Pandas DataFrame Operations - Selection, Slicing, and Filtering

WE'LL COVER THE FOLLOWING

- 4. Data Selection and Slicing
- 5. Conditional Data Selection and Filtering
- Jupyter Notebook

4. Data Selection and Slicing

We have learned how to get a high-level view of our data and some basic data summaries. Now let's focus on some more interesting DataFrame manipulation techniques, ***performing data selection, slicing, and extraction***. For a clearer understanding, we will look at working with columns and then we will learn how to manipulate DataFrames row-wise.

One important thing to remember here is that although many of the methods can be applied to both DataFrame and Series, these two have different attributes. This means we need to know which type of object we are working with. Otherwise, we can end up with errors.

a. Working With Columns

We can extract a column by using its label (column name) and the **square bracket notation**:

```
genre_col = movies_df['Genre']
```

The above will **return a Series object**. If we want to **obtain a DataFrame object as output instead**, then we need to pass the column name(s) as a list (double square brackets), as shown below:

```
# We can select any column using its label:

# To obtain a Series as output
col_as_series = movies_df['Genre']

# Print the object type and the first 5 rows of the series
print(type(col_as_series))
col_as_series.head()


# To obtain a dataFrame as output
col_as_df = movies_df[['Genre']]

# Print the object type and the first 5 rows of the DF
print(type(col_as_df))
col_as_df.head()
```

```
# To obtain a Series as output
col_as_series = movies_df['Genre']
print(type(col_as_series))

col_as_series.head()
```

```
<class 'pandas.core.series.Series'>

0     Action,Adventure,Sci-Fi
1    Adventure,Mystery,Sci-Fi
2              Horror,Thriller
3     Animation,Comedy,Family
4    Action,Adventure,Fantasy
Name: Genre, dtype: object
```

```
# To obtain a dataFrame as output
col_as_df = movies_df[['Genre']]
print(type(col_as_df))

col_as_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
```

	Genre
0	Action,Adventure,Sci-Fi
1	Adventure,Mystery,Sci-Fi
2	Horror,Thriller
3	Animation,Comedy,Family
4	Action,Adventure,Fantasy

If we want to extract multiple columns, we can simply add additional column names to the list.

```
#Since it's just a list, adding another column name is easy:
extracted_cols = movies_df_title_indexed[['Genre', 'Rating', 'Revenue (Millions)']]

extracted_cols.head()
```

```
#Since it's just a list, adding another column name is easy:
extracted_cols = movies_df_title_indexed[['Genre', 'Rating', 'Revenue (Millions)']]

extracted_cols.head()
```

Title	Genre	Rating	Revenue (Millions)
Guardians of the Galaxy	Action,Adventure,Sci-Fi	8.1	333.13
Prometheus	Adventure,Mystery,Sci-Fi	7.0	126.46
Split	Horror,Thriller	7.3	138.12
Sing	Animation,Comedy,Family	7.2	270.32
Suicide Squad	Action,Adventure,Fantasy	6.2	325.02

Notice the difference when we use the indexed DataFrame (*"movies_df_indexed"*) vs default-indexed one (*"movies_df"*): we have an index on title so in the last snippet, movie titles are getting displayed instead of row numbers.

b. Working With Rows

Now let's look at how to perform slicing by rows. Here we have essentially the following indexers:

- `loc`: the loc attribute allows indexing and slicing that always references the explicit index, i.e., **loc**ates by name. For example, in our DataFrame indexed by title, we will use the title of the movie to select the required row.
- `iloc`: the iloc attribute allows indexing and slicing that always references the implicit Python-style index, i.e., **loc**ates by numerical index. In the case of our DataFrame, we will pass the numerical index of the movie for which we are interested in fetching data.
- `ix`: this is a hybrid of the other two approaches. We will understand this better by looking at some examples.

```
# With loc we give the explicit index. In our case the title, "Guardians of the Galaxy":
gog = movies_df_title_indexed.loc["Guardians of the Galaxy"]

# With iloc we give it the numerical index of "Guardians of the Galaxy":
gog = movies_df_title_indexed.iloc[0]
```

```
# With loc we give the explicit index. In our case the title, "Guardians of the Galaxy":
gog = movies_df_title_indexed.loc["Guardians of the Galaxy"]

gog
```

```
Rank                                                        1
Genre                                      Action,Adventure,Sci-Fi
Description           A group of intergalactic criminals are forced ...
Director                                               James Gunn
Actors                Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...
Year                                                        2014
Runtime (Minutes)                                            121
Rating                                                       8.1
Votes                                                     757074
Revenue (Millions)                                        333.13
Metascore                                                     76
Name: Guardians of the Galaxy, dtype: object
```

```
# With iloc we give it the numerical index of "Guardians of the Galaxy":
gog = movies_df_title_indexed.iloc[0]
gog
```

```
Rank                                                        1
Genre                                      Action,Adventure,Sci-Fi
Description           A group of intergalactic criminals are forced ...
Director                                               James Gunn
Actors                Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...
Year                                                        2014
Runtime (Minutes)                                            121
Rating                                                       8.1
Votes                                                     757074
Revenue (Millions)                                        333.13
Metascore                                                     76
Name: Guardians of the Galaxy, dtype: object
```

We can also get **slices with multiple rows** in the same manner:

```
multiple_rows = movies_df_title_indexed.loc['Guardians of the Galaxy':'Sing']
multiple_rows = movies_df_title_indexed.iloc[0:4]

multiple_rows
```

```
multiple_rows = movies_df_title_indexed.loc['Guardians of the Galaxy':'Sing']
multiple_rows = movies_df_title_indexed.iloc[0:4]

multiple_rows
```

Title	Rank	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2014	121	8.1	757074	333.13	76.0
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	2012	124	7.0	485820	126.46	65.0
Split	3	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James McAvoy, Anya Taylor-Joy, Haley Lu Richar...	2016	117	7.3	157606	138.12	62.0
Sing	4	Animation,Comedy,Family	In a city of humanoid animals, a	Christophe Lourdelet	Matthew McConaughey,Reese	2016	108	7.2	60545	270.32	59.0

If we do not want to select all the columns, we can **specify both rows and columns** at once; the first index refers to rows while the second one (after the coma) to columns:

Remember: the dot notation is *start:step:end*. If we just have something like *:4*, it means the starting point is the *0th* index.

```
# Select all rows uptil 'Sing' and all columns uptil 'Director'
movies_df_title_indexed.loc[:'Sing', :'Director']
movies_df_title_indexed.iloc[:4, :3]
```

```
movies_df_title_indexed.loc[:'Sing', :'Director']
```

Title	Rank	Genre	Description	Director
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott
Split	3	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan
Sing	4	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet

```
movies_df_title_indexed.iloc[:4, :3]
```

Title	Rank	Genre	Description
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...
Split	3	Horror,Thriller	Three girls are kidnapped by a man with a diag...
Sing	4	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...

Now let's look at the hybrid approach, ***ix***. It's just like the other two indexing options, except that we can use a **mix of explicit and implicit indexes**:

```
# Select all rows uptil Sing and all columns uptil Director
movies_df_title_indexed.ix[:'Sing', :4]
movies_df_title_indexed.ix[:4, :'Director']
```

5. Conditional Data Selection and Filtering

We have looked at selecting rows and columns based on specific indices. But what ***if we don't know the index (implicit or explicit)*** of the row that we

what ***if we don't know the index (implicit or explicit)*** of the row that we want to perform data selection or filtering based on some conditions on?

Say we want to filter our movies DataFrame to show **only movies from 2016 or all the movies that had a rating of more than 8.0?**

We can apply boolean conditions to the columns in our DataFrame as follows:

```
# We can easily filter rows using the values of a specific row.
# For example, for geting all our 2016 movies:
movies_df_title_indexed[movies_df_title_indexed['Year'] == 2016]

# All our movies with a rating higher than 8.0
movies_df_title_indexed[movies_df_title_indexed['Rating'] > 8.0 ]
```

Now let's look at some more complex filters. We can make our **conditions richer with logical operators** like "|" and "&".

Say we want to **retrieve the latest movies (movies released between 2010 and 2016) that had a very poor rating (score less than 6.0) but were among the highest earners at the box office (revenue above the 75th percentile)**. We can write our query as follows:

```
movies_df_title_indexed[
    ((movies_df_title_indexed['Year'] >= 2010) & (movies_df_title_indexed['Year'] <= 2016))
    & (movies_df_title_indexed['Rating'] < 6.0)
    & (movies_df_title_indexed['Revenue (Millions)'] > movies_df_title_indexed['Revenue (Mil
]
```

```
movies_df_title_indexed[
    ((movies_df_title_indexed['Year'] >= 2010) & (movies_df_title_indexed['Year'] <= 2016))
    & (movies_df_title_indexed['Rating'] < 6.0)
    & (movies_df_title_indexed['Revenue (Millions)'] > movies_df_title_indexed['Revenue (Millions)'].quantile(0.75))
]
```

Title	Rank	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore
Fifty Shades of Grey	64	Drama,Romance,Thriller	Literature student Anastasia Steele's life cha...	Sam Taylor-Johnson	Dakota Johnson, Jamie Dornan, Jennifer Ehle,El...	2015	125	4.1	244474	166.15	46.0
Ghostbusters	80	Action,Comedy,Fantasy	Following a ghost invasion of Manhattan, paran...	Paul Feig	Melissa McCarthy, Kristen Wiig, Kate McKinnon,...	2016	116	5.3	147717	128.34	60.0
Transformers: Age of Extinction	127	Action,Adventure,Sci-Fi	Autobots must escape sight from a bounty hunte...	Michael Bay	Mark Wahlberg, Nicola Peltz, Jack Reynor, Stan...	2014	165	5.7	255493	245.43	32.0
The Twilight Saga: Breaking Dawn - Part 2	367	Adventure,Drama,Fantasy	After the birth of Renesmee, the Cullens gathe...	Bill Condon	Kristen Stewart, Robert Pattinson, Taylor Laut...	2012	115	5.5	194329	292.30	52.0
Grown Ups 2	395	Comedy	After moving his family back to his hometown t...	Dennis Dugan	Adam Sandler, Kevin James, Chris Rock, David S...	2013	101	5.4	114482	133.67	19.0
Clash of the Titans	576	Action,Adventure,Fantasy	Perseus demigod son of Zeus, battles the mini...	Louis Leterrier	Sam Worthington, Liam Neeson, Ralph Fiennes,Ja...	2010	106	5.8	238206	163.19	39.0

The results tell us that *"Fifty Shades of Grey"* tops the list of movies with the worst reviews but the highest revenues! In total there are 12 movies that match these criteria.

Note that the 75th percentile was given to us earlier by the *.describe()* method (it was *113.715M $*), and these are all movies with revenue above that.

Jupyter Notebook

You can see the instructions running in the Jupyter Notebook below:

> 📝 **How to Use a Jupyter NoteBook?**
>
> - Click on **"Click to Launch"** 🚀 button to work and see the code running live in the notebook.
> - You can click  to open the **Jupyter Notebook in a new tab**.
> - Go to files and click *Download as* and then choose the format of the file to **download** 📥. You can choose Notebook(.ipynb) to download the file and work locally or on your personal Jupyter Notebook.
> - ⚠ The notebook **session expires after 30 minutes of inactivity**. It will reset if there is no interaction with the notebook for 30 consecutive minutes.

Your app can be found at: https://1dgnrwwoynk5m-live-app.educative.run/notebooks/Selection%2CSlicing%26Filtering.ipynb