Pandas

January 9, 2021

1 Pandas

Pandas is a Python library for data manipulation and analysis. Pandas offers powerful and flexible data structures that make it easy to manipulate and analyse data. It is used to clean, transform, and analyse data.

Pandas is built on top of NumPy.

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

1.1 Components of Pandas

The two main components of pandas are DataFrame and Series.

- A DataFrame represents a two-dimensional table. It consists of a collection of Series.
- A Series represents one-dimensional column (actually, a 1D NumPy array).

Each row in the table has an associated Index, and the columns are connedted via the Index. The columns are identified by the column names.

1.2 Series

Series can be created with the constructor `pd.Series(data, index, dtype)`. * data can be any object e.g. np.array(), list, dict, etc. * index values must have the same length as data (defaults to np.arange(data)) * dtype represents the data type

[2]:
```
pd.Series(np.array(["a", "b", "c", "d"]))
```

[2]:
```
0    a
1    b
2    c
3    d
dtype: object
```

[3]:
```
pd.Series(np.array(["a", "b", "c", "d"]), index=[10, 11, 12, 13])
```

[3]:
```
10    a
11    b
12    c
```

```
13    d
dtype: object
```

[4]:
```
data = {
    "a" : 0.,
    "b" : 1.,
    "c" : 2.
}

pd.Series(data)
```

[4]:
```
a    0.0
b    1.0
c    2.0
dtype: float64
```

[5]:
```
pd.Series(data, index=["b", "c", "d", "a"])
```

[5]:
```
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64
```

1.2.1 Accessing elements in a Series

Elements can be accessed by their row index, just like in a NumPy array. Slicing also works.

[6]:
```
s = pd.Series(np.array(["a", "b", "c", "d", "e"]))
s
```

[6]:
```
0    a
1    b
2    c
3    d
4    e
dtype: object
```

[7]:
```
s[0]
```

[7]:
```
'a'
```

[8]:
```
s[:3]
```

[8]:
```
0    a
1    b
2    c
```

```
dtype: object
```

They can also be accessed by their index labels.

[9]:
```
s = pd.Series([1, 2, 3, 4, 5], index=["a", "b", "c", "d", "e"])
s
```

[9]:
```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

[10]:
```
s["a"]
```

[10]:
```
1
```

[11]:
```
s[["a", "d", "c"]]
```

[11]:
```
a    1
d    4
c    3
dtype: int64
```

1.3 Creating DataFrames

1.3.1 Method 1: From a list or numpy array

[12]:
```
# Creating using Python lists
arr = [["UK", "London"],
       ["France", "Paris"],
       ["Italy", "Rome"]]
pd.DataFrame(arr, columns=["country", "capital"])
```

[12]:

	country	capital
0	UK	London
1	France	Paris
2	Italy	Rome

[13]:
```
# Creating using numpy arrays
arr = np.array([["UK", "London"],
                ["France", "Paris"],
                ["Italy", "Rome"]])
column_names = np.array(["country", "capital"])
pd.DataFrame(arr, columns=column_names)
```

[13]:
```
  country capital
0      UK  London
1  France   Paris
2   Italy    Rome
```

[14]:
```
# You can provide a custom index (axis labels)
prefixes = ["+44", "+33", "+39"]
pd.DataFrame(arr, columns=column_names, index=prefixes)
```

[14]:
```
    country capital
+44      UK  London
+33  France   Paris
+39   Italy    Rome
```

1.3.2 Method 2: From a dictionary

[15]:
```
data_dict = {
    "country": ["UK", "France", "Italy"],
    "capital": ["London", "Paris", "Rome"]
}
pd.DataFrame(data_dict)
```

[15]:
```
  country capital
0      UK  London
1  France   Paris
2   Italy    Rome
```

1.3.3 Method 3: From a dictionary of series

[16]:
```
country_series = pd.Series(np.array(["UK", "France", "Italy"]))
capital_series = pd.Series(np.array(["London", "Paris", "Rome"]))
data_dict = {
    "country": country_series,
    "capital": capital_series
}
pd.DataFrame(data_dict)
```

[16]:
```
  country capital
0      UK  London
1  France   Paris
2   Italy    Rome
```

If you provide a custom index, the output index for the DataFrame will be a union of the indicies.

[17]:
```
data = {
    "one": pd.Series([1, 2, 3, 4], index=["a", "b", "c", "d"]),
    "two": pd.Series([5, 6, 7, 8, 9], index=["a", "b", "c", "e", "f"])
```

```
}
pd.DataFrame(data)
```

[17]:
```
   one  two
a  1.0  5.0
b  2.0  6.0
c  3.0  7.0
d  4.0  NaN
e  NaN  8.0
f  NaN  9.0
```

1.3.4 Method 4: From a CSV File

[18]:
```
pd.read_csv("data/pandas-sample.csv")
# Notice how '+' in code was interpreted as a plus sign
```

[18]:
```
   code country capital
0    44      UK  London
1    33  France   Paris
2    39   Italy    Rome
```

[19]:
```
pd.read_csv("data/pandas-sample.csv", dtype=str)
# Now the + is correctly parsed
```

[19]:
```
  code country capital
0  +44      UK  London
1  +33  France   Paris
2  +39   Italy    Rome
```

[20]:
```
pd.read_csv("data/pandas-sample.csv", index_col=0)
# To use one of the columns as an index
```

[20]:
```
     country capital
code
44        UK  London
33    France   Paris
39     Italy    Rome
```

1.3.5 Method 5: From a JSON File

[21]:
```
df = pd.read_json("data/pandas-sample.json")
df.set_index('code', inplace=True)
df
```

[21]:
```
     country capital
code
44        UK  London
```

```
33    France   Paris
39     Italy    Rome
```

1.4 DataFrame Operations

[22]:
```
# Setup: Importing IMDB CSV File
df = pd.read_csv("data/IMDB-Movie-Data.csv", index_col="Rank")
```

`df.head()` returns the first few entries in the dataframe. By default it returns 5 rows.

[23]:
```
# iloc[:, :2] (described below) returns the first two columns (for displaying)
df.head().iloc[:, :2]
```

[23]:

Rank	Title	Genre
1	Guardians of the Galaxy	Action,Adventure,Sci-Fi
2	Prometheus	Adventure,Mystery,Sci-Fi
3	Split	Horror,Thriller
4	Sing	Animation,Comedy,Family
5	Suicide Squad	Action,Adventure,Fantasy

You can specify the limit in the function's argument.

[24]:
```
df.head(3).iloc[:, :2]
```

[24]:

Rank	Title	Genre
1	Guardians of the Galaxy	Action,Adventure,Sci-Fi
2	Prometheus	Adventure,Mystery,Sci-Fi
3	Split	Horror,Thriller

`df.tail()`, like `df.head()`, returns the last few entries.

[25]:
```
df.tail(3).iloc[:, :2]
```

[25]:

Rank	Title	Genre
998	Step Up 2: The Streets	Drama,Music,Romance
999	Search Party	Adventure,Comedy
1000	Nine Lives	Comedy,Family,Fantasy

[26]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 1 to 1000
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
```

```
 0   Title                1000 non-null   object
 1   Genre                1000 non-null   object
 2   Description          1000 non-null   object
 3   Director             1000 non-null   object
 4   Actors               1000 non-null   object
 5   Year                 1000 non-null   int64
 6   Runtime (Minutes)    1000 non-null   int64
 7   Rating               1000 non-null   float64
 8   Votes                1000 non-null   int64
 9   Revenue (Millions)   872 non-null    float64
 10  Metascore            936 non-null    float64
dtypes: float64(3), int64(3), object(5)
memory usage: 93.8+ KB
```

[27]:
```
df.shape
```

[27]:
```
(1000, 11)
```

[28]:
```
df.size
```

[28]:
```
11000
```

[29]:
```
type(df.values) # <class 'numpy.ndarray'>
print(df.values)
```

```
[['Guardians of the Galaxy' 'Action,Adventure,Sci-Fi'
  'A group of intergalactic criminals are forced to work together to stop a
fanatical warrior from taking control of the universe.'
  … 757074 333.13 76.0]
 ['Prometheus' 'Adventure,Mystery,Sci-Fi'
  'Following clues to the origin of mankind, a team finds a structure on a
distant moon, but they soon realize they are not alone.'
  … 485820 126.46 65.0]
 ['Split' 'Horror,Thriller'
  'Three girls are kidnapped by a man with a diagnosed 23 distinct
personalities. They must try to escape before the apparent emergence of a
frightful new 24th.'
  … 157606 138.12 62.0]
 …
 ['Step Up 2: The Streets' 'Drama,Music,Romance'
  'Romantic sparks occur between two dance students from different backgrounds
at the Maryland School of the Arts.'
  … 70699 58.01 50.0]
 ['Search Party' 'Adventure,Comedy'
  'A pair of friends embark on a mission to reunite their pal with the woman he
was going to marry.'
  … 4881 nan 22.0]
 ['Nine Lives' 'Comedy,Family,Fantasy'
```

```
  "A stuffy businessman finds himself trapped inside the body of his family's
cat."
  … 12435 19.64 11.0]]
```

1.5 DataFrame Columns

`df.columns` returns the list of columns in the dataframe.

[30]:
```
df.columns
```

[30]:
```
Index(['Title', 'Genre', 'Description', 'Director', 'Actors', 'Year',
       'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)',
       'Metascore'],
      dtype='object')
```

Columns can be renamed with `dframe.rename()` method.

[31]:
```
df.rename(columns={"Runtime (Minutes)": "Runtime_mins",
                   "Revenue (Millions)": "Revenue_mils"
                  }, inplace=True)
df.columns
```

[31]:
```
Index(['Title', 'Genre', 'Description', 'Director', 'Actors', 'Year',
       'Runtime_mins', 'Rating', 'Votes', 'Revenue_mils', 'Metascore'],
      dtype='object')
```

1.5.1 Accessing DataFrame Columns

A single column can be can be accessed as a Series object by passing the column name to the dataframe.

Multiple columns can be accessed by passing a list of column names.

[32]:
```
genre_col = df["Genre"]
type(genre_col)         # <class 'pandas.core.series.Series'>
genre_col.head(3)
```

[32]:
```
Rank
1     Action,Adventure,Sci-Fi
2    Adventure,Mystery,Sci-Fi
3              Horror,Thriller
Name: Genre, dtype: object
```

[33]:
```
df_columns = df[["Genre", "Year", "Runtime_mins"]]
df_columns.head(3)
```

[33]:

Rank	Genre	Year	Runtime_mins
1	Action,Adventure,Sci-Fi	2014	121

```
2     Adventure,Mystery,Sci-Fi  2012           124
3               Horror,Thriller  2016           117
```

1.6 DataFrame Elements

Elements in a DataFrame can be accessed in two ways

- `df.loc[label]`: by the index label, like a dictionary
- `df.iloc[pos]`: byu the position (row no.), like a list

[34]:
```
df = pd.read_csv("data/IMDB-Movie-Data.csv", index_col="Title")
'''
                     Title  ...
0  Guardians of the Galaxy  ...
1               Prometheus  ...
2                    Split  ...
3                     Sing  ...
4            Suicide Squad  ...
5          The Great Wall  ...
6               La La Land  ...
7                 Mindhorn  ...
8      The Lost City of Z  ...
9               Passengers  ...
'''

# Find La La Land by its index attribute
df.loc["La La Land"]

# Find La La Land by its row position
df.iloc[6]
```

[34]:
```
Rank                                                                7
Genre                                                 Comedy,Drama,Music
Description           A jazz pianist falls for an aspiring actress i…
Director                                                 Damien Chazelle
Actors                Ryan Gosling, Emma Stone, Rosemarie DeWitt, J…
Year                                                             2016
Runtime (Minutes)                                                 128
Rating                                                           8.30
Votes                                                          258682
Revenue (Millions)                                             151.06
Metascore                                                       93.00
Name: La La Land, dtype: object
```

Slicing also works for both df.loc() and df.iloc().

[35]:
```
df.loc["Prometheus":"Suicide Squad"]
df.iloc[1:4].iloc[:, :2]    # final iloc for displaying purposes
```

[35]:
Title	Rank	Genre
Prometheus	2	Adventure,Mystery,Sci-Fi
Split	3	Horror,Thriller
Sing	4	Animation,Comedy,Family

You can also access specific rows and/or columns.

[36]:
```
# All rows, one column
all_years = df.loc[:, "Year"]

# All rows, multiple columns
all_years_and_directors = df.loc[:, ["Year", "Director"]]

# Multiple rows, multiple columns
some_movies = df.loc[["Inception", "Interstellar"], ["Year", "Director"]]
```

1.7 DataFrame Iterations

1.7.1 Iterating over Columns

Iterating over the dataframe itself gives column names.

[37]:
```
for col in df:
    print(col)
```

```
Rank
Genre
Description
Director
Actors
Year
Runtime (Minutes)
Rating
Votes
Revenue (Millions)
Metascore
```

1.7.2 Iterating over Rows

There are several options for iterating over rows.

[38]:
```
df_subset = df.iloc[:3, :2]
df_subset
```

[38]:
Title	Rank	Genre
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi
Prometheus	2	Adventure,Mystery,Sci-Fi

```
Split                        3           Horror,Thriller
```

[39]:
```
# df.iteritems()
for (key, value) in df_subset.iteritems():
    print(key)
    print(value, end='\n\n')
```

```
Rank
Title
Guardians of the Galaxy    1
Prometheus                 2
Split                      3
Name: Rank, dtype: int64

Genre
Title
Guardians of the Galaxy     Action,Adventure,Sci-Fi
Prometheus                 Adventure,Mystery,Sci-Fi
Split                              Horror,Thriller
Name: Genre, dtype: object
```

[40]:
```
# df.iterrows()
for (row_index, row) in df_subset.iterrows():
    print(row_index)
    print(row, end='\n\n')
```

```
Guardians of the Galaxy
Rank                           1
Genre    Action,Adventure,Sci-Fi
Name: Guardians of the Galaxy, dtype: object

Prometheus
Rank                            2
Genre    Adventure,Mystery,Sci-Fi
Name: Prometheus, dtype: object

Split
Rank                    3
Genre    Horror,Thriller
Name: Split, dtype: object
```

[41]:
```
# df.itertuples()
for row in df_subset.itertuples():
    print(row, end='\n\n')
```

```
Pandas(Index='Guardians of the Galaxy', Rank=1, Genre='Action,Adventure,Sci-Fi')
```

```
Pandas(Index='Prometheus', Rank=2, Genre='Adventure,Mystery,Sci-Fi')

Pandas(Index='Split', Rank=3, Genre='Horror,Thriller')
```

1.8 DataFrame Filtering

You can filter the data by selecting a column and applying a condition on it, just like a numpy array.

[42]:
```
# Applying a condition returns a True/False DataFrame
(df["Director"] == "Ridley Scott").head(3)
```

[42]:
```
Title
Guardians of the Galaxy    False
Prometheus                  True
Split                      False
Name: Director, dtype: bool
```

[43]:
```
# Passing the True/False dataframe returns a filtered DataFrame
(df[df["Director"] == "Ridley Scott"]).head(3).iloc[:, :2]
```

[43]:
```
              Rank                     Genre
Title
Prometheus       2  Adventure,Mystery,Sci-Fi
The Martian    103    Adventure,Drama,Sci-Fi
Robin Hood     388    Action,Adventure,Drama
```

[44]:
```
# "Director" is "Christopher Nolan" or "Ridley Scott"
df[df["Director"].isin(["Christopher Nolan", "Ridley Scott"])].head(3).iloc[:, :
↪2]
```

[44]:
```
                  Rank                     Genre
Title
Prometheus           2  Adventure,Mystery,Sci-Fi
Interstellar        37    Adventure,Drama,Sci-Fi
The Dark Knight     55        Action,Crime,Drama
```

[45]:
```
# Selecting movies released between 2008-2010 with a rating above 8.3
# and returning only the year and rating
between_2008_2010 = (df["Year"] >= 2008) & (df["Year"] <= 2010)
rating_gte_8_3 = df["Rating"] >= 8.3
df_cond = df[between_2008_2010 & rating_gte_8_3]
df_select = df_cond[["Year", "Rating"]]
df_select
```

[45]:
```
                      Year  Rating
Title
The Dark Knight       2008     9.0
Inglourious Basterds  2009     8.3
Inception             2010     8.8
3 Idiots              2009     8.4
Up                    2009     8.3
WALL·E                2008     8.4
Toy Story 3           2010     8.3
```

1.9 DataFrame Information

1.9.1 Summary Statistics

`pd.describe()` returns summary statics of the Series or DataFrame.

[46]:
```
# Series with numeric data
s = pd.Series([1, 2, 3])
s.describe()
```

[46]:
```
count    3.0
mean     2.0
std      1.0
min      1.0
25%      1.5
50%      2.0
75%      2.5
max      3.0
dtype: float64
```

[47]:
```
# Series with categorical data
s = pd.Series(["a", "a", "b", "c"])
s.describe()
```

[47]:
```
count     4
unique    3
top       a
freq      2
dtype: object
```

`df.value_counts()` returns the frequency counts for each element.

[48]:
```
s = pd.Series(["a", "a", "b", "a", "c", "b", "d"])
s.value_counts()
```

[48]:
```
a    3
b    2
d    1
```

```
c    1
dtype: int64
```

[49]:
```
df["Genre"].value_counts()
```

[49]:
```
Action,Adventure,Sci-Fi    50
Drama                      48
Comedy,Drama,Romance       35
Comedy                     32
Drama,Romance              31
                           ..
Comedy,Horror,Romance       1
Biography,Comedy,Crime      1
Animation,Drama,Fantasy     1
Drama,Western               1
Drama,Family,Music          1
Name: Genre, Length: 207, dtype: int64
```

1.10 Missing values

To deal with null/missing values, you can * remove the rows/columns with nulls * replace nulls with non-null values (imputation)

`df.isnull()` returns an array ob bools indicating whether each element is missing.

`df.isnull().sum()` returns the number of nulls in each column.

[50]:
```
df.isnull().sum()
```

[50]:
```
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
```

1.10.1 Removing Missing Values

`df.dropna()` drops the entire rows/columns that contain null values (depending on the axis parameter). Adding inplace=True as a parameter to .dropna() will modify df directly.

[51]:
```
df.shape    # (1000, 11)

# Drops the entire row with null values
clean_df = df.dropna()  # Using df.dropna(inplace=True) will modify df directly
clean_df.shape
```

[51]: (838, 11)

[52]:
```
df.shape    # (1000, 11)

# Drops the entire column with null values
clean_df = df.dropna(axis=1)
clean_df.shape
```

[52]: (1000, 9)

1.10.2 Replacing Missing Values

`df.fillna()` replaces all null values with the supplied value. The nulls are generally replaced with the mean or median value of the column.

[53]:
```
# Get the revenues column
revenues = df["Revenue (Millions)"]

# Replace missing revenues with the mean revenue
revenues.fillna(revenues.mean(), inplace=True)
```

1.11 Miscellaneous

1.11.1 `.append()`

`df.append(other)` returns a new dataframe with `other` appended to `df`. Other can be any dataframe, series, dict-like object, or a list of these.

1.11.2 `.drop_duplicates()`

`df.drop_duplicates()` returns a new dataframe with duplicates removed.

1.11.3 `.apply(func, axis= )`

`df.apply(func, axis= )` applies a function to the given `df`, either for each row or column depending on the axis parameter. This method is more efficient than manually iterating over the DataFrame.

It is useful when you want to create a new column with the given values.

[54]:
```
def rotten_tomatoes(rating):
    if rating >= 8.0:
        return "fresh"
```

```
    else:
        return "rotten"

df["RT"] = df["Rating"].apply(rotten_tomatoes)

df[["Rating", "RT"]].head(4)
```

[54]:

Title	Rating	RT
Guardians of the Galaxy	8.1	fresh
Prometheus	7.0	rotten
Split	7.3	rotten
Sing	7.2	rotten