Pandas DataFrame Operations - Dealing With Missing and Duplicates

WE'LL COVER THE FOLLOWING

8. Dealing With Missing Values

The difference between fake data and real-world data is that real data is rarely clean and homogeneous. One particular issue that we need to tackle when working with real data is that of missing values. And it's not just about values being missing, different data sources can indicate missing values in different ways as well.

The two flavors in which we are likely to encounter missing or null values are:

- ***None***: A Python object that is often used for missing data in Python. *None* can only be used in arrays with data type *'object'* (i.e., arrays of Python objects).
- ***NaN (Not a Number)***: A special floating-point value that is used to represent missing data. A floating-point type means that, unlike with *None*'s object array, we can perform mathematical operations. However, remember that, regardless of the operation, the result of arithmetic with *NaN* will be another *NaN*.

Run the examples in the code widget below to understand the difference between the two. Observe that performing arithmetic operations on the array with the None type throws a **run-time error** while the code executes without errors for NaN:

```
import numpy as np
import pandas as pd

# Example with None
None_example = np.array([0, None, 2, 3])
print("dtype =", None_example.dtype)
print(None_example)

# Example with NaN
NaN_example = np.array([0, np.nan, 2, 3])
print("dtype =", NaN_example.dtype)
print(NaN_example)

# Math operations fail with None but give NaN as output with NaNs
print("Arithmetic Operations")
print("Sum with NaNs:", NaN_example.sum())
print("Sum with None:", None_example.sum())
```

Pandas is built to handle both *NaN* and *None*, and it treats the two as essentially interchangeable for indicating missing or null values. Pandas also provides us with many useful methods for detecting, removing, and replacing null values in Pandas data structures: `isnull()`, `notnull()`, `dropna()`, and `fillna()`. Let's see all of these in action with some demonstrations.

a. Detecting Null Values

`isnull()` and `notnull()` are two useful methods for detecting null data for Pandas data structures. They return a Boolean mask over the data. For example, let's see if there are any movies for which we have some missing data:

```
movies_df_title_indexed.isnull()
```

```
movies_df_title_indexed.isnull()
```

Title	Rank	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore
Guardians of the Galaxy	False	False	False	False	False	False	False	False	False	False	False
Prometheus	False	False	False	False	False	False	False	False	False	False	False
Split	False	False	False	False	False	False	False	False	False	False	False
Sing	False	False	False	False	False	False	False	False	False	False	False
Suicide Squad	False	False	False	False	False	False	False	False	False	False	False
The Great Wall	False	False	False	False	False	False	False	False	False	False	False
La La Land	False	False	False	False	False	False	False	False	False	False	False
Mindhorn	False	False	False	False	False	False	False	False	False	True	False

As we can see from the snippet of the Boolean mask above, *isnull()* returns a DataFrame where each cell is either True or False depending on that cell's missing-value status. For example, we can see that we do not have the revenue information for the movie "Mindhorn".

We can also **count the number of null values** in each column using an aggregate function for summing:

```
movies_df_title_indexed.isnull().sum()
```

```
movies_df_title_indexed.isnull().sum()
```

```
Rank                    0
Genre                   0
Description             0
Director                0
Actors                  0
Year                    0
Runtime (Minutes)       0
Rating                  0
Votes                   0
Revenue (Millions)    128
Metascore              64
dtype: int64
```

Now we know that we do not know the revenue for 128 movies and metascore for 64.

b. Dropping Null Values

Removing null values is very straightforward. However, it is not always the best approach to deal with null values. And here comes the dilemma of **dropping vs imputation**, replacing nulls with some reasonable non-null values.

In general, dropping should only be performed when we have a small amount of null data because we cannot just drop single values from the DataFrame — dropping means ***removing full rows or full columns***.

`dropna()` allows us to very easily drop rows or columns. Whether we should go by rows or columns depends on the dataset at hand; there is no rule here.

- By default, this method will drop all rows in which any null value is present and return a new DataFrame without altering the original one. If we want to modify our original DataFrame inplace instead, we can specify *inplace=True*.
- Alternatively, we can drop all columns containing any null values by specifying *axis=1*.

```
# Drop all rows with any missing data
movies_df_title_indexed.dropna()

# Drop all the columns containing any missing data
movies_df_title_indexed.dropna(axis=1)
```

What does dropping data mean for our IMDB dataset?

- Dropping rows would remove 128 rows where revenue is null and 64 rows where metascore is null. This is quite some data loss since there's perfectly good data in the other columns of those dropped rows!
- Dropping columns would remove the revenue and metascore columns — not a smart move either!

To avoid losing all this good data, we can also choose to drop rows or columns based on a threshold parameter, drop only if the majority of data is missing. This can be specified using the ***how*** or ***thresh*** parameters, which allow fine control of the number of nulls to allow in through the DataFrame:

```
# Drop columns where all the values are missing
df.dropna(axis='columns', how='all')

# Thresh to specify a minimum number of non-null values
# for the row/column to be kept
df.dropna(axis='rows', thresh=10)
```

c. Imputation (Filling Null Values)

As we have just seen, dropping rows or columns with missing data can result in a losing a significant amount of interesting data. So often, rather than dropping data, we replace missing values with a valid value. This new value can be a single number, like zero, or it can be some sort of imputation or interpolation from the good values, like the mean or the median value of that column. For doing this, Pandas provides us with the very handy `fillna()` method for doing this.

For example, let's impute the missing values for the revenue column using the mean revenue:

```
# Getting the mean value for the column:
revenue = movies_df_title_indexed['Revenue (Millions)']
revenue_mean = revenue.mean()

print("Mean Revenue:", revenue_mean)

# Let's fill the nulls with the mean value:
revenue.fillna(revenue_mean, inplace=True)

# Let's get the updated status of our DataFrame:
movies_df_title_indexed.isnull().sum()
```

```
# Getting the mean value for the column:
revenue = movies_df_indexed['Revenue (Millions)']
revenue_mean = revenue.mean()

revenue_mean
```

```
82.95637614678897
```

```
# Let's fill the nulls with the mean value:
revenue.fillna(revenue_mean, inplace=True)

# Let's get the updated status of our DataFrame:
movies_df_indexed.isnull().sum()
```

```
Rank                    0
Genre                   0
Description             0
Director                0
Actors                  0
Year                    0
Runtime (Minutes)       0
Rating                  0
Votes                   0
Revenue (Millions)      0
Metascore              64
dtype: int64
```

We have now replaced all the missing values for revenue with the mean of the column, and as we can observe from the output, by using *inplace=True* we have modified the original DataFrame — it has no more nulls for revenue.

> *Note: While computing the mean, the aggregate operation did not fail, even if we had missing values, because the dataset has missing revenues denoted by NaN, as shown in the snippet below:*

```
movies_df_title_indexed.loc['Mindhorn']
```

```
Rank                                                            8
Genre                                                      Comedy
Description          A has-been actor best known for playing the ti...
Director                                                Sean Foley
Actors               Essie Davis, Andrea Riseborough, Julian Barrat...
Year                                                         2016
Runtime (Minutes)                                              89
Rating                                                        6.4
Votes                                                        2490
Revenue (Millions)                                            NaN
Metascore                                                      71
Name: Mindhorn, dtype: object
```

Revenue value before imputation was NaN

This was a very simple way of imputing values. Instead of replacing nulls with the mean of the entire column, a smarter approach could have been to be more fine-grained — we could have replaced the null values with the mean revenue specifically for the genre of that movie, instead of the mean for all the movies.

9. Handling Duplicates

We do not have duplicate rows in our movies dataset, but this is not always the case. If we do have duplicates, we want to make sure that we are not performing computations, like getting the total revenue per director, based on duplicate data.

Pandas allows us to very easily remove duplicates by using the `drop_duplicates()` method. This method returns a copy of the DataFrame with duplicates removed unless we choose to specify ***inplace=True***, just like for the

previously seen methods.

> *Note: It's a good practice to use* ***.shape*** *to confirm the change in number of rows after the drop_duplicates() method has been run.*

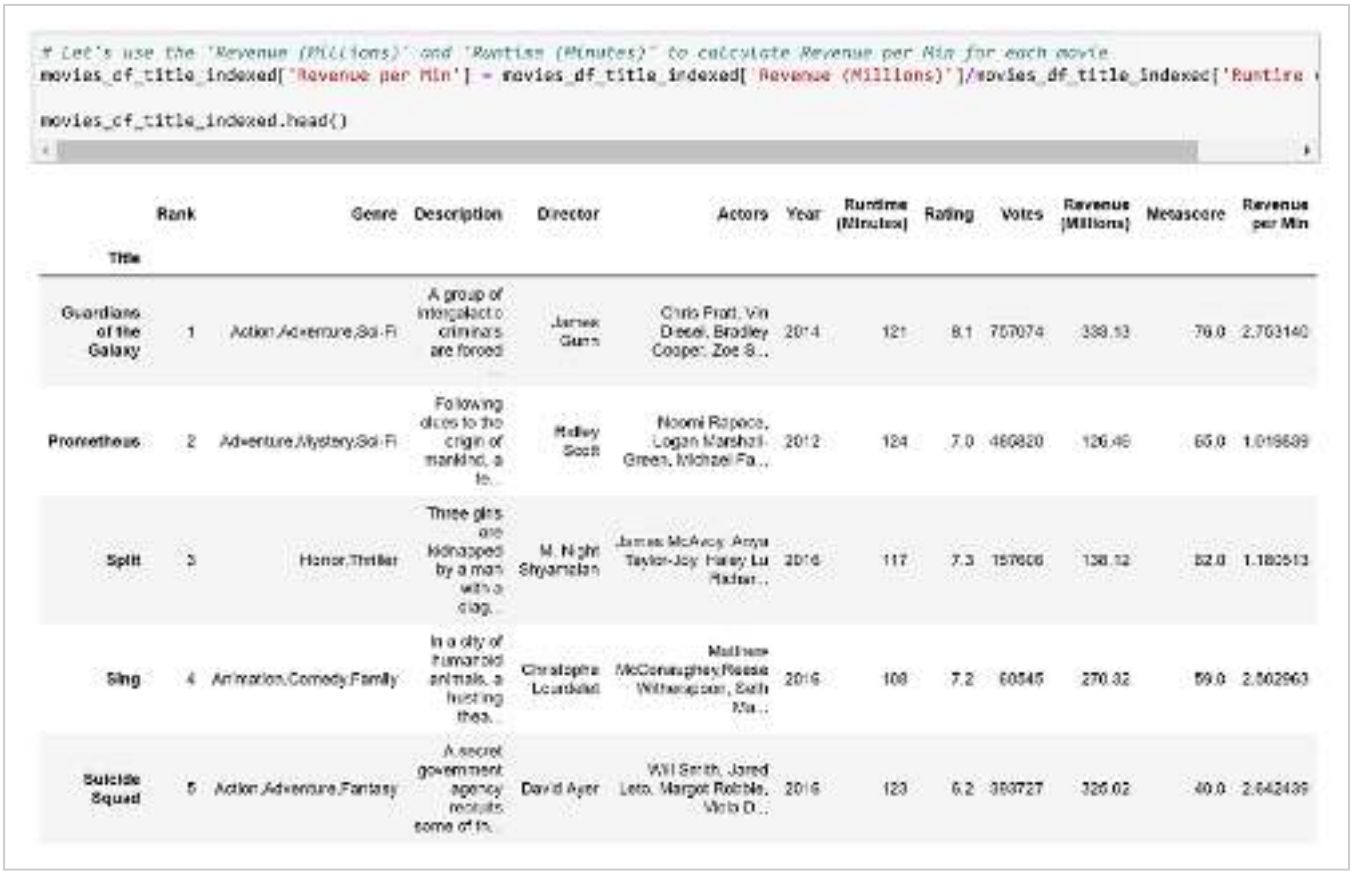
10. Creating New Columns From Existing Columns

Often while analyzing data, we find ourselves needing to create new columns from existing ones. Pandas makes this a breeze!

Say we want to introduce a new column in our DataFrame that has **revenue per minute for each movie**. We can divide the revenue by the runtime and create this new column very easily like so:

```
# We can use 'Revenue (Millions)' and 'Runtime (Minutes)' to calculate Revenue per Min for ea
movies_df_title_indexed['Revenue per Min'] =
movies_df_title_indexed['Revenue (Millions)']/movies_df_title_indexed['Runtime (Minutes)']

movies_df_title_indexed.head()
```



Title	Rank	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore	Revenue per Min
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	2.753140
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	1.019839
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	1.180513
Sing	4	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaughey,Reese Witherspoon, Seth Ma...	2016	108	7.2	60545	270.32	59.0	2.502963
Suicide Squad	5	Action,Adventure,Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith, Jared Leto, Margot Robbie, Viola D...	2016	123	6.2	393727	325.02	40.0	2.642439

From the snippet above, we can see that we have a new column at the end of the DataFrame with revenue per minute for each movie. This is not necessarily useful information, it was just an example to demonstrate how to create new columns based on existing data.

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/DealMissing%26DuplicateValues.ipynb