Session 5

Continuing with Pandas

Cleaning Data

Prepping Data

10.1.19

Link to GitHub Repo:
https://github.com/data-voyage-solutions/OAG_project_work

01 Git / GitHub / local IDE

1 hour

02 Common data cleaning tasks

1 hour

03 Understanding your workflow

1 hour

During Git/GitHub/IDE Setup

- ❑ Connect to a remote db
- ❑ The Megan Challenge

Github Repo

- Check your email for an invite.
- Go to the GitHub Repo, and get the link to git clone
- Navigate to the local directory where you want to create the git clone
- git clone
- Create a folder with your name, and add a file in there with some text.
- Push your changes and confirm.

Make sure you have anaconda python locally

- Launch Jupyter Lab
- Navigate to the **session_5.ipynb** file.
- Make a copy of the file, and move the copy to the folder that you created.
- Push changes to master.
- Work through the notebook.

Think about your workflow...

Why are we learning Python?

What's your data prep routine when you first get a project data set?

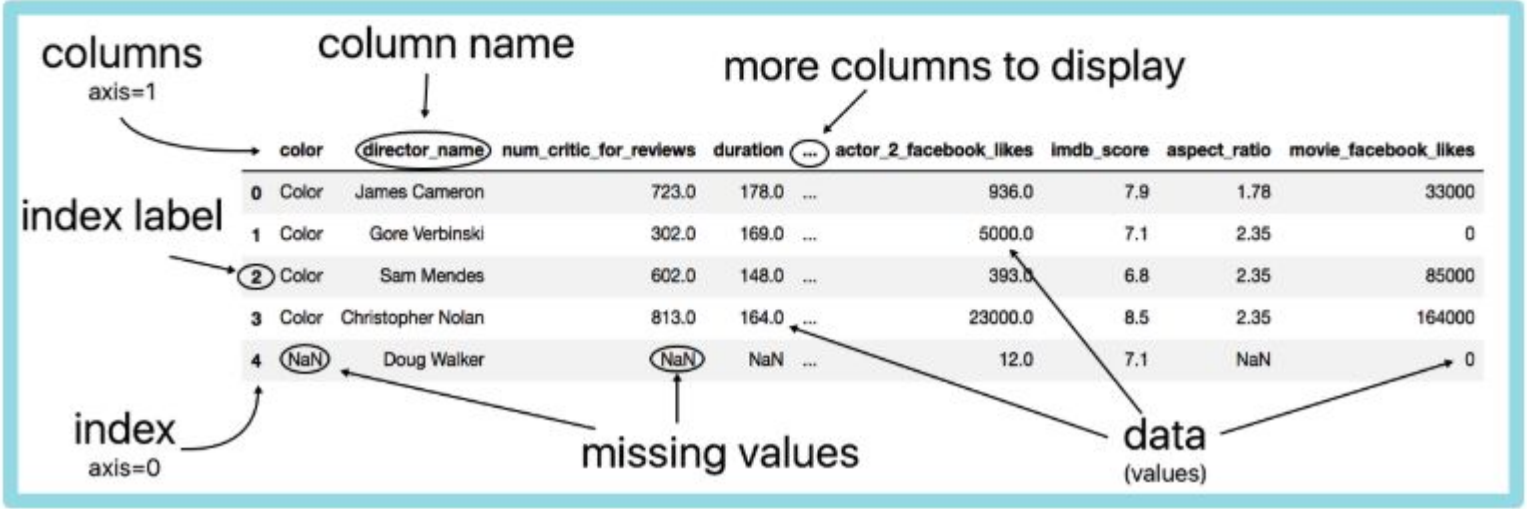
What steps do you take in Excel or SQL?

Common Data Cleaning Tasks

- *Load Data*
- *Inspect data*
- Rename columns
- Drop columns
- Data types
- Drop duplicates
- ...
- ...

Pandas DataFrames: Overview

The Pandas DataFrame is the foundation of all Pandas functions.



	color	director_name	num_critic_for_reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
0	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
1	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
2	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
3	Color	Christopher Nolan	813.0	164.0	...	23000.0	8.5	2.35	164000
4	NaN	Doug Walker	NaN	NaN	...	12.0	7.1	NaN	0

Change Data Types

Good resources:

https://stackoverflow.com/questions/15891038/change-data-type-of-columns-in-pandas

https://www.geeksforgeeks.org/change-data-type-for-one-or-more-columns-in-pandas-dataframe/

Select specific data

Selecting Rows of Pandas DataFrames

```
# Retrieve the first five rows in a dataframe
df.head()

# Retrieve the last five rows in a dataframe
df.tail()

# Retrieve a random row in the dataframe
df.sample()
```

Selecting Columns of Pandas DataFrames

```
# Select one column in a dataframe.
df[“some_column”]
```

```
# Select more than one column in a dataframe.
df[[“some_column”,”another_column”]]
```

Filtering Data

Pandas DataFrames: Filtering

Pandas DataFrames can also be filtered by:

- **Location**
    - Retrieves rows based on their index
    - Retrieves columns based on their name
    - Uses ".loc" indexer
- **Condition**
    - Retrieves rows only if they meet a certain condition
    - Uses Boolean comparison and logical operators
    - This is all in addition to the tools we learned in the last section.

Pandas DataFrames: Location-Based Filtering

```
df.loc[row_selection, column_selection]
```

Basic **“.loc”** indexer syntax:

- Always to provide a row; columns are optional
- Rows are selected using their index, while columns are selected using their name
- Rows and columns are separated by a comma
- If specifying more than one row or column, pass them in using a list

Pandas DataFrames: Filtering on Conditions

Conditional-based filtering syntax:

- Always requires the name of dataframe and brackets around the condition
- Best practice: wrap conditions in parenthesis
- Comparison operators: **>**, **>=**, **<**, **=<**, **==**, **!=**
- Logical operators: **&** (and), **|** (or)

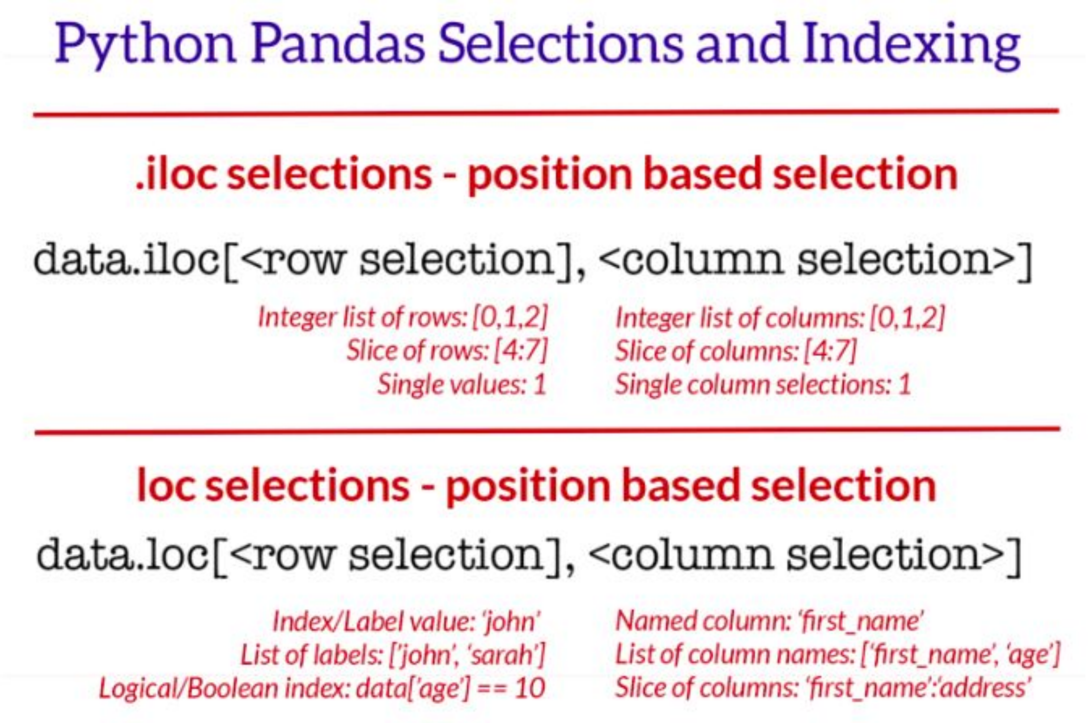
```
df[(df[“some_column”]==some_condition)]
```

```
df[(df[“some_col”]==some_condition) & (df[“other_col”]==other_condition)]
```

.loc() vs .iloc

Good resources:

- https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/
- https://www.pythonprogramming.in/what-is-difference-between-iloc-and-loc-in-pandas.html



Python Pandas Selections and Indexing

.iloc selections - position based selection

data.iloc[<row selection>, <column selection>]

Integer list of rows: [0,1,2]	Integer list of columns: [0,1,2]
Slice of rows: [4:7]	Slice of columns: [4:7]
Single values: 1	Single column selections: 1

loc selections - position based selection

data.loc[<row selection>, <column selection>]

Index/Label value: 'john'	Named column: 'first_name'
List of labels: ['john', 'sarah']	List of column names: ['first_name', 'age']
Logical/Boolean index: data['age'] == 10	Slice of columns: 'first_name':'address'

Order Data

Pandas DataFrames: Ordering

```
# Sort a dataframe by a column in ascending order
df.sort_values(by=”some_column”, ascending=True)

# Sort a dataframe by a column in descending order
df.sort_values(by=”some_column”, ascending=False)

# Sort a dataframe using multiple columns
df.sort_values(by=[“some_column”,”other_column”], ascending=False)
```

Basic EDA and Data Prep

Pandas DataFrames: Basic EDA

```
# Get the number of rows and columns in a dataframe
df.shape

# Get the number of rows in a dataframe
len(df)

# Get the data types and counts for each column in a dataframe
df.info()

# Get summary statistics for all columns in a dataframe
df.describe()
```

Pandas DataFrames: Creating New Data

In any Pandas DataFrame, we can create new columns by:
Adding, subtracting, multiplying, or dividing numeric columns

```
df[“new_col”] = df[“some_col”] arithmetic_operator df[“another_col”]
```

Arithmetic operators:
+ (Addition)
- (Subtraction)
* (Multiplication)
/ (Division)

```
df[“new_col”] = df[“some_col”] + df[“another_col”]

df[“new_col”] = df[“some_col”] * df[“another_col”]
```

Pandas DataFrames: Creating New Data

In any Pandas DataFrame, we can create new columns by:

Concatenating string columns

```
df[“new_col”] = df[“some_col”] + df[“another_col”]
```

Slicing string columns

```
df[“new_col”] = df[“some_col”].str[start_position: end_position]
```

Slicing filters a string column, while concatenating combines multiple string columns.

Pandas DataFrames: Creating New Data

In any Pandas DataFrame, we can create new columns by:

Setting the column equal to a single value

```
df[“new_col”] = some_value
```

This value can be any data type accepted by Pandas:

- String
- Integer
- Float
- Boolean value

```
df[“new_col”] = “Hello! I am a new column.”

df[“new_col”] = 29

df[“new_col”] = False
```

Pandas DataFrames: Creating New Data

In any Pandas DataFrame, we can create new columns by:

Using np.where() to create new values based on specific conditions

```
df[“new_col”] = np.where(
                         df[“some_col”]==some_condition,
                         value_if_true,
                         value_if_false
                         )
```

Similar to CASE statements in SQL

Can use nested np.where() statements to create multiple conditions

Pandas DataFrames: Pivot Tables

We can use Pandas pivot tables to build on the skills we have learned selecting, filtering, analyzing, and creating data using Pandas DataFrames.

```
new_df = pd.pivot_table(
                        df,
                        index=[“some_col”],
                        values=[“other_col”],
                        aggfunc={“other_col”:np.sum}
                     ).reset_index()
```

Pivot Tables vs. GROUP BY

```
new_df = pd.pivot_table(
    df,
    index=[“some_col”],
    values=[“other_col”],
    aggfunc={“other_col”:np.sum}
    ).reset_index()
```

```
SELECT SUM(other_col)
FROM df
GROUP BY some_col
```

- **Index:** Columns to be grouped together; equivalent to GROUP BY function
- **Values:** Columns to which aggregations are applied
- **Aggfunc:** Dictionary, specifies which calculations should be applied to which columns

Renaming Columns of Pandas DataFrames

Sometimes we want to rename certain columns, later in our analysis. To do that, you can use the **.rename()** method.

```
# Rename one or more columns in a dataframe.
df = df.rename(columns={"old_name": "new_name",
                        "old_col_2": "new_col_2"})
```

Identifying and Summarizing Missing Values

First, you need to identify missing values in your dataset.

By default the following values are interpreted by Pandas as NaN:

```
‘’, ‘N/A’, ‘NA’, ‘NULL’, ‘NaN’, ‘n/a’, ‘nan’, ‘null’
```

How did we identify NULLs in our SQL queries?

Identifying and Summarizing Missing Values

First, you need to identify and summarize missing values in your dataset.

We can achieve this using three main functions in Pandas.

```
# To evaluate to True when data is missing
df.isnull()

# Calculate total number of cells with missing data per column
df.isnull().sum()

# To evaluate to False when data is missing
df.notnull()
```

Other Key Pandas Functions for Missing Data

You can also get a count of null values in a specified column using **.value_counts(dropna=False)**

```
# To evaluate to True when data is missing
df['Column_Name'].value_counts(dropna=False)
```

Dropping Columns from DataFrame

Let's take a step back and review how to drop specific columns.

To drop specific column(s) from a pandas dataframe, use **.drop()** method with the parameter **labels=** and **axis=1.**

```
df = df.drop(labels=['Column_Name', 'Column_Name2'], axis = 1)
```

Dropping MISSING VALUES from DataFrame

Drop the missing values using **.dropna()**

Let's look at the docs:

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html

Dropping MISSING VALUES from DataFrame

Drop the missing values using **.dropna()**

Parameters (also known as kwargs -- keyword arguments)

- **how:** This tells us if we want to remove a row if any of the columns have a null, or all of the columns have a null.
- **subset:** We can input an array here, like ['Color', 'Size', 'Weight'], and it will only consider nulls in those columns. This is very useful!
- **inplace:** This is if you want to mutate (change) the source dataframe. Default is False, so it will return a copy of the source dataframe.

```
df.dropna(how='all', subset=['Column_Name'], inplace=True)
```

Filling-In Missing Values

Fill in missing values using **.fillna()**

docs:
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html

```
# Series.fillna
df['Column_Name'].fillna('Unknown')
```

We can fill missing data with a specified value or with the median, average, or mode (most frequently occurring).

Filling-In Missing Values

Fill in missing values using **.fillna()**

```
# Series.fillna
df['Column_Name'].fillna('Unknown')

# DataFrame.fillna
new_df = df.fillna(0)
new_df.head()

# DANGER: fills EVERY NaN in the entire dataframe with 0
```

Filling-In Missing Values

When applying the **.fillna()** method to the **entire** dataframe, we need to pass a dictionary to the value argument.

```
# DataFrame.fillna() with a dictionary
new_df = df.fillna({'ColumnName': 'Value to replace NaN'})
new_df.head()
```