

Pandas tutorial.

- URL: <https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/> (<https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/>)

Prepared by: Sagun Shakya

- GITAM Institute of Science.

```
In [1]: import pandas as pd
```

Creating a dataframe.

Using a dictionary:

```
In [2]: data = dict()

data['apples'] = [0,1,2,3]
data['oranges'] = [9,4,3,5]
```

```
In [3]: purchases = pd.DataFrame(data)
print(purchases)
```

	apples	oranges
0	0	9
1	1	4
2	2	3
3	3	5

Creating our own index.

```
In [4]: purchases = pd.DataFrame(data, index = ['Robert', 'Sagun', 'Alice', 'Ben'])
print(purchases)
```

	apples	oranges
Robert	0	9
Sagun	1	4
Alice	2	3
Ben	3	5

Locating a person's orders using their name as a key.

```
In [62]: purchases.loc['Sagun':'Ben']
```

```
Out[62]:
```

	apples	oranges
Sagun	1	4
Alice	2	3
Ben	3	5

```
In [61]: purchases.iloc[1:3]
```

```
Out[61]:
```

	apples	oranges
Sagun	1	4
Alice	2	3

Reading data from .csv files.

- Make sure that the .csv file is stored inside the folder where your current .ipynb file is located.
- If the .csv file is in another location, we can import os module and change the directory:
 - import os
 - os.chdir('File Path')

```
In [6]: df = pd.read_csv('Toyota.csv')

#df = pd.read_csv('Toyota.csv', index_col = 0)
#To exclude the index column which is set into the dataframe by default.
```

Reading data from json files.

```
In [7]: #df = pd.read_json('purchases.json')
```

Converting the file to a particular format.

- After doing the task of data cleaning extensively, we can convert the newly created dataframe into a new file (.csv, .json, sql and so on).

```
In [8]: #df.to_csv('purchases_new.csv')
#df.to_json('purchases_new.json')
#df.to_sql('purchases_new', con)
```

DataFrame Operations.

We will use IMDB movies list as our dataframe.

```
In [9]: movies = pd.read_csv('imdb.csv')
```

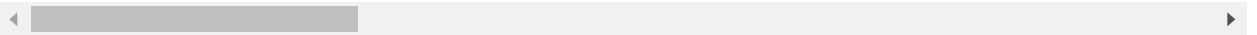
Checking few tuples for reference and better understanding of the dataset.

```
In [10]: movies.head(10)
```

```
Out[10]:
```

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook
0	Color	James Cameron	723.0	178.0	0.0	
1	Color	Gore Verbinski	302.0	169.0	563.0	
2	Color	Sam Mendes	602.0	148.0	0.0	
3	Color	Christopher Nolan	813.0	164.0	22000.0	2:
4	NaN	Doug Walker	NaN	NaN	131.0	
5	Color	Andrew Stanton	462.0	132.0	475.0	
6	Color	Sam Raimi	392.0	156.0	0.0	,
7	Color	Nathan Greno	324.0	100.0	15.0	
8	Color	Joss Whedon	635.0	141.0	0.0	1!
9	Color	David Yates	375.0	153.0	282.0	10

10 rows × 28 columns



Setting the movie_title as index column.

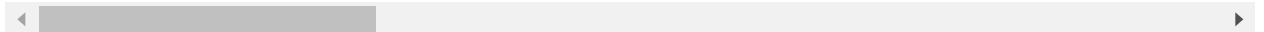
```
In [52]: movies = pd.read_csv('imdb.csv', index_col = 'movie_title')
movies.head()

#For viewing the last 'n' rows, we use .tail(n)
```

Out[52]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3.
movie_title						
Avatar	Color	James Cameron	723.0	178.0	0.0	
Pirates of the Caribbean: At World's End	Color	Gore Verbinski	302.0	169.0	563.0	
Spectre	Color	Sam Mendes	602.0	148.0	0.0	
The Dark Knight Rises	Color	Christopher Nolan	813.0	164.0	22000.0	
Star Wars: Episode VII - The Force Awakens	NaN	Doug Walker	NaN	NaN	131.0	

5 rows × 7 columns



Getting to know our data a little better.

```
In [12]: movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5043 entries, Avatar to My Date with Drew
Data columns (total 27 columns):
color                    5024 non-null object
director_name           4939 non-null object
num_critic_for_reviews  4993 non-null float64
duration                5028 non-null float64
director_facebook_likes 4939 non-null float64
actor_3_facebook_likes  5020 non-null float64
actor_2_name            5030 non-null object
actor_1_facebook_likes  5036 non-null float64
gross                   4159 non-null float64
genres                  5043 non-null object
actor_1_name            5036 non-null object
num_voted_users         5043 non-null int64
cast_total_facebook_likes 5043 non-null int64
actor_3_name            5020 non-null object
facenumber_in_poster    5030 non-null float64
plot_keywords           4890 non-null object
movie_imdb_link          5043 non-null object
num_user_for_reviews    5022 non-null float64
language                5031 non-null object
country                 5038 non-null object
content_rating           4740 non-null object
budget                  4551 non-null float64
title_year              4935 non-null float64
actor_2_facebook_likes  5030 non-null float64
imdb_score              5043 non-null float64
aspect_ratio            4714 non-null float64
movie_facebook_likes     5043 non-null int64
dtypes: float64(13), int64(3), object(11)
memory usage: 1.1+ MB
```

Checking the dimension of our dataframe.

```
In [13]: movies.shape
```

```
Out[13]: (5043, 27)
```

Handling duplicate rows.

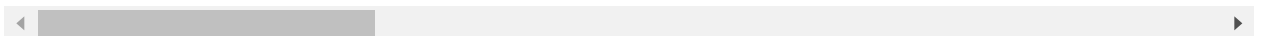
- If two rows are the same, pandas will drop the second row and **keep** the first row.

In [14]: `movies.drop_duplicates()`

Out[14]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3
movie_title						
Avatar	Color	James Cameron	723.0	178.0	0.0	
Pirates of the Caribbean: At World's End	Color	Gore Verbinski	302.0	169.0	563.0	
Spectre	Color	Sam Mendes	602.0	148.0	0.0	
The Dark Knight Rises	Color	Christopher Nolan	813.0	164.0	22000.0	
Star Wars: Episode VII - The Force Awakens	NaN	Doug Walker	NaN	NaN	131.0	
...
Signed Sealed Delivered	Color	Scott Smith	1.0	87.0	2.0	
The Following	Color	NaN	43.0	43.0	NaN	
A Plague So Pleasant	Color	Benjamin Roberds	13.0	76.0	0.0	
Shanghai Calling	Color	Daniel Hsia	14.0	100.0	0.0	
My Date with Drew	Color	Jon Gunn	43.0	90.0	16.0	

4998 rows × 27 columns



In [15]: `movies.shape`

Out[15]: (5043, 27)

The dimension remains unaffected. This means there are no duplicate rows.

Viewing the column names of the dataframe.

```
In [16]: movies.columns
```

```
Out[16]: Index(['color', 'director_name', 'num_critic_for_reviews', 'duration',
               'director_facebook_likes', 'actor_3_facebook_likes', 'actor_2_name',
               'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
               'num_voted_users', 'cast_total_facebook_likes', 'actor_3_name',
               'facenumber_in_poster', 'plot_keywords', 'movie_imdb_link',
               'num_user_for_reviews', 'language', 'country', 'content_rating',
               'budget', 'title_year', 'actor_2_facebook_likes', 'imdb_score',
               'aspect_ratio', 'movie_facebook_likes'],
              dtype='object')
```

Renaming the column name.

- from 'actor_1_name' to 'primary_actor'.
- from 'actor_2_name' to 'secondary_actor'.

```
In [17]: movies.rename(columns = { 'actor_1_name' : 'primary_actor', 'actor_2_name' : 'secondary_actor' }, inplace=True)
movies.columns
```

```
Out[17]: Index(['color', 'director_name', 'num_critic_for_reviews', 'duration',
               'director_facebook_likes', 'actor_3_facebook_likes', 'secondary_actor',
               'actor_1_facebook_likes', 'gross', 'genres', 'primary_actor',
               'num_voted_users', 'cast_total_facebook_likes', 'actor_3_name',
               'facenumber_in_poster', 'plot_keywords', 'movie_imdb_link',
               'num_user_for_reviews', 'language', 'country', 'content_rating',
               'budget', 'title_year', 'actor_2_facebook_likes', 'imdb_score',
               'aspect_ratio', 'movie_facebook_likes'],
              dtype='object')
```

Working out the missing values.

- Two ways to deal with null values:
 - Eliminating the whole row/column containing the null value.
 - Filling in the void with a non - null value (a technique called **imputation**).

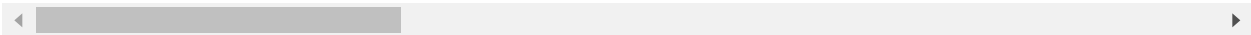
In [19]:

movies.isnull()

Out[19]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3
movie_title						
Avatar	False	False	False	False	False	
Pirates of the Caribbean: At World's End	False	False	False	False	False	
Spectre	False	False	False	False	False	
The Dark Knight Rises	False	False	False	False	False	
Star Wars: Episode VII - The Force Awakens	True	False	True	True	False	
...
Signed Sealed Delivered	False	False	False	False	False	
The Following	False	True	False	False	True	
A Plague So Pleasant	False	False	False	False	False	
Shanghai Calling	False	False	False	False	False	
My Date with Drew	False	False	False	False	False	

5043 rows × 27 columns




```
In [20]: # A more concise way.  
movies.isnull().sum()
```

```
Out[20]: color                                19  
director_name                             104  
num_critic_for_reviews                     50  
duration                                  15  
director_facebook_likes                   104  
actor_3_facebook_likes                    23  
secondary_actor                           13  
actor_1_facebook_likes                     7  
gross                                    884  
genres                                    0  
primary_actor                             7  
num_voted_users                           0  
cast_total_facebook_likes                 0  
actor_3_name                              23  
facenumber_in_poster                     13  
plot_keywords                             153  
movie_imdb_link                           0  
num_user_for_reviews                      21  
language                                  12  
country                                    5  
content_rating                            303  
budget                                    492  
title_year                                108  
actor_2_facebook_likes                    13  
imdb_score                                 0  
aspect_ratio                              329  
movie_facebook_likes                       0  
dtype: int64
```

Dropping rows is useful when the null values exist in small numbers.

- `movies.dropna()`
- `movies.dropna(axis = 1)` (For dropping column.)

Imputation

- Filling in the null value with mean or a median value.

```
In [23]: grossing = movies['gross']      #Dictionary key - value pairs.

grossing.head()
```

```
Out[23]: movie_title
Avatar                                760505847.0
Pirates of the Caribbean: At World's End  309404152.0
Spectre                               200074175.0
The Dark Knight Rises                  448130642.0
Star Wars: Episode VII - The Force Awakens      NaN
Name: gross, dtype: float64
```

```
In [24]: grossing.isnull().sum()
```

```
Out[24]: 884
```

```
In [25]: grossing_mean = grossing.mean()
print(round(grossing_mean, 2))
```

```
48468407.53
```

Filling the null values with the mean.

```
In [26]: grossing.fillna(grossing_mean, inplace=True)
```

```
In [27]: movies.isnull().sum()
```

```
Out[27]: color                                19
director_name                             104
num_critic_for_reviews                     50
duration                                  15
director_facebook_likes                   104
actor_3_facebook_likes                     23
secondary_actor                           13
actor_1_facebook_likes                     7
gross                                      0
genres                                    0
primary_actor                             7
num_voted_users                           0
cast_total_facebook_likes                  0
actor_3_name                              23
facenumber_in_poster                      13
plot_keywords                             153
movie_imdb_link                            0
num_user_for_reviews                      21
language                                  12
country                                    5
content_rating                             303
budget                                    492
title_year                                108
actor_2_facebook_likes                     13
imdb_score                                 0
aspect_ratio                              329
movie_facebook_likes                       0
dtype: int64
```

Imputing an entire column with the same value like this is a basic example.

Better to try a more granular approach.

- Find the mean of the revenue generated in each genre individually and impute the nulls in each genre with that genre's mean.

Statistical summary of the entire dataframe using describe().

In [30]: `movies.describe()`

Out[30]:

book_likes	actor_3_facebook_likes	actor_1_facebook_likes	gross	num_voted_users	cast_to
4939.000000	5020.000000	5036.000000	5.043000e+03	5.043000e+03	
686.509212	645.009761	6560.047061	4.846841e+07	8.366816e+04	
2813.328607	1665.041728	15020.759120	6.216318e+07	1.384853e+05	
0.000000	0.000000	0.000000	1.620000e+02	5.000000e+00	
7.000000	133.000000	614.000000	8.460992e+06	8.593500e+03	
49.000000	371.500000	988.000000	3.743230e+07	3.435900e+04	
194.500000	636.000000	11000.000000	5.135707e+07	9.630900e+04	
3000.000000	23000.000000	640000.000000	7.605058e+08	1.689764e+06	

In [29]: `movies['gross'].describe()`

Out[29]:

```
count    5.043000e+03
mean     4.846841e+07
std      6.216318e+07
min      1.620000e+02
25%      8.460992e+06
50%      3.743230e+07
75%      5.135707e+07
max      7.605058e+08
Name: gross, dtype: float64
```

In [33]: `movies['genres'].describe()`

Out[33]:

```
count      5043
unique      914
top        Drama
freq       236
Name: genres, dtype: object
```

In [35]: `movies['genres'].value_counts().head(7)`

Out[35]:

```
Drama      236
Comedy     209
Comedy|Drama  191
Comedy|Drama|Romance  187
Comedy|Romance  158
Drama|Romance  152
Crime|Drama|Thriller  101
Name: genres, dtype: int64
```

Relationship between continuous variables (correlation).

In [36]: `movies.corr()`

Out[36]:

	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes
num_critic_for_reviews	1.000000	0.258486	0.180674	0.2
duration	0.258486	1.000000	0.173296	0.1
director_facebook_likes	0.180674	0.173296	1.000000	0.1
actor_3_facebook_likes	0.271646	0.123558	0.120199	1.0
actor_1_facebook_likes	0.190016	0.088449	0.090723	0.2
gross	0.442045	0.204998	0.139254	0.3
num_voted_users	0.624943	0.314765	0.297057	0.2
cast_total_facebook_likes	0.263203	0.123074	0.119549	0.4
facenumber_in_poster	-0.033897	0.013469	-0.041268	0.0
num_user_for_reviews	0.609387	0.328403	0.221890	0.2
budget	0.119994	0.074276	0.021090	0.0
title_year	0.275707	-0.135038	-0.063820	0.0
actor_2_facebook_likes	0.282306	0.131673	0.119601	0.5
imdb_score	0.305303	0.261662	0.170802	0.0
aspect_ratio	-0.049786	-0.090071	0.001642	-0.0
movie_facebook_likes	0.683176	0.196605	0.162048	0.2

DataFrame slicing, selecting, extracting.

Type of attribute.

In [37]: `genre = movies['genres']`
`type(genre)` *# Series.*

Out[37]: `pandas.core.series.Series`

In [40]: `genre_1 = movies[['genres']]` *#Should be a list.*
`type(genre_1)` *#DataFrame.*

Out[40]: `pandas.core.frame.DataFrame`

In [45]: `sub = movies[['genres', 'gross', 'imdb_score']]`
`type(sub)` *#DataFrame.*

Out[45]: `pandas.core.frame.DataFrame`

```
In [48]: sub.head()
```

```
Out[48]:
```

	genres	gross	imdb_score
movie_title			
Avatar	Action Adventure Fantasy Sci-Fi	7.605058e+08	7.9
Pirates of the Caribbean: At World's End	Action Adventure Fantasy	3.094042e+08	7.1
Spectre	Action Adventure Thriller	2.000742e+08	6.8
The Dark Knight Rises	Action Thriller	4.481306e+08	8.5
Star Wars: Episode VII - The Force Awakens	Documentary	4.846841e+07	7.1

Getting data by row.

-Two Options:

- `.loc` = locates by name.
- `.iloc` = locates by numerical index.

```
In [58]: ava = movies.iloc[1]
ava
```

```
Out[58]: color                                Color
director_name                                Gore Verbinski
num_critic_for_reviews                        302
duration                                      169
director_facebook_likes                       563
actor_3_facebook_likes                        1000
actor_2_name                                  Orlando Bloom
actor_1_facebook_likes                        40000
gross                                          3.09404e+08
genres                                         Action|Adventure|Fantasy
actor_1_name                                  Johnny Depp
num_voted_users                               471220
cast_total_facebook_likes                     48350
actor_3_name                                  Jack Davenport
facenumber_in_poster                          0
plot_keywords                                goddess|marriage ceremony|marriage proposal|pi...
movie_imdb_link                               http://www.imdb.com/title/tt0449088/?ref=fn_t...
(http://www.imdb.com/title/tt0449088/?ref=fn_t...)
num_user_for_reviews                          1238
language                                       English
country                                       USA
content_rating                                PG-13
budget                                         3e+08
title_year                                    2007
actor_2_facebook_likes                        5000
imdb_score                                    7.1
aspect_ratio                                  2.35
movie_facebook_likes                          0
Name: Pirates of the Caribbean: At World's End , dtype: object
```

```
In [75]: sub = movies[['title_year', 'actor_1_name', 'genres', 'imdb_score', 'gross']]
sub.iloc[5:10]
```

```
Out[75]:
```

	title_year	actor_1_name	genres	imdb_score	
movie_title					
John Carter	2012.0	Daryl Sabara	Action Adventure Sci-Fi	6.6	
Spider-Man 3	2007.0	J.K. Simmons	Action Adventure Romance	6.2	3
Tangled	2010.0	Brad Garrett	Adventure Animation Comedy Family Fantasy Musi...	7.8	2
Avengers: Age of Ultron	2015.0	Chris Hemsworth	Action Adventure Sci-Fi	7.5	4
Harry Potter and the Half- Blood Prince	2009.0	Alan Rickman	Adventure Family Fantasy Mystery	7.5	3

Conditional Selection.

Finding the movies which were made in 2015.

```
In [77]: condition = (sub['title_year'] == 2015)
condition.head(10)
```

```
#This method is quite messy.
```

```
Out[77]: movie_title
Avatar                                     False
Pirates of the Caribbean: At World's End   False
Spectre                                    True
The Dark Knight Rises                      False
Star Wars: Episode VII - The Force Awakens False
John Carter                               False
Spider-Man 3                              False
Tangled                                   False
Avengers: Age of Ultron                    True
Harry Potter and the Half-Blood Prince    False
Name: title_year, dtype: bool
```



```
In [79]: # Filtering out all the movies which were not made in 2015.

sub[ sub['title_year'] == 2015]

#Read like "Select ALL from 'sub' where 'sub title_year' equals 2015."
```

Out[79]:

	title_year	actor_1_name	genres	imdb_score	g
movie_title					
Spectre	2015.0	Christoph Waltz	Action Adventure Thriller	6.8	2000741
Avengers: Age of Ultron	2015.0	Chris Hemsworth	Action Adventure Sci-Fi	7.5	4589915
Jurassic World	2015.0	Bryce Dallas Howard	Action Adventure Sci-Fi Thriller	7.0	6521772
Furious 7	2015.0	Jason Statham	Action Crime Thriller	7.2	350034
The Good Dinosaur	2015.0	A.J. Buckley	Adventure Animation Comedy Family Fantasy	6.8	1230703
...
The Gallows	2015.0	Pfeifer Brown	Horror Thriller	4.2	227578
Queen Crab	2015.0	Michelle Simone Miller	Action Sci-Fi Thriller	4.5	
Counting	2015.0	NaN	Documentary	6.0	
Dutch Kills	2015.0	Tjasa Ferme	Crime Drama Thriller	4.8	
Exeter	2015.0	Ashley Tramonte	Horror Mystery Thriller	4.6	

226 rows × 5 columns



Finding the movies which have rating at least 8.5.

```
In [89]: good_movies = sub[sub['imdb_score'] >= 8.5]
good_movies.sort_values(by = 'imdb_score', ascending = False)['imdb_score']
```

```
Out[89]: movie_title
Towering Inferno          9.5
The Shawshank Redemption  9.3
The Godfather             9.2
Dekalog                   9.1
Dekalog                   9.1
...
Entourage                 8.5
Raiders of the Lost Ark   8.5
Psycho                    8.5
Alien                     8.5
Back to the Future        8.5
Name: imdb_score, Length: 73, dtype: float64
```

- Using Logical operators like | for "OR" and & for "AND".

Finding top 10 movies which were released in 2016 and have rating greater than or equal to 8.5.

```
In [95]: good_movies_2015 = sub[ (sub['imdb_score'] >= 8.5) & (sub['title_year'] == 2016)
good_movies_2015.sort_values(by = 'imdb_score', ascending = False)[ ['imdb_score
```

```
Out[95]:
```

	imdb_score	title_year
movie_title		
Kickboxer: Vengeance	9.1	2016.0
A Beginner's Guide to Snuff	8.7	2016.0
Airlift	8.5	2016.0

Applying functions into the dataframe.

- We will categorize the movies as follows:
- Movies with rating
 - greater than or equal to 8.0 is 'good'.
 - greater than or equal to 7.0 and less than 8.0 is 'mediocre'.
 - less than 7.0 is 'bad'.

```
In [96]: def rating_func(score):
        if score >=8.0:
            return 'Good'
        elif score >= 7.0 and score < 8.0:
            return 'Mediocre'
        elif score <7.0:
            return 'Bad'
```

```
In [102]: movies['rating_category'] = movies['imdb_score'].apply(rating_func)

movies[['imdb_score', 'rating_category']].head(10)
```

Out[102]:

	imdb_score	rating_category
movie_title		
Avatar	7.9	Mediocre
Pirates of the Caribbean: At World's End	7.1	Mediocre
Spectre	6.8	Bad
The Dark Knight Rises	8.5	Good
Star Wars: Episode VII - The Force Awakens	7.1	Mediocre
John Carter	6.6	Bad
Spider-Man 3	6.2	Bad
Tangled	7.8	Mediocre
Avengers: Age of Ultron	7.5	Mediocre
Harry Potter and the Half-Blood Prince	7.5	Mediocre

End of Chapter.

Next: plotting tips using seaborn and matplotlib.

Prepared by: Sagun Shakya

- GITAM Institute of Science