Pandas Basics

August 1, 2020

0.1 Pandas Basics

0.1.1 Created by Shridhar Galande

- The pandas package is the most important tool at the disposal of Data Scientists and Analysts working in Python today.
- Pandas is built on top of the NumPy package, meaning a lot of the structure of NumPy is used or replicated in Pandas.
- Data in pandas is often used to feed statistical analysis in SciPy, plotting functions from Matplotlib, and machine learning algorithms in Scikit-learn.

Install and import !pip install pandas

- The ! at the beginning runs cells as if they were in a terminal.
- To import pandas we usually import it with a shorter name since it's used so much:

import pandas as pd

[116]:
```
# pandas version
pandas.__version__
```

[116]:
```
'0.25.3'
```

0.1.2 Core components of pandas: Series and DataFrames

- The primary two components of pandas are the Series and DataFrame.
- A Series is essentially a column, and a DataFrame is a multi-dimensional table made up of a collection of Series.
- DataFrames and Series are quite similar in that many operations that you can do with one you can do with the other, such as filling in null values and calculating the mean.

0.1.3 Creating DataFrames

1. using dict

[1]:
```
# Create dictionary
data = {
    'apples': [3, 2, 0, 1],
```

```
    'oranges': [0, 3, 7, 2]
}

# Convert to Data frame
df = pd.DataFrame(data)
# Print the dataframe
df
```

```
<IPython.core.display.Javascript object>
```

[1]:

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

Creates a indexes DataFrame using arrays

[2]:
```
df=pd.DataFrame(data, index=['June', 'Robert', 'Lily', 'David'])
df
```

```
<IPython.core.display.Javascript object>
```

[2]:

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

2. Creating Pandas DataFrame from lists of lists.

[3]:
```
# initialize list of lists
data = [['tom', 10], ['nick', 15], ['juli', 14]]

# Create the pandas DataFrame
df = pd.DataFrame(data, columns = ['Name', 'Age'])

# print dataframe.
df
```

```
<IPython.core.display.Javascript object>
```

[3]:

	Name	Age
0	tom	10

```
1  nick   15
2  juli   14
```

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

[ ]:
```
pd.DataFrame
```

3. Creating Dataframe from list of dicts

[4]:
```
# Initialise data to lists.
data = [{'a': 1, 'b': 2, 'c':3}, {'a':10, 'b': 20, 'c': 30}]

# Creates DataFrame.
df = pd.DataFrame(data)

# Print the data
df
```

```
<IPython.core.display.Javascript object>
```

[4]:
```
    a   b   c
0   1   2   3
1  10  20  30
```

[9]:
```
df=pd.DataFrame(np.random.randint(0,10,size=(100,4)),columns=list('ABCD'))
df
```

[9]:
```
     A  B  C  D
0    6  5  3  4
1    5  6  4  4
2    2  2  3  0
3    7  6  0  2
4    9  8  5  0
..  .. .. .. ..
95   5  4  5  8
96   0  4  9  8
97   7  0  0  4
98   2  1  0  1
99   5  2  8  0

[100 rows x 4 columns]
```

4. Creating DataFrame using zip() function

[5]:
```
# List1
Name = ['tom', 'krish', 'nick', 'juli']

# List2
Age = [25, 30, 26, 22]
```

```
# pandas Dataframe.
df = pd.DataFrame(list(zip(Name, Age)), columns = ['Name', 'Age'])

# Print data.
df
```

```
<IPython.core.display.Javascript object>
```

[5]:

	Name	Age
0	tom	25
1	krish	30
2	nick	26
3	juli	22

5. Creating DataFrame from Dicts of series.

[6]:
```
# Intialise data to Dicts of series.
d = {'one' : pd.Series([10, 20, 30, 40], index =['a', 'b', 'c', 'd']),
     'two' : pd.Series([10, 20, 30, 40], index =['a', 'b', 'c', 'd'])}

# creates Dataframe.
df = pd.DataFrame(d)

# print the data.
df
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

[6]:

	one	two
a	10	10
b	20	20
c	30	30
d	40	40

0.1.4 Reading data from CSVs

[ ]:
```
pd.read_csv
```

df = pd.read$_{c}$sv('*purchases.csv*')

df

image.png

CSVs don't have indexes like our DataFrames, so all we need to do is just designate the index_col when reading: df = pd.read$_{c}sv('purchases.csv', index_col = 0)$

df

image.png

0.1.5 Reading data from JSON

If you have a JSON file — which is essentially a stored Python dict — pandas can read this just as easily: df = pd.read$_{j}son('purchases.json')$

df

image.png

0.1.6 Reading data from Excel

df= pd.read_excel('puchase.xlsx')

df

0.1.7 Reading data from a SQL database

If you're working with data from a SQL database you need to first establish a connection using an appropriate Python library, then pass a query to pandas. Here we'll use SQLite to demonstrate.

First, we need *pysqlite3* installed, so run this command in your terminal !pip install pysqlite3 **sqlite3** is used to create a connection to a database which we can then use to generate a DataFrame through a **SELECT** query.

So first we'll make a connection to a SQLite database file: import sqlite3

con = sqlite3.connect("database.db") In this SQLite database we have a table called purchases, and our index is in a column called "index".

By passing a SELECT query and our con, we can read from the purchases table: df = pd.read$_{s}ql_{q}uery("SELECT * FROM purchases", con)$

df

image.png

we could pass index_col='index', but we can also set an index after-the-fact: df = df.set$_{i}ndex('index')$

df

image.png

0.1.8 Converting back to a CSV, JSON, or SQL

df.to$_{c}$sv('new$_{p}$urchases.csv')
df.to$_{j}$son('new$_{p}$urchases.json')
df.to$_{e}$xcel('new$_{p}$urchase.xlsx')
df.to$_{s}$ql('new$_{p}$urchases', con)

0.1.9 Important DataFrame operations

We're loading this dataset from a CSV and designating the movie titles to be our index. https://github.com/LearnDataSci/article-resources/tree/master/Python

[1]:
```
data=pd.read_csv("IMDB-Movie-Data.csv", index_col="Title")
```

```
<IPython.core.display.Javascript object>
```

[2]:
```
data.head(2)
```

[2]:
```
                        Rank                    Genre  \
Title
Guardians of the Galaxy    1   Action,Adventure,Sci-Fi
Prometheus                 2  Adventure,Mystery,Sci-Fi

                                                    Description  \
Title
Guardians of the Galaxy  A group of intergalactic criminals are forced ...
Prometheus               Following clues to the origin of mankind, a te...

                            Director  \
Title
Guardians of the Galaxy    James Gunn
Prometheus               Ridley Scott

                                                         Actors  \
Title
Guardians of the Galaxy  Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...
Prometheus               Noomi Rapace, Logan Marshall-Green, Michael Fa...

                        Year  Runtime (Minutes)  Rating   Votes  \
Title
Guardians of the Galaxy  2014                121     8.1  757074
Prometheus               2012                124     7.0  485820

                        Revenue (Millions)  Metascore
Title
Guardians of the Galaxy             333.13       76.0
Prometheus                          126.46       65.0
```

[3]:
```
data.tail(2)
```

[3]:
```
              Rank                  Genre  \
Title
Search Party   999       Adventure,Comedy
Nine Lives    1000  Comedy,Family,Fantasy

                                                  Description  \
Title
Search Party  A pair of friends embark on a mission to reuni...
Nine Lives    A stuffy businessman finds himself trapped ins...

                     Director  \
Title
Search Party    Scot Armstrong
Nine Lives    Barry Sonnenfeld

                                                      Actors  Year  \
Title
Search Party  Adam Pally, T.J. Miller, Thomas Middleditch,Sh...  2014
Nine Lives    Kevin Spacey, Jennifer Garner, Robbie Amell,Ch...  2016

              Runtime (Minutes)  Rating  Votes  Revenue (Millions)  Metascore
Title
Search Party                 93     5.6   4881                 NaN       22.0
Nine Lives                   87     5.3  12435               19.64       11.0
```

Info():

It provides the essential details about your dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory your DataFrame is using.

[4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Guardians of the Galaxy to Nine Lives
Data columns (total 11 columns):
Rank                   1000 non-null int64
Genre                  1000 non-null object
Description            1000 non-null object
Director               1000 non-null object
Actors                 1000 non-null object
Year                   1000 non-null int64
Runtime (Minutes)      1000 non-null int64
Rating                 1000 non-null float64
Votes                  1000 non-null int64
Revenue (Millions)     872 non-null float64
Metascore              936 non-null float64
dtypes: float64(3), int64(4), object(4)
memory usage: 93.8+ KB
```

[5]:
```
data.shape
```

[5]:
```
(1000, 11)
```

0.1.10 Handling duplicates

[6]:
```
temp_df = data.append(data)

temp_df.shape
```

[6]:
```
(2000, 11)
```

- The **drop_duplicates()** method will also return a copy of your DataFrame, but this time with duplicates removed.

[7]:
```
temp_df = temp_df.drop_duplicates(inplace=True)

temp_df.shape
```

```
␣
↪---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call␣
↪last)

<ipython-input-7-76715e17eef9> in <module>
  1 temp_df = temp_df.drop_duplicates(inplace=True)
  2
----> 3 temp_df.shape


AttributeError: 'NoneType' object has no attribute 'shape'
```

- It's a little verbose to keep assigning DataFrames to the same variable like in this example. For this reason, pandas has the inplace keyword argument on many of its methods.

[9]:
```
temp_df.drop_duplicates(inplace=True)
```

```
␣
↪---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call␣
↪last)
```

```
    <ipython-input-9-d185a6b6c56f> in <module>
----> 1 temp_df.drop_duplicates(inplace=True)


AttributeError: 'NoneType' object has no attribute 'drop_duplicates'
```

Another important argument for **drop_duplicates()** is **keep**, which has three possible options:

1. first: (default) Drop duplicates except for the first occurrence.
2. last: Drop duplicates except for the last occurrence.
3. False: Drop all duplicates.

[10]:
```
temp_df = data.append(data)  # make a new copy

temp_df.drop_duplicates(inplace=True, keep=False)

temp_df.shape
```

[10]:
```
(0, 11)
```

Column cleanup

[11]:
```
data.columns
```

[11]:
```
Index(['Rank', 'Genre', 'Description', 'Director', 'Actors', 'Year',
       'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)',
       'Metascore'],
      dtype='object')
```

[12]:
```
data.rename(columns={
        'Runtime (Minutes)': 'Runtime',
        'Revenue (Millions)': 'Revenue_millions'
    }, inplace=True)
data.columns
```

[12]:
```
Index(['Rank', 'Genre', 'Description', 'Director', 'Actors', 'Year', 'Runtime',
       'Rating', 'Votes', 'Revenue_millions', 'Metascore'],
      dtype='object')
```

But what if we want to lowercase all names? Instead of using .rename() we could also set a list of names to the columns like so:

[13]:
```
data.columns = ['rank', 'genre', 'description', 'director', 'actors', 'year',␣
 ↪'runtime',
                    'rating', 'votes', 'revenue_millions', 'metascore']


data.columns
```

[13]:
```
Index(['rank', 'genre', 'description', 'director', 'actors', 'year', 'runtime',
       'rating', 'votes', 'revenue_millions', 'metascore'],
```

```
      dtype='object')
```

Instead of just renaming each column manually we can do a list comprehension:

[14]:
```
data.columns=[col.lower() for col in data]
data.columns
```

[14]:
```
Index(['rank', 'genre', 'description', 'director', 'actors', 'year', 'runtime',
       'rating', 'votes', 'revenue_millions', 'metascore'],
      dtype='object')
```

0.1.11 Missing Data

When exploring data, you'll most likely encounter missing or null values, which are essentially placeholders for non-existent values. Most commonly you'll see Python's **None** or NumPy's **np.nan**, each of which are handled differently in some situations.

There are two options in dealing with nulls:

1. Get rid of rows or columns with nulls
2. Replace nulls with non-null values, a technique known as imputation

[15]:
```
data.isnull()
```

[15]:

Title	rank	genre	description	director	actors	year	runtime	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
...	...	...	...	...	...	...	...	...	...	...	...
Secret in Their Eyes	False	False	False	False	False	False	False	False	False	True	False
Hostel: Part II	False	False	False	False	False	False	False	False	False	False	False
Step Up 2: The Streets	False	False	False	False	False	False	False	False	False	False	False
Search Party	False	False	False	False	False	False	False	False	False	True	False
Nine Lives	False	False	False	False	False	False	False	False	False	False	False

```
[1000 rows x 11 columns]
```

Notice isnull() returns a DataFrame where each cell is either True or False depending on that cell's null status.

To count the number of nulls in each column we use an aggregate function for summing:

[16]:
```
data.isnull().sum()
```

[16]:
```
rank                   0
genre                  0
description            0
director               0
actors                 0
year                   0
runtime                0
rating                 0
votes                  0
revenue_millions     128
metascore             64
dtype: int64
```

Remove nulls is pretty simple:

[17]:
```
data.dropna()
```

[17]:
```
                           rank                      genre  \
Title
Guardians of the Galaxy       1   Action,Adventure,Sci-Fi
Prometheus                    2  Adventure,Mystery,Sci-Fi
Split                         3           Horror,Thriller
Sing                          4   Animation,Comedy,Family
Suicide Squad                 5  Action,Adventure,Fantasy
...                         ...                       ...
Resident Evil: Afterlife    994   Action,Adventure,Horror
Project X                   995                    Comedy
Hostel: Part II             997                    Horror
Step Up 2: The Streets      998       Drama,Music,Romance
Nine Lives                 1000     Comedy,Family,Fantasy


                                                                  description  \
Title
Guardians of the Galaxy   A group of intergalactic criminals are forced ...
Prometheus                Following clues to the origin of mankind, a te...
Split                     Three girls are kidnapped by a man with a diag...
Sing                      In a city of humanoid animals, a hustling thea...
Suicide Squad             A secret government agency recruits some of th...
...                                                                     ...
Resident Evil: Afterlife  While still out to destroy the evil Umbrella C...
Project X                 3 high school seniors throw a birthday party t...
```

```
Hostel: Part II           Three American college students studying abroa...
Step Up 2: The Streets    Romantic sparks occur between two dance studen...
Nine Lives                A stuffy businessman finds himself trapped ins...

                                    director  \
Title
Guardians of the Galaxy            James Gunn
Prometheus                       Ridley Scott
Split                      M. Night Shyamalan
Sing                      Christophe Lourdelet
Suicide Squad                     David Ayer
...                                      ...
Resident Evil: Afterlife     Paul W.S. Anderson
Project X                       Nima Nourizadeh
Hostel: Part II                    Eli Roth
Step Up 2: The Streets             Jon M. Chu
Nine Lives                   Barry Sonnenfeld

                                                                  actors  \
Title
Guardians of the Galaxy   Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...
Prometheus                Noomi Rapace, Logan Marshall-Green, Michael Fa...
Split                     James McAvoy, Anya Taylor-Joy, Haley Lu Richar...
Sing                      Matthew McConaughey,Reese Witherspoon, Seth Ma...
Suicide Squad             Will Smith, Jared Leto, Margot Robbie, Viola D...
...                                                                     ...
Resident Evil: Afterlife  Milla Jovovich, Ali Larter, Wentworth Miller,K...
Project X                 Thomas Mann, Oliver Cooper, Jonathan Daniel Br...
Hostel: Part II           Lauren German, Heather Matarazzo, Bijou Philli...
Step Up 2: The Streets    Robert Hoffman, Briana Evigan, Cassie Ventura,...
Nine Lives                Kevin Spacey, Jennifer Garner, Robbie Amell,Ch...

                          year  runtime  rating   votes  revenue_millions  \
Title
Guardians of the Galaxy   2014      121     8.1  757074            333.13
Prometheus                2012      124     7.0  485820            126.46
Split                     2016      117     7.3  157606            138.12
Sing                      2016      108     7.2   60545            270.32
Suicide Squad             2016      123     6.2  393727            325.02
...                        ...      ...     ...     ...               ...
Resident Evil: Afterlife  2010       97     5.9  140900             60.13
Project X                 2012       88     6.7  164088             54.72
Hostel: Part II           2007       94     5.5   73152             17.54
Step Up 2: The Streets    2008       98     6.2   70699             58.01
Nine Lives                2016       87     5.3   12435             19.64

                          metascore
```

```
Title
Guardians of the Galaxy        76.0
Prometheus                     65.0
Split                          62.0
Sing                           59.0
Suicide Squad                  40.0
...                             ...
Resident Evil: Afterlife       37.0
Project X                      48.0
Hostel: Part II                46.0
Step Up 2: The Streets         50.0
Nine Lives                     11.0

[838 rows x 11 columns]
```

Other than just dropping rows, you can also drop columns with null values by setting axis=1: data.dropna(axis=1) **Imputation**

- Imputation is a conventional feature engineering technique used to keep valuable data that have null values.
- There may be instances where dropping every row with a null value removes too big a chunk from your dataset, so instead we can impute that null with another value, usually the mean or the median of that column.

Let's look at imputing the missing values in the revenue_millions column. First we'll extract that column into its own variable:

[18]:
```
revenue=data['revenue_millions']
```

[19]:
```
revenue.head()
```

[19]:
```
Title
Guardians of the Galaxy    333.13
Prometheus                 126.46
Split                      138.12
Sing                       270.32
Suicide Squad              325.02
Name: revenue_millions, dtype: float64
```

Note: Using square brackets is the general way we select columns in a DataFrame.

[20]:
```
revenue_mean = revenue.mean()

revenue_mean
```

[20]:
```
82.95637614678898
```

With the mean, let's fill the nulls using **fillna():**

[21]:
```
revenue.fillna(revenue_mean, inplace=True)
```

We have now replaced all nulls in revenue with the mean of the column. Notice that by using inplace=True

[22]:
```
data.isnull().sum()
```

[22]:
```
rank                  0
genre                 0
description           0
director              0
actors                0
year                  0
runtime               0
rating                0
votes                 0
revenue_millions      0
metascore            64
dtype: int64
```

Using describe() on an entire DataFrame we can get a summary of the distribution of continuous variables:

[23]:
```
data.describe()
```

[23]:
```
              rank         year      runtime       rating         votes  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1.000000e+03
mean    500.500000  2012.783000   113.172000     6.723200  1.698083e+05
std     288.819436     3.205962    18.810908     0.945429  1.887626e+05
min       1.000000  2006.000000    66.000000     1.900000  6.100000e+01
25%     250.750000  2010.000000   100.000000     6.200000  3.630900e+04
50%     500.500000  2014.000000   111.000000     6.800000  1.107990e+05
75%     750.250000  2016.000000   123.000000     7.400000  2.399098e+05
max    1000.000000  2016.000000   191.000000     9.000000  1.791916e+06

       revenue_millions   metascore
count       1000.000000  936.000000
mean          82.956376   58.985043
std           96.412043   17.194757
min            0.000000   11.000000
25%           17.442500   47.000000
50%           60.375000   59.500000
75%           99.177500   72.000000
max          936.630000  100.000000
```

[24]:
```
data['genre'].describe()
```

[24]:
```
count                        1000
unique                        207
top       Action,Adventure,Sci-Fi
freq                           50
Name: genre, dtype: object
```

[25]:
```
data['genre'].value_counts().head(10)
```

[25]:
```
Action,Adventure,Sci-Fi       50
Drama                         48
Comedy,Drama,Romance          35
Comedy                        32
Drama,Romance                 31
Animation,Adventure,Comedy    27
Action,Adventure,Fantasy      27
Comedy,Drama                  27
Comedy,Romance                26
Crime,Drama,Thriller          24
Name: genre, dtype: int64
```

0.1.12 Relationships between continuous variables

By using the correlation method .corr() we can generate the relationship between each continuous variable:

[26]:
```
data.corr()
```

[26]:
```
                     rank      year   runtime    rating     votes  \
rank             1.000000 -0.261605 -0.221739 -0.219555 -0.283876
year            -0.261605  1.000000 -0.164900 -0.211219 -0.411904
runtime         -0.221739 -0.164900  1.000000  0.392214  0.407062
rating          -0.219555 -0.211219  0.392214  1.000000  0.511537
votes           -0.283876 -0.411904  0.407062  0.511537  1.000000
revenue_millions -0.252996 -0.117562  0.247834  0.189527  0.607941
metascore       -0.191869 -0.079305  0.211978  0.631897  0.325684

                 revenue_millions  metascore
rank                    -0.252996  -0.191869
year                    -0.117562  -0.079305
runtime                  0.247834   0.211978
rating                   0.189527   0.631897
votes                    0.607941   0.325684
revenue_millions         1.000000   0.133328
metascore                0.133328   1.000000
```

0.1.13 DataFrame slicing, selecting, extracting

[27]:
```
data.genre
```

[27]:
```
Title
Guardians of the Galaxy     Action,Adventure,Sci-Fi
Prometheus                 Adventure,Mystery,Sci-Fi
Split                                Horror,Thriller
Sing                        Animation,Comedy,Family
Suicide Squad              Action,Adventure,Fantasy
                                          ...
```

```
Secret in Their Eyes            Crime,Drama,Mystery
Hostel: Part II                                Horror
Step Up 2: The Streets          Drama,Music,Romance
Search Party                       Adventure,Comedy
Nine Lives                    Comedy,Family,Fantasy
Name: genre, Length: 1000, dtype: object
```

[28]:
```
genre_col=data['genre']
type(genre_col)
```

[28]:
```
pandas.core.series.Series
```

- This will return a Series. To extract a column as a DataFrame, you need to pass a list of column names.

adding another column name is easy:

[29]:
```
subset = data[['genre', 'rating']]

subset.head()
```

[29]:
```
                                        genre  rating
Title
Guardians of the Galaxy   Action,Adventure,Sci-Fi     8.1
Prometheus               Adventure,Mystery,Sci-Fi     7.0
Split                             Horror,Thriller     7.3
Sing                       Animation,Comedy,Family     7.2
Suicide Squad             Action,Adventure,Fantasy     6.2
```

Now we'll look at getting data by rows.
By rows For rows, we have two options:

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

Remember that we are still indexed by movie Title, so to use .loc we give it the Title of a movie:

[30]:
```
prom = data.loc["Prometheus"]

prom
```

[30]:
```
rank                                                        2
genre                                  Adventure,Mystery,Sci-Fi
description         Following clues to the origin of mankind, a te...
director                                              Ridley Scott
actors              Noomi Rapace, Logan Marshall-Green, Michael Fa...
year                                                     2012
runtime                                                   124
rating                                                      7
votes                                                  485820
revenue_millions                                       126.46
metascore                                                  65
```

```
Name: Prometheus, dtype: object
```

On the other hand, with iloc we give it the numerical index of Prometheus:

[31]:
```
prom = data.iloc[1]
prom
```

[31]:
```
rank                                                        2
genre                                    Adventure,Mystery,Sci-Fi
description         Following clues to the origin of mankind, a te...
director                                             Ridley Scott
actors              Noomi Rapace, Logan Marshall-Green, Michael Fa...
year                                                         2012
runtime                                                       124
rating                                                          7
votes                                                      485820
revenue_millions                                           126.46
metascore                                                      65
Name: Prometheus, dtype: object
```

[37]:
```
# movie_subset = data.loc[:,'Prometheus':'Sing']

movie_subset = data.iloc[1:4,1:2]

movie_subset
```

[37]:
```
                               genre
Title
Prometheus  Adventure,Mystery,Sci-Fi
Split                Horror,Thriller
Sing          Animation,Comedy,Family
```

- One important distinction between using **.loc** and **.iloc** to select multiple rows is that .locincludes the movie Sing in the result, but when using **.iloc** we're getting rows 1:4 but the movie at index 4 (Suicide Squad) is not included.
- Slicing with **.iloc** follows the same rules as slicing with lists, the object at the index at the end is not included.

Conditional selections Select movies_df where movies_df director equals Ridley Scott.

[54]:
```
condition = (data['director'] == "Ridley Scott")

condition.head()
```

[54]:
```
Title
Guardians of the Galaxy    False
Prometheus                  True
Split                      False
Sing                       False
```

```
Suicide Squad              False
Name: director, dtype: bool
```

conditional selections using numerical values by filtering the DataFrame by ratings:

[55]:
```
data[data['rating'] >= 8.6].head(3)
```

[55]:
```
                  rank                     genre  \
Title
Interstellar        37   Adventure,Drama,Sci-Fi
The Dark Knight     55       Action,Crime,Drama
Inception           81  Action,Adventure,Sci-Fi

                                                  description  \
Title
Interstellar     A team of explorers travel through a wormhole ...
The Dark Knight  When the menace known as the Joker wreaks havo...
Inception        A thief, who steals corporate secrets through ...

                        director  \
Title
Interstellar     Christopher Nolan
The Dark Knight  Christopher Nolan
Inception        Christopher Nolan

                                                    actors  year  \
Title
Interstellar     Matthew McConaughey, Anne Hathaway, Jessica Ch...  2014
The Dark Knight  Christian Bale, Heath Ledger, Aaron Eckhart,Mi...  2008
Inception        Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen...  2010

                  runtime  rating    votes  revenue_millions  metascore
Title
Interstellar          169     8.6  1047747            187.99       74.0
The Dark Knight       152     9.0  1791916            533.32       82.0
Inception             148     8.8  1583625            292.57       74.0
```

We can make some richer conditionals by using logical operators | for "or" and & for "and".

[56]:
```
data[(data['director'] == 'Christopher Nolan') | (data['director'] == 'Ridley␣
↪Scott')].head()
```

[56]:
```
                  rank                       genre  \
Title
Prometheus           2  Adventure,Mystery,Sci-Fi
Interstellar        37    Adventure,Drama,Sci-Fi
The Dark Knight     55        Action,Crime,Drama
The Prestige        65      Drama,Mystery,Sci-Fi
Inception           81   Action,Adventure,Sci-Fi
```

```
                                                      description  \
Title
Prometheus       Following clues to the origin of mankind, a te...
Interstellar     A team of explorers travel through a wormhole ...
The Dark Knight  When the menace known as the Joker wreaks havo...
The Prestige     Two stage magicians engage in competitive one-...
Inception        A thief, who steals corporate secrets through ...

                         director  \
Title
Prometheus            Ridley Scott
Interstellar     Christopher Nolan
The Dark Knight  Christopher Nolan
The Prestige     Christopher Nolan
Inception        Christopher Nolan

                                                         actors  year  \
Title
Prometheus       Noomi Rapace, Logan Marshall-Green, Michael Fa...  2012
Interstellar     Matthew McConaughey, Anne Hathaway, Jessica Ch...  2014
The Dark Knight  Christian Bale, Heath Ledger, Aaron Eckhart,Mi...  2008
The Prestige     Christian Bale, Hugh Jackman, Scarlett Johanss...  2006
Inception        Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen...  2010

                 runtime  rating    votes  revenue_millions  metascore
Title
Prometheus           124     7.0   485820            126.46       65.0
Interstellar         169     8.6  1047747            187.99       74.0
The Dark Knight      152     9.0  1791916            533.32       82.0
The Prestige         130     8.5   913152             53.08       66.0
Inception            148     8.8  1583625            292.57       74.0
```

We need to make sure to group evaluations with parentheses so Python knows how to evaluate the conditional. Using the isin() method we could make this more concise though:

[57]:
```
data[data['director'].isin(['Christopher Nolan', 'Ridley Scott'])].head()
```

[57]:
```
                 rank                      genre  \
Title
Prometheus          2  Adventure,Mystery,Sci-Fi
Interstellar       37    Adventure,Drama,Sci-Fi
The Dark Knight    55        Action,Crime,Drama
The Prestige       65      Drama,Mystery,Sci-Fi
Inception          81   Action,Adventure,Sci-Fi

                                                      description  \
Title
```

```
Prometheus       Following clues to the origin of mankind, a te...
Interstellar     A team of explorers travel through a wormhole ...
The Dark Knight  When the menace known as the Joker wreaks havo...
The Prestige     Two stage magicians engage in competitive one-...
Inception        A thief, who steals corporate secrets through ...

                         director  \
Title
Prometheus            Ridley Scott
Interstellar     Christopher Nolan
The Dark Knight  Christopher Nolan
The Prestige     Christopher Nolan
Inception        Christopher Nolan

                                                          actors  year  \
Title
Prometheus       Noomi Rapace, Logan Marshall-Green, Michael Fa...  2012
Interstellar     Matthew McConaughey, Anne Hathaway, Jessica Ch...  2014
The Dark Knight  Christian Bale, Heath Ledger, Aaron Eckhart,Mi...  2008
The Prestige     Christian Bale, Hugh Jackman, Scarlett Johanss...  2006
Inception        Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen...  2010

                 runtime  rating    votes  revenue_millions  metascore
Title
Prometheus           124     7.0   485820            126.46       65.0
Interstellar         169     8.6  1047747            187.99       74.0
The Dark Knight      152     9.0  1791916            533.32       82.0
The Prestige         130     8.5   913152             53.08       66.0
Inception            148     8.8  1583625            292.57       74.0
```

Let's say we want all movies that were released between 2005 and 2010, have a rating above 8.0, but made below the 25th percentile in revenue.

[58]:
```
data[
    ((data['year'] >= 2005) & (data['year'] <= 2010))
    & (data['rating'] > 8.0)
    & (data['revenue_millions'] < data['revenue_millions'].quantile(0.25))
]
```

[58]:
```
                    rank               genre  \
Title
3 Idiots             431        Comedy,Drama
The Lives of Others  477      Drama,Thriller
Incendies            714   Drama,Mystery,War
Taare Zameen Par     992  Drama,Family,Music

                                                        description  \
Title
```

```
3 Idiots             Two friends are searching for their long lost ...
The Lives of Others  In 1984 East Berlin, an agent of the secret po...
Incendies            Twins journey to the Middle East to discover t...
Taare Zameen Par     An eight-year-old boy is thought to be a lazy ...

                                          director  \
Title
3 Idiots                             Rajkumar Hirani
The Lives of Others  Florian Henckel von Donnersmarck
Incendies                           Denis Villeneuve
Taare Zameen Par                          Aamir Khan

                                                                actors  year  \
Title
3 Idiots                 Aamir Khan, Madhavan, Mona Singh, Sharman Joshi  2009
The Lives of Others  Ulrich Mühe, Martina Gedeck,Sebastian Koch, Ul...  2006
Incendies            Lubna Azabal, Mélissa Désormeaux-Poulin, Maxim...  2010
Taare Zameen Par     Darsheel Safary, Aamir Khan, Tanay Chheda, Sac...  2007

                     runtime  rating   votes  revenue_millions  metascore
Title
3 Idiots                 170     8.4  238789              6.52       67.0
The Lives of Others      137     8.5  278103             11.28       89.0
Incendies                131     8.2   92863              6.86       80.0
Taare Zameen Par         165     8.5  102697              1.20       42.0
```

0.1.14 Applying functions

- It is possible to iterate over a DataFrame or Series as you would with a list, but doing so — especially on large datasets — is very slow.
- An efficient alternative is to apply() a function to the dataset. For example, we could use a function to convert movies with an 8.0 or greater to a string value of "good" and the rest to "bad" and use this transformed values to create a new column.
- First we would create a function that, when given a rating, determines if it's good or bad:

[59]:
```
def rating_function(x):
    if x>=8.0:
        return "good"
    else:
        return "bad"
```

Now we want to send the entire rating column through this function, which is what apply() does:

[60]:
```
data["rating_category"] = data["rating"].apply(rating_function)

data.head(2)
```

[60]:
```
                          rank                     genre  \
Title
Guardians of the Galaxy      1    Action,Adventure,Sci-Fi
Prometheus                   2   Adventure,Mystery,Sci-Fi

                                                                 description  \
Title
Guardians of the Galaxy  A group of intergalactic criminals are forced ...
Prometheus               Following clues to the origin of mankind, a te...

                             director  \
Title
Guardians of the Galaxy    James Gunn
Prometheus                Ridley Scott

                                                                       actors  \
Title
Guardians of the Galaxy  Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...
Prometheus               Noomi Rapace, Logan Marshall-Green, Michael Fa...

                         year  runtime  rating   votes  revenue_millions  \
Title
Guardians of the Galaxy  2014      121     8.1  757074            333.13
Prometheus               2012      124     7.0  485820            126.46

                         metascore rating_category
Title
Guardians of the Galaxy       76.0            good
Prometheus                    65.0             bad
```

You can also use anonymous functions as well. This lambda function achieves the same result as rating_function

[62]:
```
data["rating_category"] = data["rating"].apply(lambda x: 'good' if x >= 8.0␣
 ↪else 'bad')

data.head(2)
```

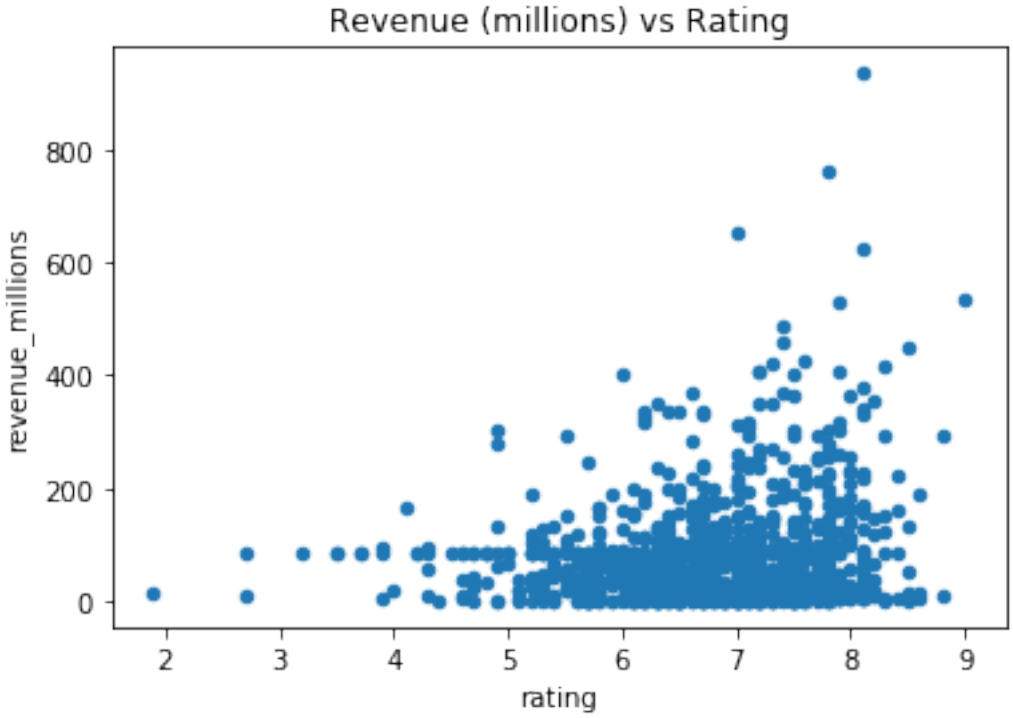
[62]:
```
                          rank                     genre  \
Title
Guardians of the Galaxy      1    Action,Adventure,Sci-Fi
Prometheus                   2   Adventure,Mystery,Sci-Fi

                                                                 description  \
Title
Guardians of the Galaxy  A group of intergalactic criminals are forced ...
Prometheus               Following clues to the origin of mankind, a te...

                             director  \
```

```
Title
Guardians of the Galaxy    James Gunn
Prometheus                Ridley Scott

                                                                  actors  \
Title
Guardians of the Galaxy  Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...
Prometheus               Noomi Rapace, Logan Marshall-Green, Michael Fa...

                         year  runtime  rating   votes  revenue_millions  \
Title
Guardians of the Galaxy  2014      121     8.1  757074            333.13
Prometheus               2012      124     7.0  485820            126.46

                         metascore rating_category
Title
Guardians of the Galaxy       76.0            good
Prometheus                    65.0             bad
```

0.1.15 Plotting

- Let's plot the relationship between ratings and revenue. All we need to do is call .plot() on data with some info about how to construct the plot:

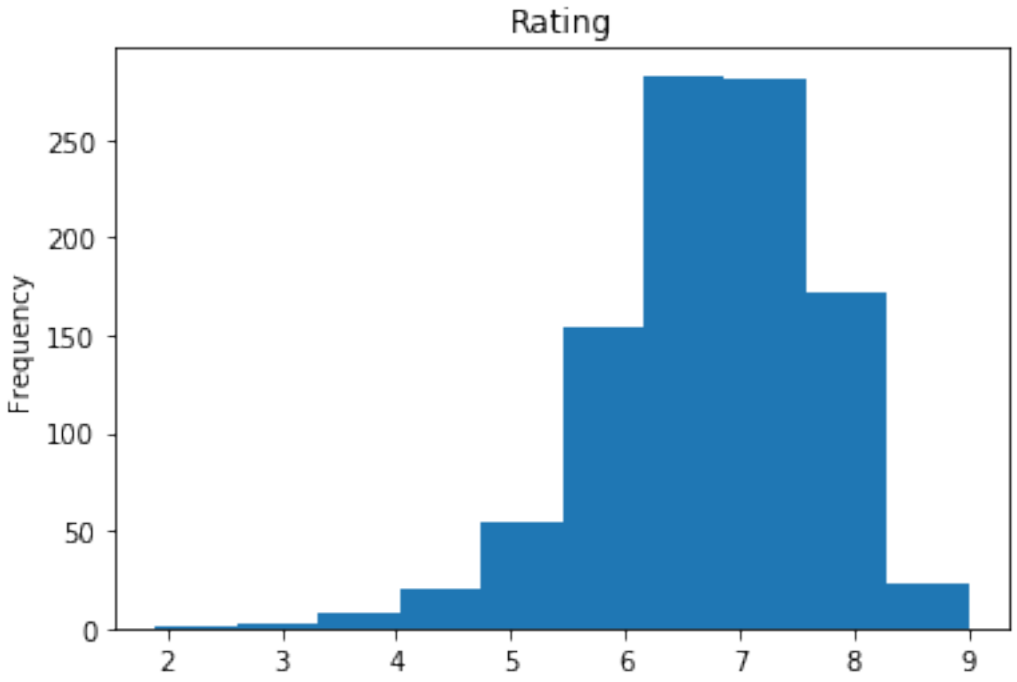
[64]:
```
data.plot(kind='scatter', x='rating', y='revenue_millions', title='Revenue␣
↪(millions) vs Rating');
```



If we want to plot a simple Histogram based on a single column, we can call plot on a column:

[67]:
```
data['rating'].plot(kind='hist', title='Rating')
```

[67]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x160b3b55908>
```



Using a Boxplot we can visualize this data

[68]:
```
data['rating'].plot(kind="box")
```

[68]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x160b54e99e8>
```

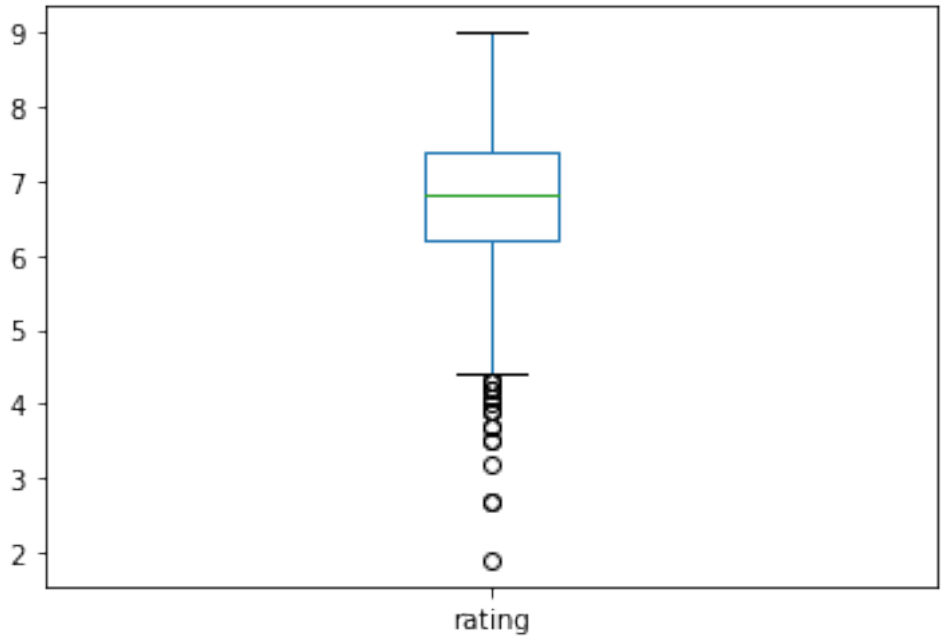
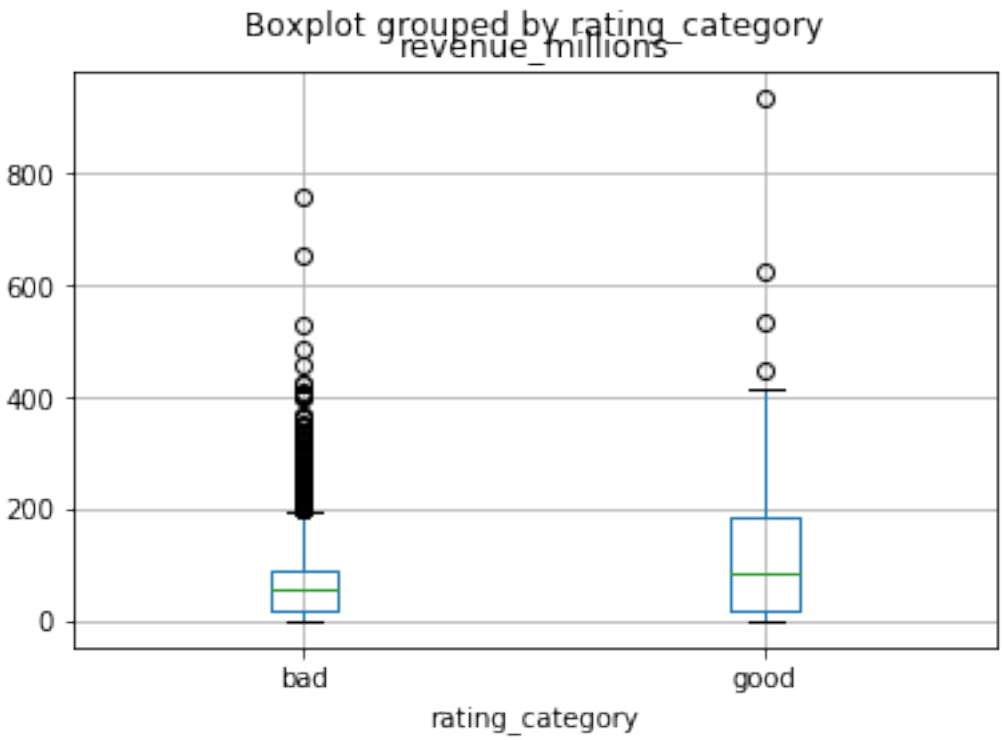


image.png

By combining categorical and continuous data, we can create a Boxplot of revenue that is grouped by the Rating Category we created above:

[69]:
```
data.boxplot(column='revenue_millions', by='rating_category')
```

[69]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x160b5576278>
```



0.2 How to efficiently loop through Pandas DataFrame

[1]:
```
df=pd.DataFrame(np.random.randint(0,10,size=(100,4)),columns=list('ABCD'))
df.head()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

[1]:

	A	B	C	D
0	7	7	4	4
1	7	0	4	6
2	8	0	8	3
3	8	3	2	1
4	1	4	6	8

1. for loop and iloc()

[2]:
```
def loop_with_for(df):
    temp=0
    for index in range(len(df)):
```

```
        temp += df['A'].iloc[index]+ df['B'].iloc[index]
    return temp
```

[3]:
```
%timeit loop_with_for(df)
```

```
4.12 ms ś 505 ţs per loop (mean ś std. dev. of 7 runs, 100 loops each)
```

2. iterrows()

[4]:
```
def loop_with_iterrows(df):
    temp=0
    for _,row in df.iterrows():
        temp += row.A + row.B
    return temp
```

[5]:
```
%timeit loop_with_iterrows(df)
```

```
19.2 ms ś 4.39 ms per loop (mean ś std. dev. of 7 runs, 10 loops each)
```

3. pandas itertuples function The pandas itertuples function is similar to iterrows, except it returns a namedtuple for each row, and preserves dtypes across rows.

[6]:
```
def loop_with_itertuples(df):
    temp=0
    for row_tuple in df.itertuples():
        temp +=row_tuple.A + row_tuple.B
    return temp
```

[7]:
```
%timeit loop_with_itertuples(df)
```

```
2.02 ms ś 313 ţs per loop (mean ś std. dev. of 7 runs, 100 loops each)
```

4. python zip

[8]:
```
def loop_with_zip(df):
    temp=0
    for a, b in zip(df['A'],df['B']):
        temp+=a+b
    return temp
```

[9]:
```
%timeit loop_with_zip(df)
```

```
87.8 ţs ś 5.1 ţs per loop (mean ś std. dev. of 7 runs, 10000 loops each)
```

5. Pandas apply function

[10]:
```
def using_apply(df):
    return df.apply(lambda x: x['A']+x['B'],axis=1).sum()
```

[11]:
```
%timeit using_apply(df)
```

```
5.67 ms ś 869 ţs per loop (mean ś std. dev. of 7 runs, 100 loops each)
```

[12]:
```
def using_pandas_builtin(df):
    return (df['A'] + df['B']).sum()
```

[13]:
```
%timeit using_pandas_builtin(df)
```

```
599 ţs ś 182 ţs per loop (mean ś std. dev. of 7 runs, 1000 loops each)
```

7. numpy builtin function

[14]:
```
def using_numpy_builtin(df):
    return (df['A'].values + df['B'].values).sum()
```

[15]:
```
%timeit using_numpy_builtin(df)
```

```
18.5 ţs ś 5.91 ţs per loop (mean ś std. dev. of 7 runs, 10000 loops each)
```

0.3 Combining DataFrames

[72]:
```
# Create dictionary1
data1 = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2],
    'Mango': [5,2,3,2]
}

# Convert to Data frame
df1 = pd.DataFrame(data1)
# Print the dataframe
df1

# Create dictionary2
data2 = {
    'apples': [5,2,3,2],
    'oranges': [5,2,3,2],
    'Mango': [5,2,3,2]
}

# Convert to Data frame
df2 = pd.DataFrame(data2)
# Print the dataframe
df2
```

[72]:

	apples	oranges	Mango
0	5	5	5
1	2	2	2
2	3	3	3
3	2	2	2

0.3.1 Concatenating DataFrames

[73]:
```
# Stack the DataFrames on top of each other
vertical_stack = pd.concat([df1,df2],axis=0)
vertical_stack
```

[73]:

	apples	oranges	Mango
0	3	0	5
1	2	3	2
2	0	7	3
3	1	2	2
0	5	5	5
1	2	2	2
2	3	3	3
3	2	2	2

[76]:
```
# We can reindex the new dataframe using the reset_index() method
vertical_stack=vertical_stack.reset_index(drop=True)
vertical_stack
```

[76]:

	apples	oranges	Mango
0	3	0	5
1	2	3	2
2	0	7	3
3	1	2	2
4	5	5	5
5	2	2	2
6	3	3	3
7	2	2	2

[74]:
```
# Place the DataFrames side by side
horizontal_stack = pd.concat([df1,df2],axis=1)
horizontal_stack
```

[74]:

	apples	oranges	Mango	apples	oranges	Mango
0	3	0	5	5	5	5
1	2	3	2	2	2	2
2	0	7	3	3	3	3
3	1	2	2	2	2	2

0.3.2 Joining DataFrames

- When we concatenated our DataFrames we simply added them to each other - stacking them either vertically or side by side.
- Another way to combine DataFrames is to use columns in each dataset that contain common values (a common unique id).
- Combining DataFrames using a common field is called "joining". The columns containing the common values are called "join key(s)".

[81]:
```
# Create dictionary3
data3 = {
    'apples': [3, 5, 0, 1],
    'grapes': [0, 3, 7, 2],
    'pomegranate': [5,2,3,2]
}

# Convert to Data frame
df3 = pd.DataFrame(data3)
# Print the dataframe
df3
```

[81]:

	apples	grapes	pomegranate
0	3	0	5
1	5	3	2
2	0	7	3
3	1	2	2

0.3.3 1. Inner Joins

- The most common type of join is called an inner join. An inner join combines two DataFrames based on a join key and returns a new DataFrame that contains only those rows that have matching values in both of the original DataFrames.

image.png

[79]:
```
merged_inner=pd.merge(left=df1,right=df2,left_on='apples',right_on='apples')
merged_inner
```

[79]:

	apples	oranges_x	Mango_x	oranges_y	Mango_y
0	3	0	5	3	3
1	2	3	2	2	2
2	2	3	2	2	2

[82]:
```
merged_inner=pd.merge(df1,df3,how='inner',on='apples')
merged_inner
```

[82]:

	apples	oranges	Mango	grapes	pomegranate
0	3	0	5	0	5
1	0	7	3	7	3
2	1	2	2	2	2

0.3.4 2. Left joins

- A left join will return all of the rows from the left DataFrame, even those rows whose join key(s) do not have values in the right DataFrame.

- Rows in the left DataFrame that are missing values for the join key(s) in the right DataFrame will simply have null (i.e., NaN or None) values for those columns in the resulting joined DataFrame.

Note: a left join will still discard rows from the right DataFrame that do not have values for the join key(s) in the left DataFrame.

image.png

[83]:
```
merged_left=pd.merge(df1,df3,how='left',on='apples')
merged_left
```

[83]:

	apples	oranges	Mango	grapes	pomegranate
0	3	0	5	0.0	5.0
1	2	3	2	NaN	NaN
2	0	7	3	7.0	3.0
3	1	2	2	2.0	2.0

0.3.5 3. Right joins

[84]:
```
merged_right=pd.merge(df1,df3,how='right',on='apples')
merged_right
```

[84]:

	apples	oranges	Mango	grapes	pomegranate
0	3	0.0	5.0	0	5
1	0	7.0	3.0	7	3
2	1	2.0	2.0	2	2
3	5	NaN	NaN	3	2

0.3.6 4. Outer joins

[86]:
```
merged_Outer=pd.merge(df1,df3,how='outer',on='apples')
merged_Outer
```

[86]:

	apples	oranges	Mango	grapes	pomegranate
0	3	0.0	5.0	0.0	5.0
1	2	3.0	2.0	NaN	NaN
2	0	7.0	3.0	7.0	3.0
3	1	2.0	2.0	2.0	2.0
4	5	NaN	NaN	3.0	2.0

0.3.7 Time-series friendly merging

- Pandas provides special functions for merging Time-series DataFrames. Perhaps the most useful and popular one is the **merge_asof()** function.
- The **merge_asof()** is similar to an ordered left-join except that you match on nearest key rather than equal keys.

- For each row in the left DataFrame, you select the last row in the right DataFrame whose on key is less than the left's key. Both DataFrames must be sorted by the key.
- Optionally an asof merge can perform a group-wise merge. This matches the by key equally, in addition to the nearest match on the on key.

For example, you might have trades and quotes, and you want to asof merge them. Here the left DataFrame is chosen as trades and right DataFrame as quotes. They are asof merged on key time and group-wise merged by their ticker symbol.

[87]:
```
trades = pd.DataFrame({
    'time': pd.to_datetime(['20160525 13:30:00.023',
                            '20160525 13:30:00.038',
                            '20160525 13:30:00.048',
                            '20160525 13:30:00.048',
                            '20160525 13:30:00.048']),
    'ticker': ['MSFT', 'MSFT','GOOG', 'GOOG', 'AAPL'],
    'price': [51.95, 51.95,720.77, 720.92, 98.00],
    'quantity': [75, 155,100, 100, 100]},
    columns=['time', 'ticker', 'price', 'quantity'])

quotes = pd.DataFrame({
    'time': pd.to_datetime(['20160525 13:30:00.023',
                            '20160525 13:30:00.023',
                            '20160525 13:30:00.030',
                            '20160525 13:30:00.041',
                            '20160525 13:30:00.048',
                            '20160525 13:30:00.049',
                            '20160525 13:30:00.072',
                            '20160525 13:30:00.075']),
    'ticker': ['GOOG', 'MSFT', 'MSFT','MSFT', 'GOOG', 'AAPL', 'GOOG','MSFT'],
    'bid': [720.50, 51.95, 51.97, 51.99,720.50, 97.99, 720.50, 52.01],
    'ask': [720.93, 51.96, 51.98, 52.00,720.93, 98.01, 720.88, 52.03]},
    columns=['time', 'ticker', 'bid', 'ask'])
```

[88]:
```
trades
```

[88]:
```
                     time ticker   price  quantity
0 2016-05-25 13:30:00.023   MSFT   51.95        75
1 2016-05-25 13:30:00.038   MSFT   51.95       155
2 2016-05-25 13:30:00.048   GOOG  720.77       100
3 2016-05-25 13:30:00.048   GOOG  720.92       100
4 2016-05-25 13:30:00.048   AAPL   98.00       100
```

[89]:
```
quotes
```

[89]:
```
                     time ticker     bid     ask
0 2016-05-25 13:30:00.023   GOOG  720.50  720.93
1 2016-05-25 13:30:00.023   MSFT   51.95   51.96
2 2016-05-25 13:30:00.030   MSFT   51.97   51.98
3 2016-05-25 13:30:00.041   MSFT   51.99   52.00
```

```
4 2016-05-25 13:30:00.048   GOOG  720.50  720.93
5 2016-05-25 13:30:00.049   AAPL   97.99   98.01
6 2016-05-25 13:30:00.072   GOOG  720.50  720.88
7 2016-05-25 13:30:00.075   MSFT   52.01   52.03
```

[90]:
```
df_merge_asof = pd.merge_asof(trades, quotes,
                              on='time',
                              by='ticker')

df_merge_asof
```

[90]:
```
                     time ticker   price  quantity     bid     ask
0 2016-05-25 13:30:00.023   MSFT   51.95        75   51.95   51.96
1 2016-05-25 13:30:00.038   MSFT   51.95       155   51.97   51.98
2 2016-05-25 13:30:00.048   GOOG  720.77       100  720.50  720.93
3 2016-05-25 13:30:00.048   GOOG  720.92       100  720.50  720.93
4 2016-05-25 13:30:00.048   AAPL   98.00       100     NaN     NaN
```

- If you observe carefully, you can notice the reason behind NaN appearing in the AAPL ticker row. Since the right DataFrame quotes didn't have any time value less than 13:30:00.048 (the time in the left table) for AAPL ticker, NaNs were introduced in the bid and ask columns.
- You can also set a predefined tolerance level for time column. Suppose you only want asof merge within 2ms between the quote time and the trade time, then you will have to specify tolerance argument:

[91]:
```
df_merge_asof_tolerance = pd.merge_asof(trades, quotes,
                              on='time',
                              by='ticker',
                              tolerance=pd.Timedelta('2ms'))

df_merge_asof_tolerance
```

[91]:
```
                     time ticker   price  quantity     bid     ask
0 2016-05-25 13:30:00.023   MSFT   51.95        75   51.95   51.96
1 2016-05-25 13:30:00.038   MSFT   51.95       155     NaN     NaN
2 2016-05-25 13:30:00.048   GOOG  720.77       100  720.50  720.93
3 2016-05-25 13:30:00.048   GOOG  720.92       100  720.50  720.93
4 2016-05-25 13:30:00.048   AAPL   98.00       100     NaN     NaN
```

Notice the difference between the above and previous result. Rows are not merged if the time tolerance didn't match 2ms.

0.4 GroupBy

Any groupby operation involves one of the following operations on the original object. They are

```
- Splitting the Object

- Applying a function
```

- Combining the results

we split the data into sets and we apply some functionality on each subset. In the apply functionality, we can perform the following operations

- Aggregation computing a summary statistic

- Transformation perform some group-specific operation

- Filtration discarding the data with some condition

[93]:
```
# Define data frame
ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)
df
```

[93]:

	Team	Rank	Year	Points
0	Riders	1	2014	876
1	Riders	2	2015	789
2	Devils	2	2014	863
3	Devils	3	2015	673
4	Kings	3	2014	741
5	kings	4	2015	812
6	Kings	1	2016	756
7	Kings	1	2017	788
8	Riders	2	2016	694
9	Royals	4	2014	701
10	Royals	1	2015	804
11	Riders	2	2017	690

0.4.1 Split Data into Groups

There are multiple ways to split an object like - obj.groupby('key') - obj.groupby(['key1','key2']) - obj.groupby(key,axis=1)

[95]:
```
# The grouping objects can be applied to the DataFrame object
df.groupby('Team').groups
```

[95]:
```
{'Devils': Int64Index([2, 3], dtype='int64'),
 'Kings': Int64Index([4, 6, 7], dtype='int64'),
 'Riders': Int64Index([0, 1, 8, 11], dtype='int64'),
 'Royals': Int64Index([9, 10], dtype='int64'),
 'kings': Int64Index([5], dtype='int64')}
```

[96]:
```
df.groupby(['Team','Year']).groups
```

[96]:
```
{('Devils', 2014): Int64Index([2], dtype='int64'),
 ('Devils', 2015): Int64Index([3], dtype='int64'),
 ('Kings', 2014): Int64Index([4], dtype='int64'),
 ('Kings', 2016): Int64Index([6], dtype='int64'),
 ('Kings', 2017): Int64Index([7], dtype='int64'),
 ('Riders', 2014): Int64Index([0], dtype='int64'),
 ('Riders', 2015): Int64Index([1], dtype='int64'),
 ('Riders', 2016): Int64Index([8], dtype='int64'),
 ('Riders', 2017): Int64Index([11], dtype='int64'),
 ('Royals', 2014): Int64Index([9], dtype='int64'),
 ('Royals', 2015): Int64Index([10], dtype='int64'),
 ('kings', 2015): Int64Index([5], dtype='int64')}
```

0.4.2 Iterating through Groups

[97]:
```
groupby = df.groupby('Year')
for name,group in groupby:
    print(name)
    print(group)
```

```
2014
     Team  Rank  Year  Points
0  Riders     1  2014     876
2  Devils     2  2014     863
4   Kings     3  2014     741
9  Royals     4  2014     701
2015
      Team  Rank  Year  Points
1   Riders     2  2015     789
3   Devils     3  2015     673
5    kings     4  2015     812
10  Royals     1  2015     804
2016
     Team  Rank  Year  Points
6   Kings     1  2016     756
8  Riders     2  2016     694
2017
      Team  Rank  Year  Points
7    Kings     1  2017     788
11  Riders     2  2017     690
```

0.4.3 Select a Group

Using the get_group() method, we can select a single group.

[98]:
```
grouped = df.groupby('Year')
print(grouped.get_group(2014))
```

```
     Team  Rank  Year  Points
0  Riders     1  2014     876
2  Devils     2  2014     863
4   Kings     3  2014     741
9  Royals     4  2014     701
```

0.4.4 Aggregations

- An aggregated function returns a single aggregated value for each group. Once the group by object is created, several aggregation operations can be performed on the grouped data.

[99]:
```
grouped = df.groupby('Year')
print(grouped['Points'].agg(np.mean))
```

```
<IPython.core.display.Javascript object>


Year
2014    795.25
2015    769.50
2016    725.00
2017    739.00
Name: Points, dtype: float64
```

Size of each group is by applying the **size()** function

[100]:
```
grouped = df.groupby('Team')
print (grouped.agg(np.size))
```

```
<IPython.core.display.Javascript object>


        Rank  Year  Points
Team
Devils     2     2       2
Kings      3     3       3
Riders     4     4       4
Royals     2     2       2
kings      1     1       1
```

0.4.5 Applying Multiple Aggregation Functions at Once

[101]:
```
grouped = df.groupby('Team')
print(grouped['Points'].agg([np.sum, np.mean, np.std]))
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
         sum        mean         std
Team
Devils  1536  768.000000  134.350288
Kings   2285  761.666667   24.006943
Riders  3049  762.250000   88.567771
Royals  1505  752.500000   72.831998
kings    812  812.000000         NaN
```

0.4.6 Transformations

- Transformation on a group or a column returns an object that is indexed the same size of that is being grouped.
- Thus, the transform should return a result that is the same size as that of a group chunk.

[102]:
```
grouped = df.groupby('Team')
score = lambda x: (x - x.mean()) / x.std()*10
print(grouped.transform(score))
```

```
         Rank       Year     Points
0  -15.000000 -11.618950  12.843272
1    5.000000  -3.872983   3.020286
2   -7.071068  -7.071068   7.071068
3    7.071068   7.071068  -7.071068
4   11.547005 -10.910895  -8.608621
5         NaN        NaN        NaN
6   -5.773503   2.182179  -2.360428
7   -5.773503   8.728716  10.969049
8    5.000000   3.872983  -7.705963
9    7.071068  -7.071068  -7.071068
10  -7.071068   7.071068   7.071068
11   5.000000  11.618950  -8.157595
```

0.4.7 Filtration

- Filtration filters the data on a defined criteria and returns the subset of data. The filter() function is used to filter the data.

[103]:
```
print(df.groupby('Team').filter(lambda x: len(x) >= 3))
```

```
     Team  Rank  Year  Points
0  Riders     1  2014     876
1  Riders     2  2015     789
```

```
4    Kings     3  2014     741
6    Kings     1  2016     756
7    Kings     1  2017     788
8   Riders     2  2016     694
11  Riders     2  2017     690
```

0.4.8 Date Functionality

Create a Range of Dates

- Using the date.range() function by specifying the periods and the frequency, we can create the date series. By default, the frequency of range is Days.

[104]:
```
print(pd.date_range('1/1/2011', periods=5))
```

```
DatetimeIndex(['2011-01-01', '2011-01-02', '2011-01-03', '2011-01-04',
               '2011-01-05'],
              dtype='datetime64[ns]', freq='D')
```

Change the Date Frequency

[105]:
```
print(pd.date_range('1/1/2011', periods=5,freq='M'))
```

```
DatetimeIndex(['2011-01-31', '2011-02-28', '2011-03-31', '2011-04-30',
               '2011-05-31'],
              dtype='datetime64[ns]', freq='M')
```

bdate_range

- stands for business date ranges. Unlike date_range(), it excludes Saturday and Sunday.

[106]:
```
print(pd.date_range('1/1/2011', periods=5))
```

```
DatetimeIndex(['2011-01-01', '2011-01-02', '2011-01-03', '2011-01-04',
               '2011-01-05'],
              dtype='datetime64[ns]', freq='D')
```

[107]:
```
start = pd.datetime(2011, 1, 1)
end = pd.datetime(2011, 1, 5)

print(pd.date_range(start, end))
```

```
DatetimeIndex(['2011-01-01', '2011-01-02', '2011-01-03', '2011-01-04',
               '2011-01-05'],
              dtype='datetime64[ns]', freq='D')
```

0.4.9 Timedelta

- Timedeltas are differences in times, expressed in difference units, for example, days, hours, minutes, seconds.

String
By passing a string literal, we can create a timedelta object.

[108]:
```
print(pd.Timedelta('2 days 2 hours 15 minutes 30 seconds'))
```

```
2 days 02:15:30
```

Integer By passing an integer value with the unit,

[109]:
```
print(pd.Timedelta(6,unit='h'))
```

```
0 days 06:00:00
```

Data Offsets Data offsets such as - weeks, days, hours, minutes, seconds, milliseconds, microseconds, nanoseconds can also be used in construction.

[110]:
```
print(pd.Timedelta(days=2))
```

```
2 days 00:00:00
```

0.4.10 Categorical Data

- Categorical variables can take on only a limited, and usually fixed number of possible values.
- Besides the fixed length, categorical data might have an order but cannot perform numerical operation. Categorical are a Pandas data type.

Object Creation

[111]:
```
# By specifying the dtype as "category" in pandas object creation.
s = pd.Series(["a","b","c","a"], dtype="category")
print(s)
```

```
0    a
1    b
2    c
3    a
dtype: category
Categories (3, object): [a, b, c]
```

[112]:
```
cat = pd.Categorical(['a', 'b', 'c', 'a', 'b', 'c'])
print(cat)
```

```
[a, b, c, a, b, c]
Categories (3, object): [a, b, c]
```

Here, the second argument signifies the categories. Thus, any value which is not present in the categories will be treated as NaN

[113]:
```
cat = cat=pd.Categorical(['a','b','c','a','b','c','d'], ['c', 'b', 'a'])
print (cat)
```

```
[a, b, c, a, b, c, NaN]
Categories (3, object): [c, b, a]
```

[114]:
```
cat = cat=pd.Categorical(['a','b','c','a','b','c','d'], ['c', 'b',␣
↪'a'],ordered=True)
print (cat)
```

```
[a, b, c, a, b, c, NaN]
Categories (3, object): [c < b < a]
```

0.5 Example

[118]:
```
names = ['Bob','Jessica','Mary','John','Mel']
births = [968, 155, 77, 578, 973]
```

[119]:
```
dataset=list(zip(names,births))
dataset
```

[119]:
```
[('Bob', 968), ('Jessica', 155), ('Mary', 77), ('John', 578), ('Mel', 973)]
```

[120]:
```
df=pd.DataFrame(data=dataset,columns=["names","births"])
df
```

[120]:

	names	births
0	Bob	968
1	Jessica	155
2	Mary	77
3	John	578
4	Mel	973

[ ]:
```
df.to_csv('births.csv',index=True,header=True)
```

[121]:
```
df.sort_values(['births'],ascending=True).reset_index(drop=True)
```

[121]:

	names	births
0	Mary	77
1	Jessica	155
2	John	578
3	Bob	968
4	Mel	973

[122]:
```
df['births'].max()
```

[122]:
```
973
```

[123]:
```
df[df['births']==df['births'].max()]
```

[123]:
```
  names  births
4   Mel     973
```

[124]:
```
for x in df['names'].unique():
    print(x)
```

```
Bob
Jessica
Mary
John
Mel
```

[125]:
```
df['names'].describe()
```

[125]:
```
count       5
unique      5
top       Mel
freq        1
Name: names, dtype: object
```

[126]:
```
df.index
```

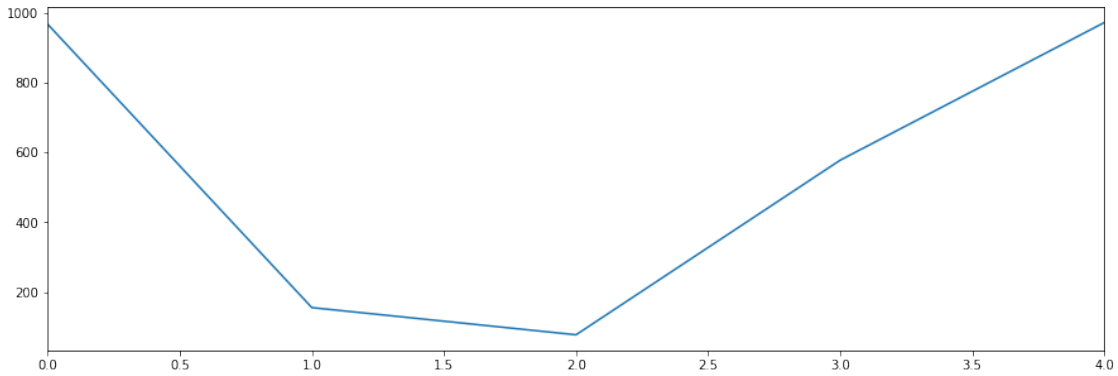
[126]:
```
RangeIndex(start=0, stop=5, step=1)
```

[127]:
```
df['names']=df.names.apply(lambda x: x.upper())
df
```

[127]:
```
     names  births
0      BOB     968
1  JESSICA     155
2     MARY      77
3     JOHN     578
4      MEL     973
```

[128]:
```
df['births'].plot(figsize=(15,5))
```

[128]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x160b595c128>
```



0.5.1 Pandas Styling

[1]:
```
import pandas as pd
import numpy as np

np.random.seed(24)
df = pd.DataFrame({'A': np.linspace(1, 10, 10)})
df = pd.concat([df, pd.DataFrame(np.random.randn(10, 4), columns=list('BCDE'))],
               axis=1)
df.iloc[0, 2] = np.nan
```

[2]:
```
df.style
```

[2]: <pandas.io.formats.style.Styler at 0x1b9fd985f28>

[3]:
```
def color_negative_red(val):
    """
    Takes a scalar and returns a string with
    the css property `'color: red'` for negative
    strings, black otherwise.
    """
    color = 'red' if val < 0 else 'black'
    return 'color: %s' % color
```

[4]:
```
s = df.style.applymap(color_negative_red)
s
```

[4]: <pandas.io.formats.style.Styler at 0x1b98681beb8>

[5]:
```
def highlight_max(s):
    '''
    highlight the maximum in a Series yellow.
    '''
    is_max = s == s.max()
    return ['background-color: yellow' if v else '' for v in is_max]
```

[6]:
```
df.style.apply(highlight_max)
```

[6]: <pandas.io.formats.style.Styler at 0x1b986876eb8>

[7]:
```
df.style.\
    applymap(color_negative_red).\
    apply(highlight_max)
```

[7]: <pandas.io.formats.style.Styler at 0x1b986876dd8>

[8]:
```
import seaborn as sns
cm=sns.light_palette('pink',as_cmap=True)
s=df.style.background_gradient(cmap=cm)
s
```

```
c:\users\gllb5989\appdata\local\programs\python\python36\lib\site-
packages\matplotlib\colors.py:504: RuntimeWarning: invalid value encountered in
```

```
less
  xa[xa < 0] = -1
```

[8]:
```
<pandas.io.formats.style.Styler at 0x1b98688b898>
```

[ ]:
```

```