Pandas

What is Pandas?

Pandas is a Python library used for working with data sets.
It has functions for analyzing, cleaning, exploring, and manipulating data.
The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories.
Pandas can clean messy data sets, and make them readable and relevant.
Relevant data is very important in data science.

What Can Pandas Do?

Pandas gives you answers about the data. Like:

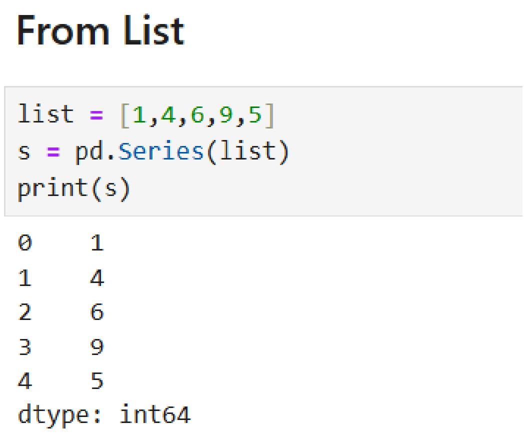
- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Pandas Series

What is a Series?

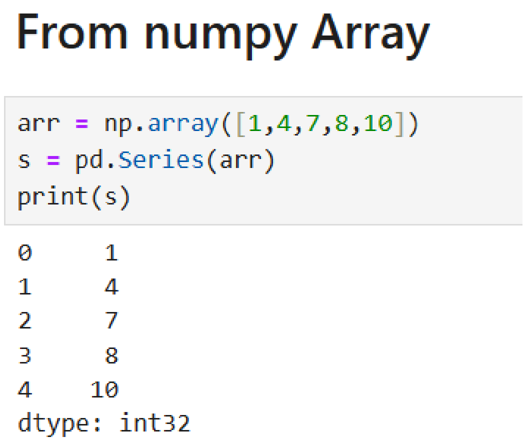
A Pandas Series is like a column in a table.
It is a one-dimensional array holding data of any type.



From List

```
list = [1,4,6,9,5]
s = pd.Series(list)
print(s)
```

```
0    1
1    4
2    6
3    9
4    5
dtype: int64
```

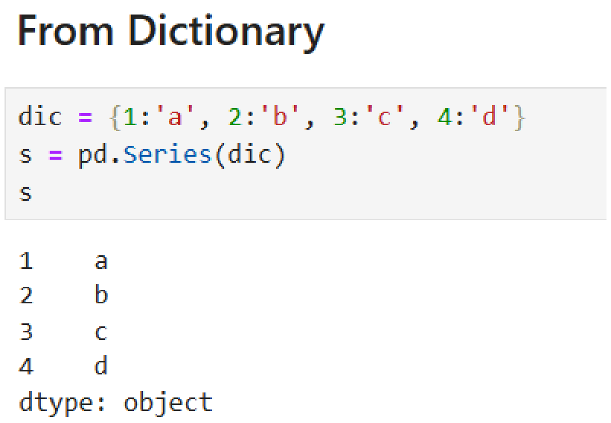


From numpy Array

```
arr = np.array([1,4,7,8,10])
s = pd.Series(arr)
print(s)
```

```
0     1
1     4
2     7
3     8
4    10
dtype: int32
```

,



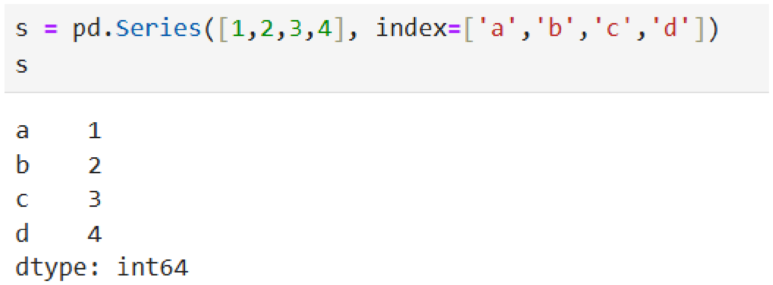
From Dictionary

```
dic = {1:'a', 2:'b', 3:'c', 4:'d'}
s = pd.Series(dic)
s
```

```
1    a
2    b
3    c
4    d
dtype: object
```

Create Labels

With the index argument, you can name your own labels.

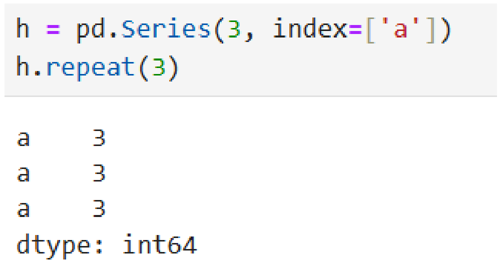


```
s = pd.Series([1,2,3,4], index=['a','b','c','d'])
s
```

```
a    1
b    2
c    3
d    4
dtype: int64
```

Using repeat function along with creating a Series

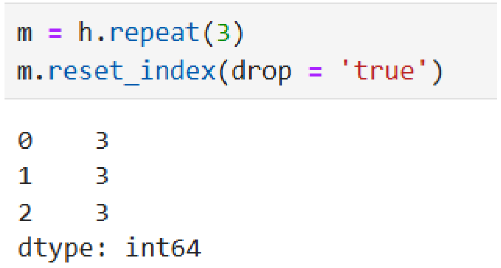
Pandas Series.repeat() function repeat elements of a Series. It returns a new Series where each element of the current Series is repeated consecutively a given number of times.



```
h = pd.Series(3, index=['a'])
h.repeat(3)
```

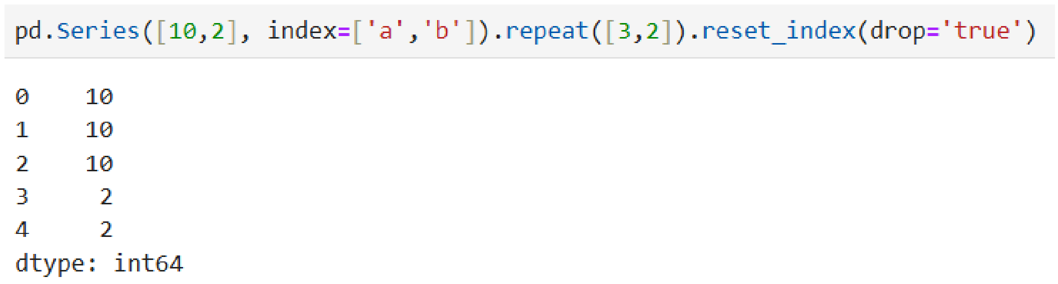
```
a    3
a    3
a    3
dtype: int64
```

we can use the reset function to make the index accurate



```
m = h.repeat(3)
m.reset_index(drop = 'true')
```

```
0    3
1    3
2    3
dtype: int64
```

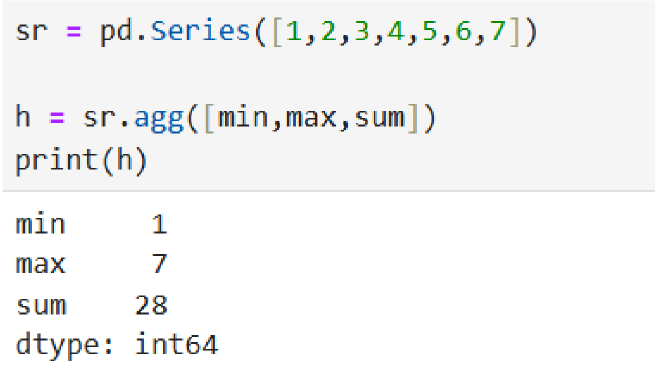


```
pd.Series([10,2], index=['a','b']).repeat([3,2]).reset_index(drop='true')
```

```
0    10
1    10
2    10
3     2
4     2
dtype: int64
```

Aggregate function on pandas Series

Pandas Series.aggregate() function aggregate using one or more operations over the specified axis in the given series object.

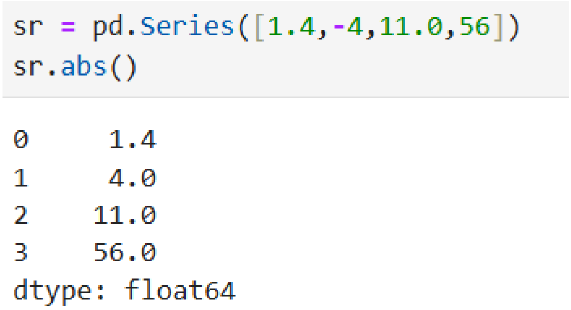


```
sr = pd.Series([1,2,3,4,5,6,7])

h = sr.agg([min,max,sum])
print(h)
```

```
min     1
max     7
sum    28
dtype: int64
```

Series absolute function

Pandas.Series.abs() method is used to get the absolute numeric value of each element in Series/DataFrame.



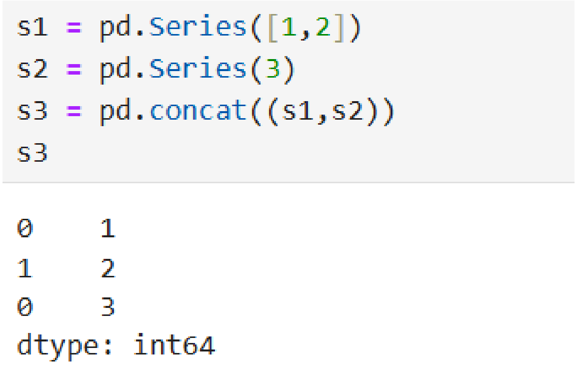
```
sr = pd.Series([1.4,-4,11.0,56])
sr.abs()
```

```
0     1.4
1     4.0
2    11.0
3    56.0
dtype: float64
```

Concatenate Series

pd.concat() function is used to concatenate two or more series object.

Syntax: pd.concat(*objs)

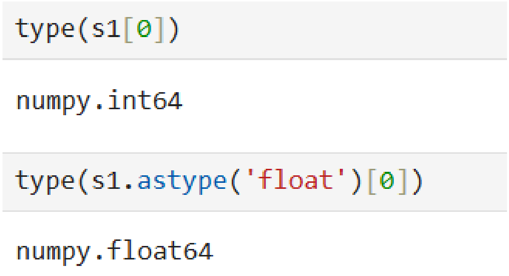


```
s1 = pd.Series([1,2])
s2 = pd.Series(3)
s3 = pd.concat((s1,s2))
s3
```

```
0    1
1    2
0    3
dtype: int64
```

Astype function

Pandas astype() is the one of the most important methods. It is used to change data type of a series. When data frame is made from a csv file, the columns are imported and data type is set automatically which many times is not what it actually should have.



```
type(s1[0])
```

```
numpy.int64
```

```
type(s1.astype('float')[0])
```

```
numpy.float64
```

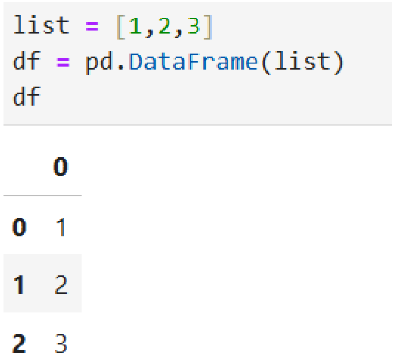
DataFrames

What is a DataFrame?

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

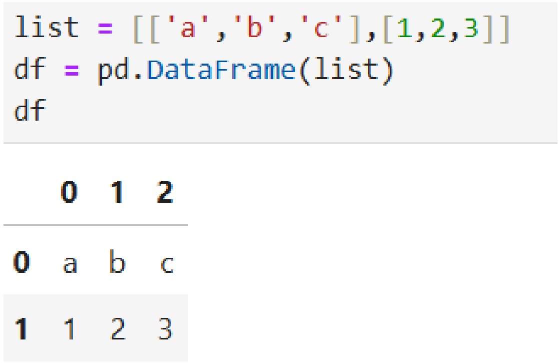
Creating a dataframe

DataFrame can be created using a list or a list, ndarray or dictionaries.



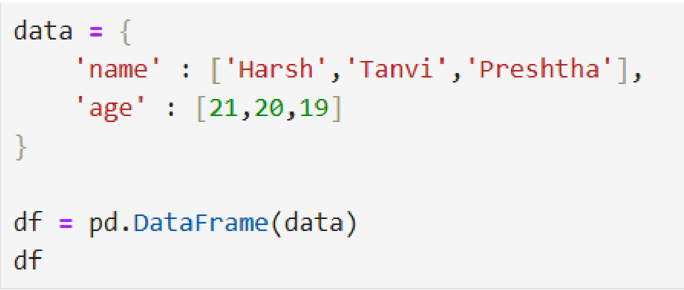
```
list = [1,2,3]
df = pd.DataFrame(list)
df
```

	0
0	1
1	2
2	3

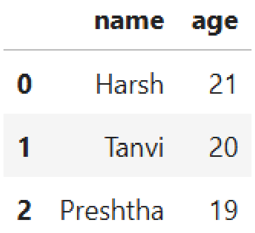


```
list = [['a','b','c'],[1,2,3]]
df = pd.DataFrame(list)
df
```

	0	1	2
0	a	b	c
1	1	2	3



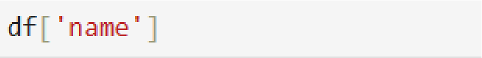
```
data = {
    'name' : ['Harsh','Tanvi','Preshtha'],
    'age' : [21,20,19]
}

df = pd.DataFrame(data)
df
```



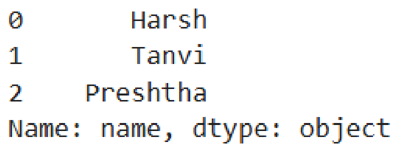
	name	age
0	Harsh	21
1	Tanvi	20
2	Preshtha	19

Column Selection:

In Order to select a column in Pandas DataFrame, we can either access the columns by calling them by their columns name.



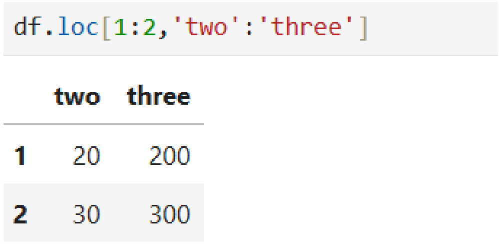
```
df['name']
```



```
0       Harsh
1       Tanvi
2    Preshtha
Name: name, dtype: object
```

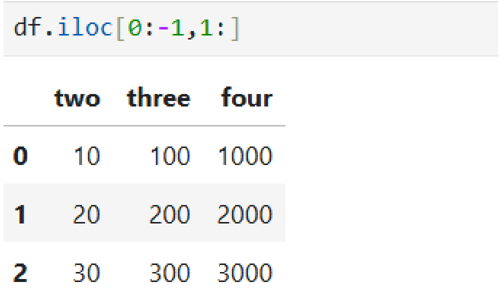
Slicing in DataFrames Using iloc and loc

Pandas comprises many methods for its proper functioning. loc() and iloc() are one of those methods. These are used in slicing data from the Pandas DataFrame. They help in the convenient selection of data from the DataFrame in Python. They are used in filtering the data according to some conditions.



```
df.loc[1:2,'two':'three']
```

	two	three
1	20	200
2	30	300



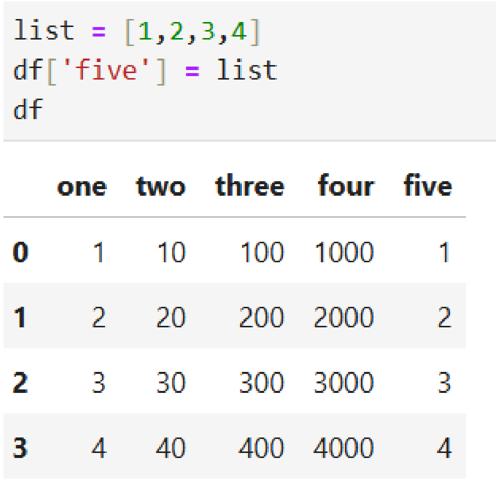
```
df.iloc[0:-1,1:]
```

	two	three	four
0	10	100	1000
1	20	200	2000
2	30	300	3000

Column Addition in DataFrame

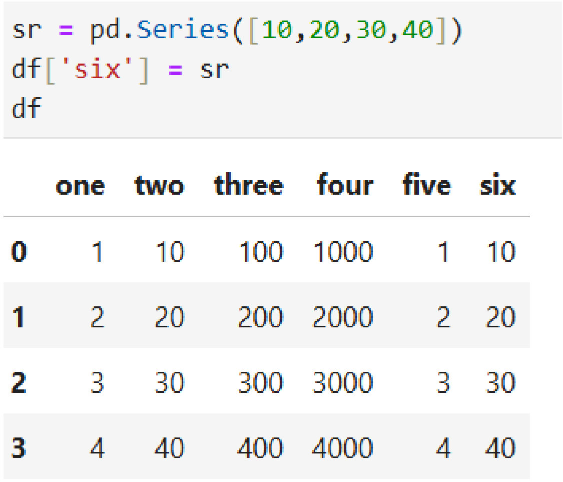
We can add a column in many ways. Let us discuss three ways how we can add column here

- Using List
- Using Pandas Series
- Using an existing Column(we can modify that column in the way we want and that modified part can also be displayed)



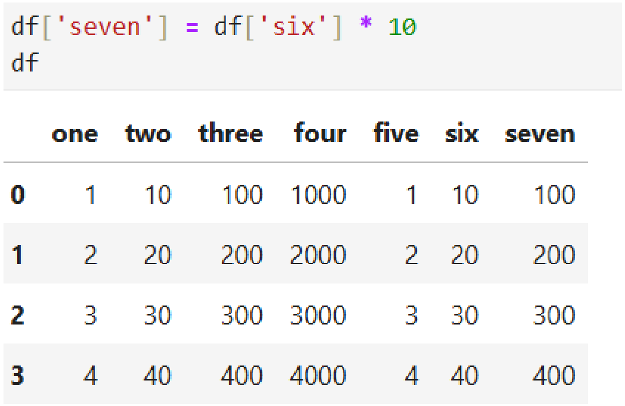
```
list = [1,2,3,4]
df['five'] = list
df
```

	one	two	three	four	five
0	1	10	100	1000	1
1	2	20	200	2000	2
2	3	30	300	3000	3
3	4	40	400	4000	4



```
sr = pd.Series([10,20,30,40])
df['six'] = sr
df
```

	one	two	three	four	five	six
0	1	10	100	1000	1	10
1	2	20	200	2000	2	20
2	3	30	300	3000	3	30
3	4	40	400	4000	4	40



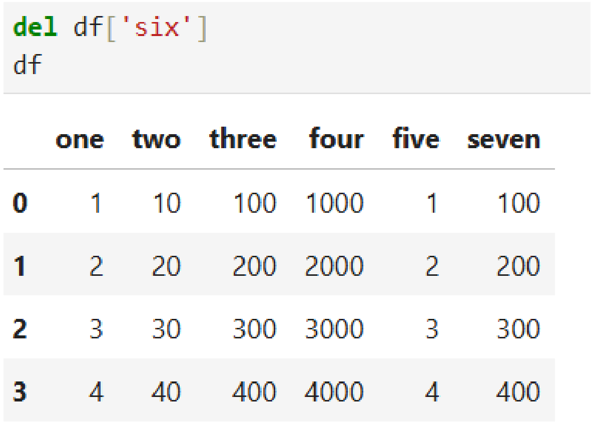
```
df['seven'] = df['six'] * 10
df
```

	one	two	three	four	five	six	seven
0	1	10	100	1000	1	10	100
1	2	20	200	2000	2	20	200
2	3	30	300	3000	3	30	300
3	4	40	400	4000	4	40	400

Column Deletion in Dataframes

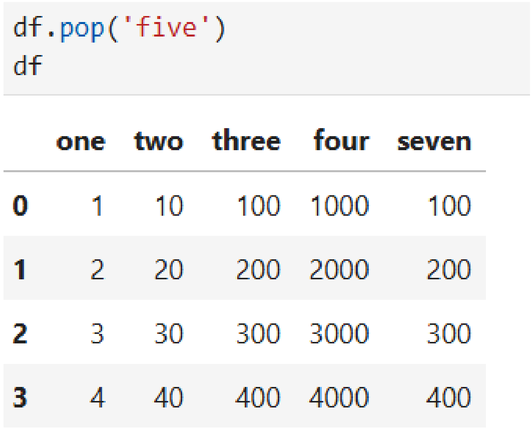
2 ways to delete a column

- using del
- using pop



```
del df['six']
df
```

	one	two	three	four	five	seven
0	1	10	100	1000	1	100
1	2	20	200	2000	2	200
2	3	30	300	3000	3	300
3	4	40	400	4000	4	400



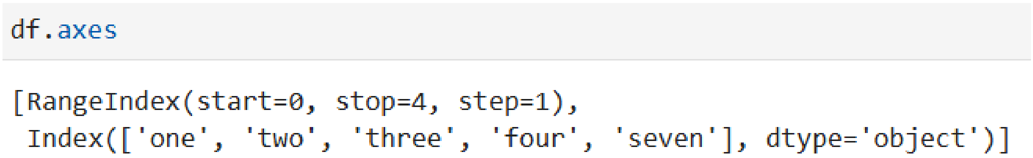
```
df.pop('five')
df
```

	one	two	three	four	seven
0	1	10	100	1000	100
1	2	20	200	2000	200
2	3	30	300	3000	300
3	4	40	400	4000	400

More DataFrame Functionalities

axes function

The .axes attribute in a Pandas DataFrame returns a list with the row and column labels of the DataFrame. The first element of the list is the row labels (index), and the second element is the column labels.

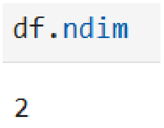


```
df.axes
```

```
[RangeIndex(start=0, stop=4, step=1),
 Index(['one', 'two', 'three', 'four', 'seven'], dtype='object')]
```

ndim function

The .ndim attribute in a Pandas DataFrame returns the number of dimensions of the dataframe, which is always 2 for a DataFrame (row-and-column format).

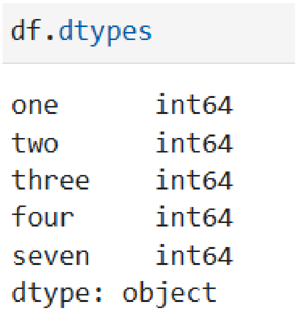


```
df.ndim
```

```
2
```

dtypes

The .dtypes attribute in a Pandas DataFrame returns the data types of the columns in the DataFrame. The result is a Series with the column names as index and the data types of the columns as values.

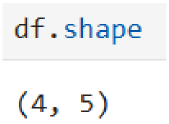


```
df.dtypes
```

```
one      int64
two      int64
three    int64
four     int64
seven    int64
dtype: object
```

shape function

The .shape attribute in a Pandas DataFrame returns the dimensions (number of rows, number of columns) of the DataFrame as a tuple.



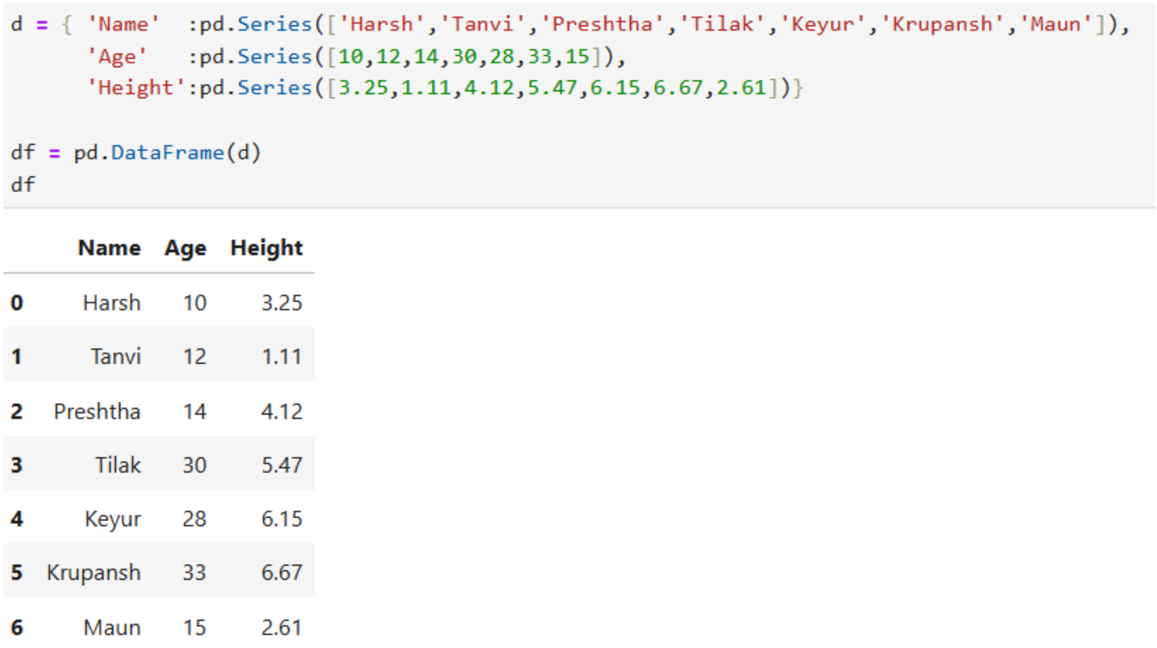
```
df.shape
```

```
(4, 5)
```

head() function

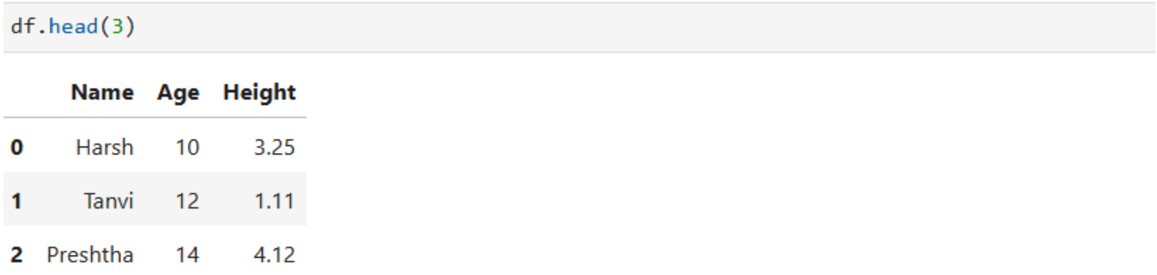
The .head() method in a Pandas DataFrame returns the first n rows (by default, n=5) of the DataFrame. This method is useful for quickly examining the first few rows of a large DataFrame to get a sense of its structure and content.

- By default it will display first 5 rows
- We can mention the number of starting rows we want to see
- We will see this function more often further since the dataframe is so small at this point so we cannot use something like df.head(20)



```
d = { 'Name'   :pd.Series(['Harsh','Tanvi','Preshtha','Tilak','Keyur','Krupansh','Maun']),
      'Age'    :pd.Series([10,12,14,30,28,33,15]),
      'Height':pd.Series([3.25,1.11,4.12,5.47,6.15,6.67,2.61])}

df = pd.DataFrame(d)
df
```

	Name	Age	Height
0	Harsh	10	3.25
1	Tanvi	12	1.11
2	Preshtha	14	4.12
3	Tilak	30	5.47
4	Keyur	28	6.15
5	Krupansh	33	6.67
6	Maun	15	2.61

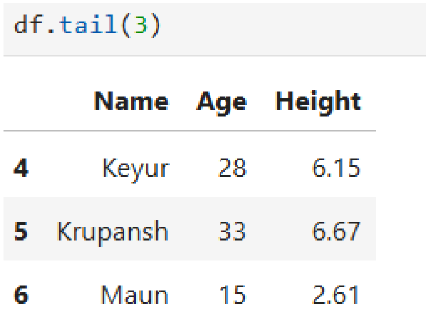


```
df.head(3)
```

	Name	Age	Height
0	Harsh	10	3.25
1	Tanvi	12	1.11
2	Preshtha	14	4.12

df.tail() function

The .tail() method in a Pandas DataFrame returns the last n rows (by default, n=5) of the DataFrame. This method is useful for quickly examining the last few rows of a large DataFrame to get a sense of its structure and content.



```
df.tail(3)
```

	Name	Age	Height
4	Keyur	28	6.15
5	Krupansh	33	6.67
6	Maun	15	2.61

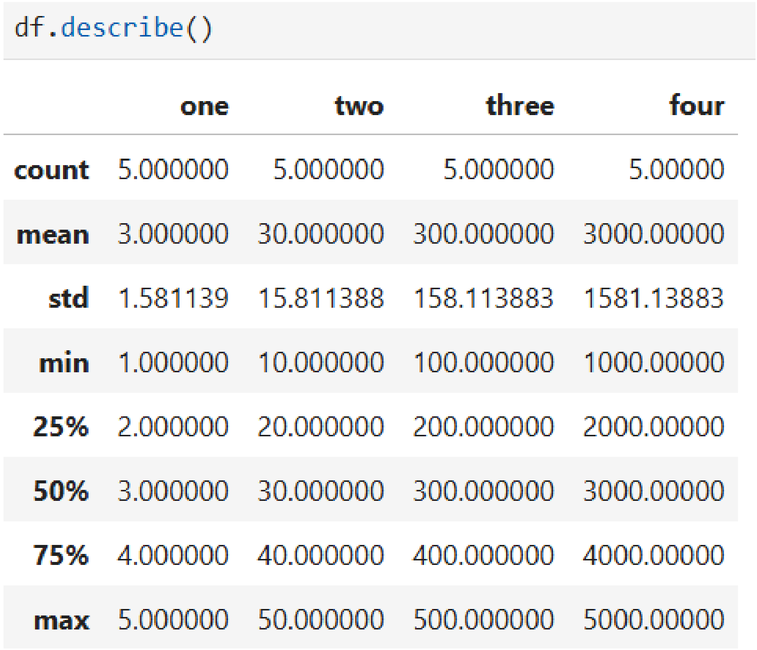
Statistical or Mathematical Functions

- Sum [df.sum()]
- Mean [df.mean()]
- Median [df.median()]
- Mode [df.mode()]
- Variance [df.var()]
- Min [df.min()]
- Max [df.max()]
- Standard Deviation [df.std()]

Describe Function

The describe() method in a Pandas DataFrame returns descriptive statistics of the data in the DataFrame. It provides a quick summary of the central tendency, dispersion, and shape of the distribution of a set of numerical data.

The default behavior of describe() is to compute descriptive statistics for all numerical columns in the DataFrame. If you want to compute descriptive statistics for a specific column, you can pass the name of the column as an argument.



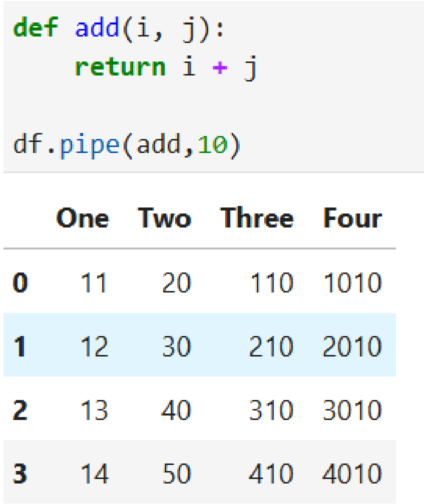
```
df.describe()
```

	one	two	three	four
count	5.000000	5.000000	5.000000	5.00000
mean	3.000000	30.000000	300.000000	3000.00000
std	1.581139	15.811388	158.113883	1581.13883
min	1.000000	10.000000	100.000000	1000.00000
25%	2.000000	20.000000	200.000000	2000.00000
50%	3.000000	30.000000	300.000000	3000.00000
75%	4.000000	40.000000	400.000000	4000.00000
max	5.000000	50.000000	500.000000	5000.00000

Pipe Functions

1. Pipe Function

The pipe() method in a Pandas DataFrame allows you to apply a function to the DataFrame, similar to the way the apply() method works. The difference is that pipe() allows you to chain multiple operations together by passing the output of one function to the input of the next function.

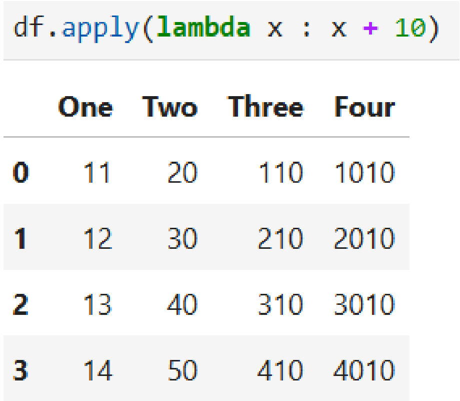


```
def add(i, j):
    return i + j

df.pipe(add,10)
```

	One	Two	Three	Four
0	11	20	110	1010
1	12	30	210	2010
2	13	40	310	3010
3	14	50	410	4010

2. Apply Function

The apply() method in a Pandas DataFrame allows you to apply a function to the DataFrame, either to individual elements or to the entire DataFrame. The function can be either a built-in Python function or a user-defined function.

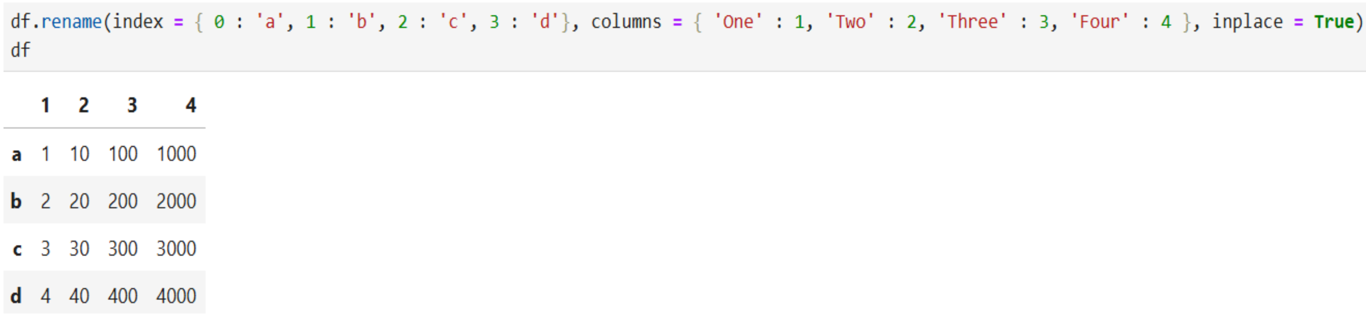


```
df.apply(lambda x : x + 10)
```

	One	Two	Three	Four
0	11	20	110	1010
1	12	30	210	2010
2	13	40	310	3010
3	14	50	410	4010

Renaming Columns in Pandas DataFrame

The rename function in Pandas is used to change the row labels and/or column labels of a DataFrame. It can be used to update the names of one or multiple rows or columns by passing a dictionary of new names as its argument. The dictionary should have the old names as keys and the new names as values.



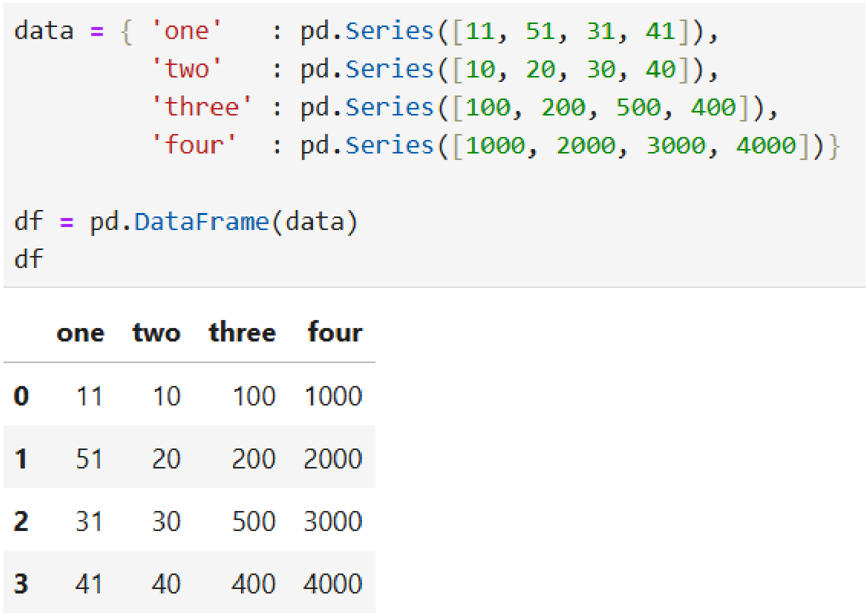
```
df.rename(index = { 0 : 'a', 1 : 'b', 2 : 'c', 3 : 'd'}, columns = { 'One' : 1, 'Two' : 2, 'Three' : 3, 'Four' : 4 }, inplace = True)
df
```

	1	2	3	4
a	1	10	100	1000
b	2	20	200	2000
c	3	30	300	3000
d	4	40	400	4000

Sorting in Pandas DataFrame

Pandas provides several methods to sort a DataFrame based on one or more columns.

- sort_values: This method sorts the DataFrame based on one or more columns. The default sorting order is ascending, but you can change it to descending by passing the ascending argument with a value of False.



```
data = { 'one'   : pd.Series([11, 51, 31, 41]),
         'two'   : pd.Series([10, 20, 30, 40]),
         'three' : pd.Series([100, 200, 500, 400]),
         'four'  : pd.Series([1000, 2000, 3000, 4000])}

df = pd.DataFrame(data)
df
```

	one	two	three	four
0	11	10	100	1000
1	51	20	200	2000
2	31	30	500	3000
3	41	40	400	4000

```
[127]: df.sort_values(by = 'one', ascending = False)
```

[127]:

	one	two	three	four
1	51	20	200	2000
3	41	40	400	4000
2	31	30	500	3000
0	11	10	100	1000

```
[129]: df.sort_values(by = ['one'], kind = 'quicksort')
```

[129]:

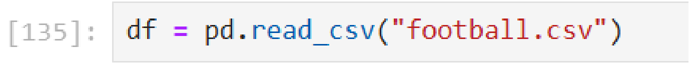
	one	two	three	four
0	11	10	100	1000
2	31	30	500	3000
3	41	40	400	4000
1	51	20	200	2000

Working with CSV Files :

Reading csv from Local

Reading csv files from local system.

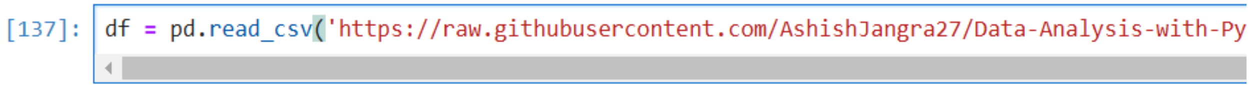
df = pd.read_csv(“file_name.csv”)



```
[135]: df = pd.read_csv("football.csv")
```

Reading csv file from link

df = pd.read_csv(“link”)

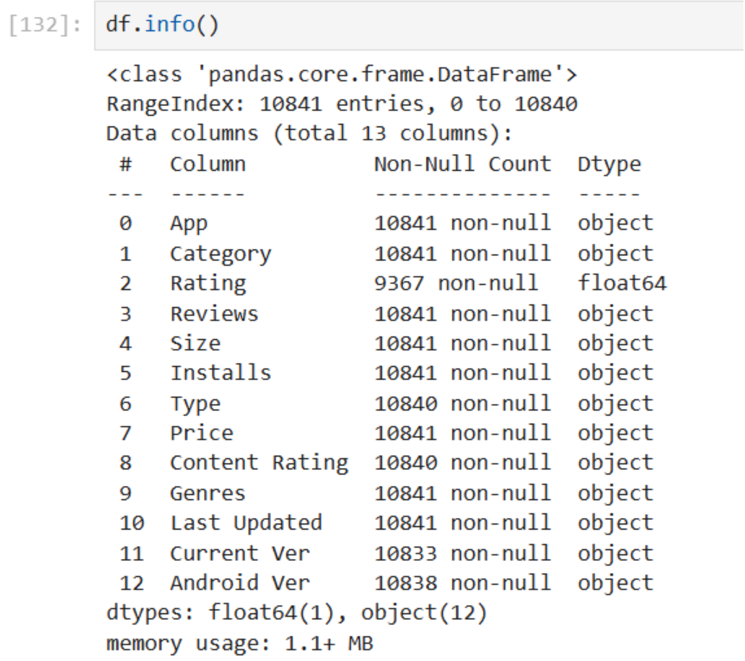


```
[137]: df = pd.read_csv('https://raw.githubusercontent.com/AshishJangra27/Data-Analysis-with-Py
```

Info Function

Pandas dataframe.info() function is used to get a concise summary of the dataframe. It comes really handy when doing exploratory analysis of the data. To get a quick overview of the dataset we use the dataframe.info() function.

Syntax: DataFrame.info(verbose=None, buf=None, max_cols=None, memory_usage=None, null_counts=None)

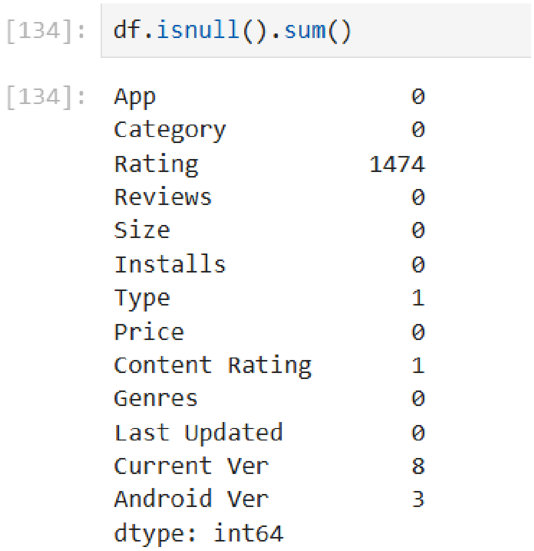


```
[132]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10841 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   App             10841 non-null  object 
 1   Category        10841 non-null  object 
 2   Rating          9367 non-null   float64
 3   Reviews         10841 non-null  object 
 4   Size            10841 non-null  object 
 5   Installs        10841 non-null  object 
 6   Type            10840 non-null  object 
 7   Price           10841 non-null  object 
 8   Content Rating  10840 non-null  object 
 9   Genres          10841 non-null  object 
 10  Last Updated    10841 non-null  object 
 11  Current Ver     10833 non-null  object 
 12  Android Ver     10838 non-null  object 
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
```

isnull()

function to check if there are nan values present

So we will get a boolean kind of a table giving True and False
If we use the sum function along with it then we can get how many null values are present in each columns

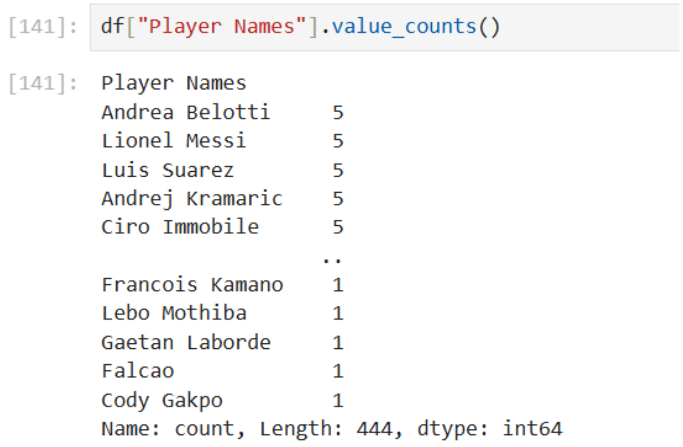


```
[134]: df.isnull().sum()

[134]: App                  0
       Category             0
       Rating            1474
       Reviews              0
       Size                 0
       Installs             0
       Type                 1
       Price                0
       Content Rating       1
       Genres               0
       Last Updated         0
       Current Ver          8
       Android Ver          3
       dtype: int64
```

Value Counts function

Pandas Series.value_counts() function return a Series containing counts of unique values. The resulting object will be in descending order so that the first element is the most frequently-occurring element. Excludes NA values by default.

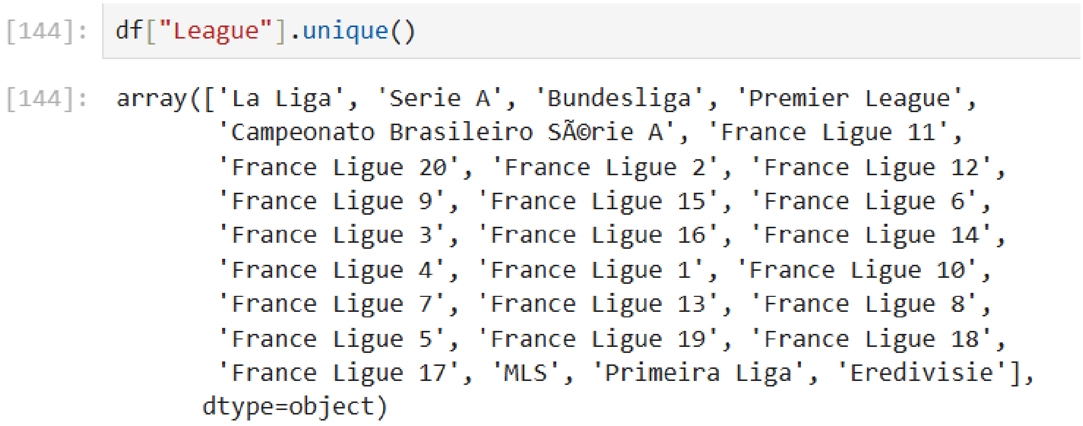
Syntax: Series.value_counts(normalize=False, sort=True, ascending=False, bins=None, dropna=True)



```
[141]: df["Player Names"].value_counts()

[141]: Player Names
       Andrea Belotti     5
       Lionel Messi       5
       Luis Suarez        5
       Andrej Kramaric    5
       Ciro Immobile      5
                         ..
       Francois Kamano    1
       Lebo Mothiba       1
       Gaetan Laborde     1
       Falcao             1
       Cody Gakpo         1
       Name: count, Length: 444, dtype: int64
```

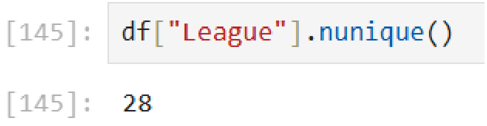
Unique and Nunique Function

While analyzing the data, many times the user wants to see the unique values in a particular column, which can be done using Pandas unique() function.



```
[144]:  df["League"].unique()

[144]:  array(['La Liga', 'Serie A', 'Bundesliga', 'Premier League',
               'Campeonato Brasileiro SÃ©rie A', 'France Ligue 11',
               'France Ligue 20', 'France Ligue 2', 'France Ligue 12',
               'France Ligue 9', 'France Ligue 15', 'France Ligue 6',
               'France Ligue 3', 'France Ligue 16', 'France Ligue 14',
               'France Ligue 4', 'France Ligue 1', 'France Ligue 10',
               'France Ligue 7', 'France Ligue 13', 'France Ligue 8',
               'France Ligue 5', 'France Ligue 19', 'France Ligue 18',
               'France Ligue 17', 'MLS', 'Primeira Liga', 'Eredivisie'],
              dtype=object)
```

While analyzing the data, many times the user wants to see the unique values in a particular column. Pandas nunique() is used to get a count of unique values.



```
[145]:  df["League"].nunique()

[145]:  28
```

dropna() function

Sometimes csv file has null values, which are later displayed as NaN in Data Frame. Pandas dropna() method allows the user to analyze and drop Rows/Columns with Null values in different ways.

Syntax :

DataFrameName.dropna(axis=0,inplace=False)

```
[149]: df.dropna(inplace = True, axis = 0)
```

```
[150]: df.dropna(inplace = True, axis = 1)
```

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

```
[151]: App               0
       Category          0
       Rating            0
       Reviews           0
       Size              0
       Installs          0
       Type              0
       Price             0
       Content Rating    0
       Genres            0
       Last Updated      0
       Current Ver       0
       Android Ver       0
       dtype: int64
```

Fillna Function

Pandas Series.fillna() function is used to fill NA/NaN values using the specified method.

Suppose if we want to fill the null values with something instead of removing them then we can use fillna function
Here we will be filling the numerical columns with its mean values and Categorical columns with its mode.

```
[161]: mis = round(df['Rating'].mean(),2)

       df['Rating'] = df['Rating'].fillna(mis)

       print(len(df))
```

```
10841
```

```
[162]: df['Current Ver'] = df['Current Ver'].fillna('Varies on Device')
```

to_csv() function

Pandas Series.to_csv() function write the given series object to a comma-separated values (csv) file/format.

Syntax: Series.to_csv(*args, **kwargs)

Duplicates

To discover duplicates, we can use the duplicated() method.

The duplicated() method returns a Boolean values for each row.

Returns True for every row that is a duplicate, otherwise False.

```
print(df.duplicated())
```

Removing Duplicates

To remove duplicates, use the drop_duplicates() method.

```
df.drop_duplicates(inplace = True)
```

Read JSON

Big data sets are often stored, or extracted as JSON.

JSON is plain text, but has the format of an object, and is well known in the world of programming, including Pandas.

In our examples we will be using a JSON file called 'data.json'.

```
df = pd.read_json('data.json')

print(df.to_string())
```