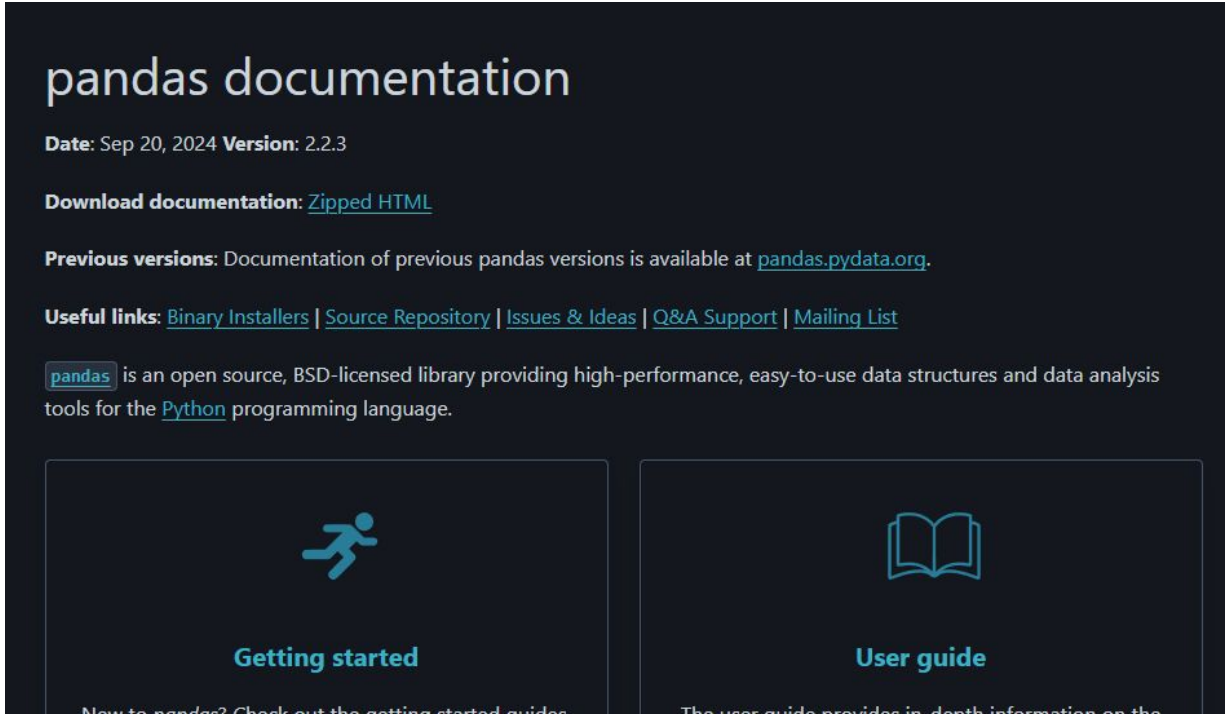
A Reminder - Data Analysis



Remember, the steps you take as a data analyst such as…

- **Transforming** your dataset
- Calculating **descriptive statistics**
- Making **pivot tables**
- Making **visualizations**

Must be in support of answering a larger business-related question. *How much money did we make? How much money did we lose? Can we predict this?*



Everything we discuss today is located in the docs: https://pandas.pydata.org/docs/

Don't rely on our word for the programs you write, **look up the docs!**

Pandas Basics

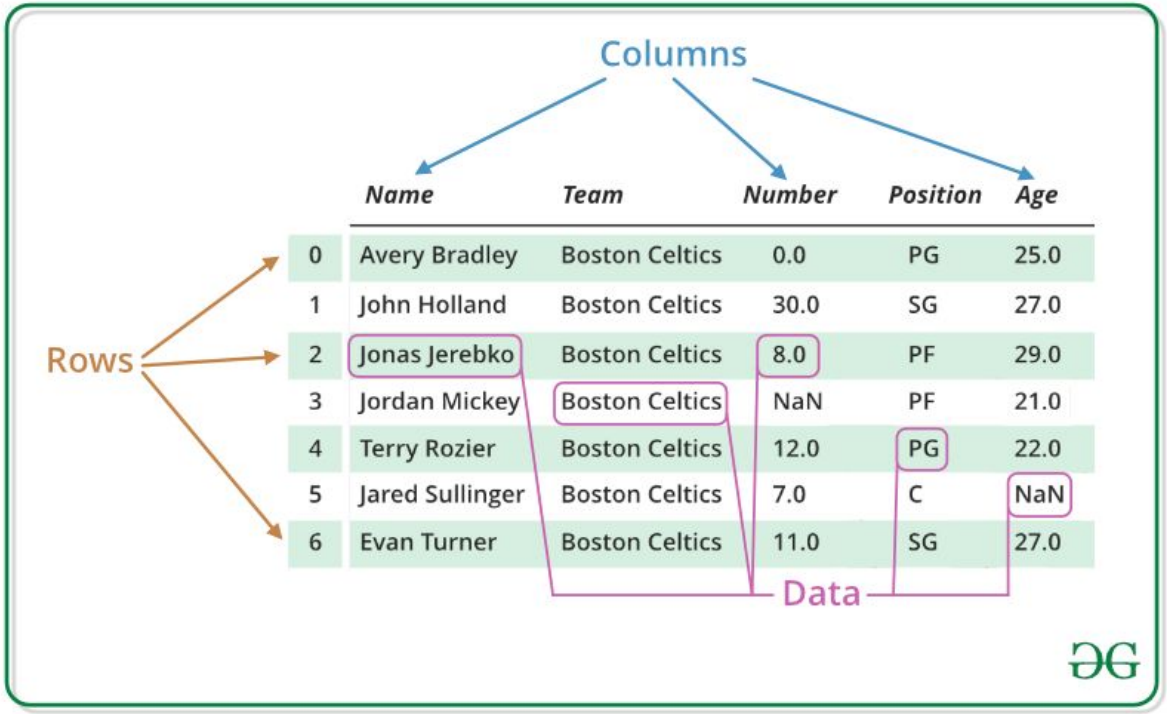
Python & Pandas Review - Pandas

Going forward, we will always use **pandas for data manipulation.**

At the end of the day, the meat and potatoes (*or beans and potatoes*) of data scientists is **pandas** and its **application programming interface** (API).

Whenever we talk about **pandas API**, we're primarily talking about its **methods and subsequent documentation**.

Before we learn about its API, lets learn a little bit about the **core concepts of pandas**.



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Just like any spreadsheet technology, we are primarily focused on manipulating a **structured data format** consisting of **columns** and **rows**.

Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

In pandas, we consider individual rows or columns to be the **series object**, and entire datasets to be **dataframe object**.

For today, we will primarily focus on manipulating dataframes.

Python & Pandas Review - Pandas

Let's discuss more about what we can do with **dataframes** in pandas:

- Quickly calculate **descriptive statistics**
- **Observe** raw data
- Note **missing values**
- **Filter** data
- ...and more!

Before we get to all of this however, we must figure out how to actually **import** this data into your jupyter notebooks.

```
import pandas as pd

df = pd.read_csv("file.csv")
```

This is done simply by calling the "**read_csv**" method on the path where your CSV file is located!

There are a couple of **other data formats** we can read, but we will focus solely on CSV's for today.

```
import pandas as pd
```

```
df = pd.read_csv("file.csv")
```

Notice that we **alias** the package so that we don't have to write "pandas" each time we use the package.

```
import pandas as pd
```

```
df = pd.read_csv("file.csv")
```

Furthermore, notice that we save the return value of **read_csv** to a variable. Remember, what is not saved to a variable is lost.

Method	Purpose
df.head()/df.tail()	gives the **first/last n rows** of your dataframe, respectively
df.describe()	computes **summary statistic**s for all Series/columns, excludes NA values
df.info()	Get quick information on each column
df.dropna()	Drops unknown/None/NULL values

There are many more pandas operations as listed in the docs. We will begin with the **most commonly used.**

df = pd.read_csv(“file.csv”)

Once we have this dataframe object created, we can use this variable name for any additional functionality (**df**).

df.method()

The most common pattern entails calling a **method** off of this dataframe.

df.isna()

Col1	Col2
True	True
False	False
False	True
False	True
True	False
True	True
True	True

On its own, this method simply returns a **series of Booleans (this pattern will come up often)**

df.isna().sum()

Col1 4
Col2 5

By doing **method chaining,** we can instead get the count of missing values from a column

Cleaning Data

Basic Cleaning

As we create DataFrames from file sources, **we may need to clean our data**, there are all methods we perform on the DataFrame itself

Method	Purpose
df.fillna()	fill NaN values with a specified method
df.dropna()	drops columns/rows with NaN values
df.drop()	drops defined columns/rows
df.replace()	replaces given values with other values

You shouldn't fill this with arbitrary values. We will talk about the statistics behind this.

df.fillna()

we can fill NaN values in our data with **something that makes sense**

NaN values are an actual data type and not just a string that says "NaN"

This means **nothing was present at all for that data entry**

We can fill either with numbers (0, 1, 200), strings ("filled", "no good"), or equations (np.mean(column))

Pandas Fill NA

pd.DataFrame.fillna(value="Filled")

Index	Name	num_customers
0	Foreign Cinema	50
1	<NA>	45
2	500 Club	<NA>
3	The Square	<NA>

Index	Cust_Name	num_customers
0	Foreign Cinema	50
1	Filled	45
2	500 Club	Filled
3	The Square	Filled

Fill in your <NA> values with another value

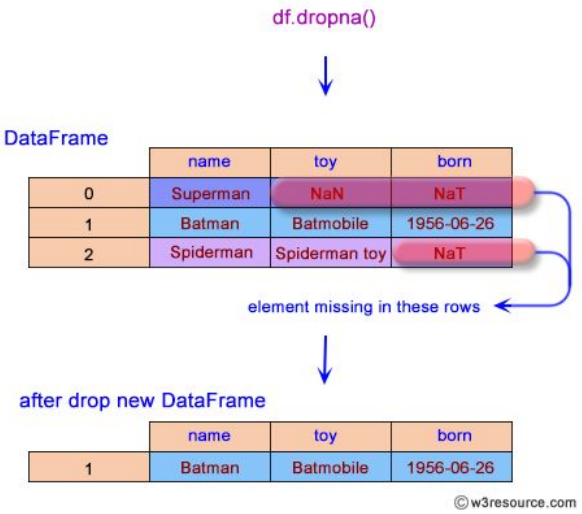
df.dropna()

Very simply will drop values that have NA, **but we can define the behavior**

Parameters:

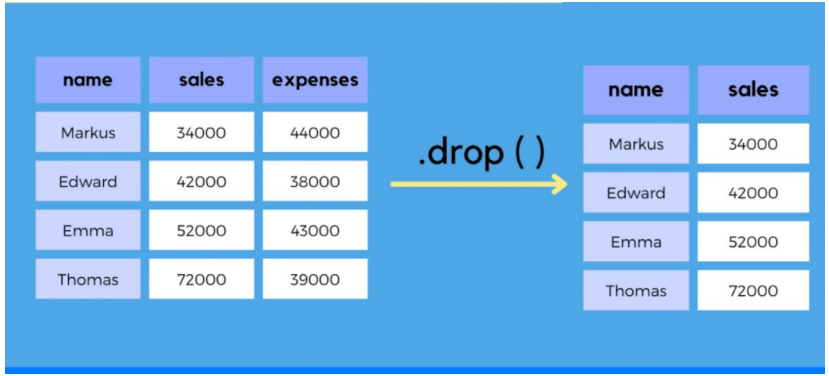
df.dropna(how='any') is the default, will drop if ANY value in a row/column has NA

df.dropna(how='all') will drop only if the ENTIRE row/column is NA



df.drop()

Very similar to dropna() but instead of looking for NaNs, we simply define what we want to drop.



df.drop(columns=[“col_name1”, “col_name2”]) will drop specific columns.

df.drop(axis=0) default behavior will drop based on rows

df.drop(axis=1) will drop based on columns

Pandas Modification In-Place

Pandas Review - Basics

It's important to understand the common patterns we take when writing pandas code…

```
df = pd.read_csv("...")
```

Pandas Review - Basics

Often times we perform **variable assignment** in pandas. This is when we **call** a **method** that returns something we want to perform further calculations on…

```
df = pd.read_csv("...")
```

It does not make sense to just call **pd.read_csv("...")** without saving this into a variable. We will be using this variable in later sections of our code!

Pandas Review - Basics

This also applies to **methods that do not modify data frames in place**. Keep in mind that by default, most methods **do not** modify data frames in place!

```
df.dropna()
```

By itself, this is ineffective piece of code…

Pandas Review - Basics

This also applies to **methods that do not modify data frames in place**. Keep in mind that by default, most methods **do not** modify data frames in place!

```
df = df.dropna()
```

We must reassign it into another variable (could be the same var name if desired).

Pandas Review - Basics

However, there is a way to ensure that we modify this dataframe in place, looking at the documentation, we can see that we can use a parameter…

df.dropna()

Pandas Review - Basics

We can use **inplace = True**. Remember, a **parameter** is a **variable** that belongs to a **method** whose values we can change.

```
df.dropna(inplace=True)
```

inplace = True is actually a bit limiting, as we'll see in later slides.

pandas.DataFrame.dropna

```
DataFrame.dropna(*, axis=0, how=_NoDefault.no_default,
thresh=_NoDefault.no_default, subset=None, inplace=False, ignore_index=False)
```

Remove missing values. [source]

No one expects you to carry these parameters in your mind. Look up the documentation.

Everything that has a value is a **parameter** of "dropna." This includes ***"axis"***, ***"how"***, ***"thresh"***, ***"subset"***, ***"inplace"***, ***"ignore_index"***

Pandas Review - Basics

In summary, these are the most **common lines of pandas** you are going to be writing. Almost all the **data manipulations** can be handled via these patterns. Always consider the pandas API before making your own solution.

```
df = df.method(...)
```

```
df.method(...)
```

Indexing/Selecting Data

Pandas - Column Indexing

We can access information in a variety of ways, starting with column-based operations. Notice the parallels between this syntax and what we've been coding in Python!

df["col1"] selects a single column from a dataframe (*as a Series*)

df[["col1", "col2"]] selects a multiple columns from a dataframe (*as a dataframe*)

df.loc[:, "col1"] does the same as the first-line, with some slight nuance…

df.loc[:, "col1":"col5"] similar to how lists work, we can also use the ":" operator to slice a dataframe. (*Notice we specify rows then columns*)

Single brackets for one column (returns a series)

df[‘col1’]

df[[‘col1’, ‘col2’]]

Double brackets for multiple columns (returns a dataframe)

Indexing by **columns** just uses the **square bracket notation**

Pandas - Row Indexing

Moving forward to row based operations

df.loc[0] this selects the first row of data

df.loc[0:10] this selects the first 10 row of data

What if we want to filter our data according to some **conditional** instead?

```
df.loc[row, column]
```

This is the general pattern

```
df.loc["a", ]
```

And here we select the row labeled as "a"

```
df.iloc[5:, ]
```

Here we select all rows after "5"

To index by **rows**, we can either use "***loc***" or "***iloc***." "loc" allows us to use label-based indexing (if it's available).

Assumptions of list slicing apply here.

pandas Filtering

We index “boolean” arrays to achieve a filter.

Namely, we can create an array of boolean values (T or F) by checking for the equality or inequality in a certain **column**.

Let’s say we want to select only rows that include “City Hotel”

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month
40060	City Hotel	0	6	2015	July	27	1
40061	City Hotel	1	88	2015	July	27	1
40062	City Hotel	1	65	2015	July	27	1
40063	City Hotel	1	92	2015	July	27	1
40064	City Hotel	1	100	2015	July	27	2
...	...	...	...	...	...	...	...
119385	City Hotel	0	23	2017	August	35	30
119386	City Hotel	0	102	2017	August	35	31
119387	City Hotel	0	34	2017	August	35	31

pandas Filtering

Firstly, what is the name of the **feature(**aka columns) that describes which hotel our clientele booked with?

We can check this by running **df.head()**

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month
40060	City Hotel	0	6	2015	July	27	1
40061	City Hotel	1	88	2015	July	27	1
40062	City Hotel	1	65	2015	July	27	1
40063	City Hotel	1	92	2015	July	27	1
40064	City Hotel	1	100	2015	July	27	2
...	...	...	...	...	...	...	...
119385	City Hotel	0	23	2017	August	35	30
119386	City Hotel	0	102	2017	August	35	31
119387	City Hotel	0	34	2017	August	35	31

pandas Filtering

“hotel”

This will be the **“hotel” column**.

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month
40060	City Hotel	0	6	2015	July	27	1
40061	City Hotel	1	88	2015	July	27	1
40062	City Hotel	1	65	2015	July	27	1
40063	City Hotel	1	92	2015	July	27	1
40064	City Hotel	1	100	2015	July	27	2
...	...	...	...	...	...	...	...
119385	City Hotel	0	23	2017	August	35	30
119386	City Hotel	0	102	2017	August	35	31
119387	City Hotel	0	34	2017	August	35	31

pandas Filtering

Next, we will use our **equality operator** (==) to check if a row in the "*hotel*" column is equal to the string of "*City Hotel*"

df["hotel"] == "City Hotel"

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month
40060	City Hotel	0	6	2015	July	27	1
40061	City Hotel	1	88	2015	July	27	1
40062	City Hotel	1	65	2015	July	27	1
40063	City Hotel	1	92	2015	July	27	1
40064	City Hotel	1	100	2015	July	27	2
...	...	...	...	...	...	...	...
119385	City Hotel	0	23	2017	August	35	30
119386	City Hotel	0	102	2017	August	35	31
119387	City Hotel	0	34	2017	August	35	31

pandas Filtering

Next, we will use our **equality operator** (==) to check if a row in the "*hotel*" column is equal to the string of "*City Hotel*"

Notice that this does not give us the **filtered dataframe yet**, but instead gives us a Series of True & False

df["hotel"] == "City Hotel"

```
0         False
1         False
2         False
3         False
4         False
          ...
119385     True
119386     True
119387     True
119388     True
119389     True
Name: hotel, Length: 119390, dtype: bool
```

pandas Filtering

Finally, we place this boolean index inside of our dataframe to get our filtered dataframe.

```
df[df["hotel"] == "City Hotel"]
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month
40060	City Hotel	0	6	2015	July	27	1
40061	City Hotel	1	88	2015	July	27	1
40062	City Hotel	1	65	2015	July	27	1
40063	City Hotel	1	92	2015	July	27	1
40064	City Hotel	1	100	2015	July	27	2
...	...	...	...	...	...	...	...
119385	City Hotel	0	23	2017	August	35	30
119386	City Hotel	0	102	2017	August	35	31
119387	City Hotel	0	34	2017	August	35	31

The general pattern

```
df[conditional]
```

```
df[ df['col1'] < 10]
```

Select all rows where 'col1' is less than 10

Boolean indexing entails bracket notation, but this time with a conditional…

Pandas Lab

Pandas Lab

Open the **pandas_lab** folder and begin by reading the instructions!

Complete this lab in your groups!

We will take the last 10 minutes of lecture to revisit this lab together.

Wednesday

Wednesday will entail:

- How do we actually **write** about the insights that we discover?

Pandas: SettingWithCopy Warning

If you understand what you're doing, you're not learning anything. - Anonymous