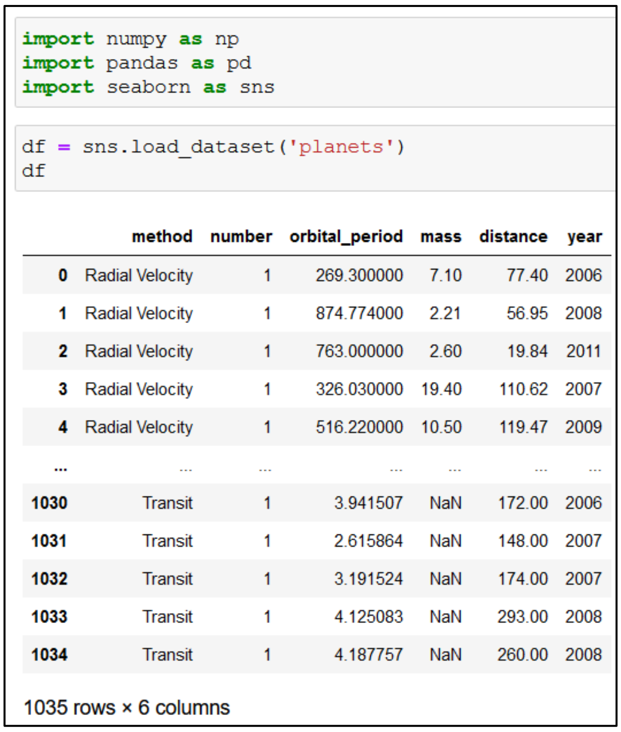
Objective: This and few other related worksheets introduces different methods how data sets are read into the Python environment and Pandas Data Frames are prepared from them. **These methods are widely utilized in the area of data science, so participants are expected to gain full familiarity and practice with these**. This and few other related worksheets use other Python packages to download the data sets. These packages include Seaborn and Scikit-Learn. Other features of these packages will covered in the relevant worksheets, but here they are limited for accessing the data sets.

- Seaborn package provide sample data sets, that are available in CSV format (Comma Separated Values) and they can be downloaded using ***load_dataset ()*** function of the package. This function primarily takes name of the data set as an argument and returns the Pandas DataFrame.
- In this worksheet ***planets.csv*** dataset of the Seaborn package will be downloaded.



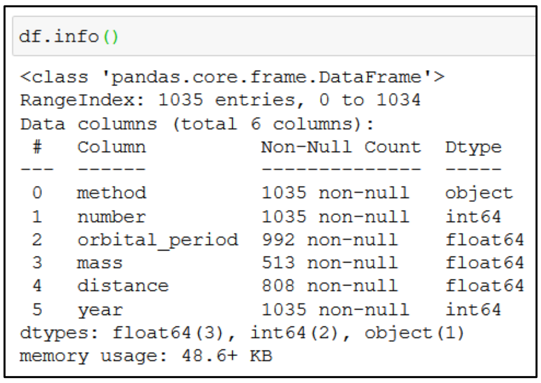
```
import numpy as np
import pandas as pd
import seaborn as sns
```

```
df = sns.load_dataset('planets')
df
```

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300000	7.10	77.40	2006
1	Radial Velocity	1	874.774000	2.21	56.95	2008
2	Radial Velocity	1	763.000000	2.60	19.84	2011
3	Radial Velocity	1	326.030000	19.40	110.62	2007
4	Radial Velocity	1	516.220000	10.50	119.47	2009
...	...	...	...	...	...	...
1030	Transit	1	3.941507	NaN	172.00	2006
1031	Transit	1	2.615864	NaN	148.00	2007
1032	Transit	1	3.191524	NaN	174.00	2007
1033	Transit	1	4.125083	NaN	293.00	2008
1034	Transit	1	4.187757	NaN	260.00	2008

1035 rows × 6 columns

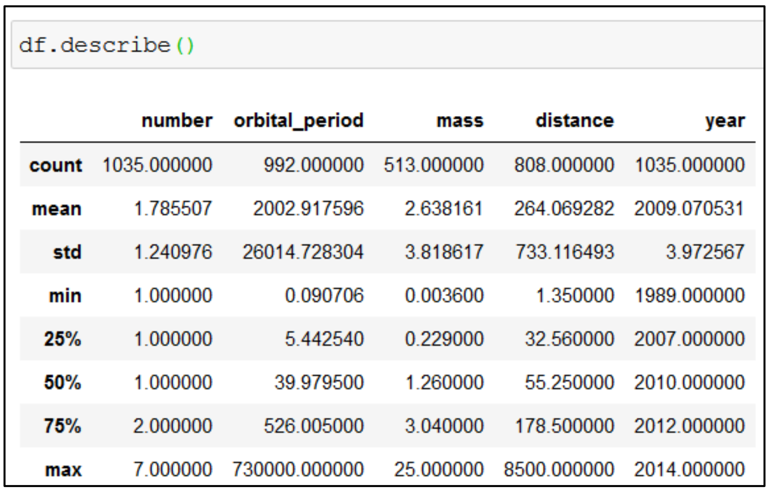
- As it can be observed, that the dataset contains 1035 records with 6 attribute columns. The downloaded dataset is stored in the ***df*** variable as DataFrame.
- The basic details of the dataset can be see using ***info ()*** function. Note that the data set occupies only 48.6 KB in the memory.



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1035 entries, 0 to 1034
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   method          1035 non-null   object
 1   number          1035 non-null   int64
 2   orbital_period  992 non-null    float64
 3   mass            513 non-null    float64
 4   distance        808 non-null    float64
 5   year            1035 non-null   int64
dtypes: float64(3), int64(2), object(1)
memory usage: 48.6+ KB
```

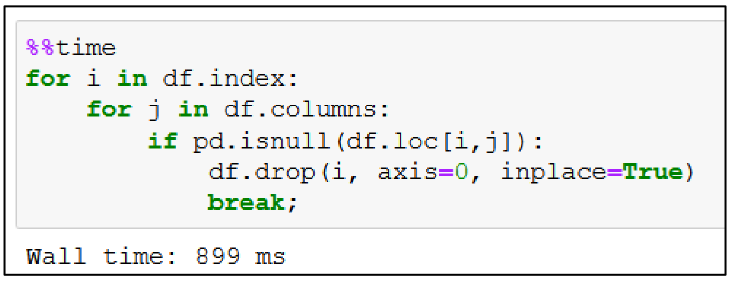
- The descriptive statistics can be seen using the ***describe ()*** function.
- Note that there are several cells in the data set which are having missing values (shown as NaN – Not a Number). Therefore count in the descripting statistics below is different for each attribute because the function excludes the NaN cells.
- Also note that the ***describe ()*** function is applied to only those attributes which are numeric. Therefore the description of ***method*** attribute is not shown below.



```
df.describe()
```

	number	orbital_period	mass	distance	year
count	1035.000000	992.000000	513.000000	808.000000	1035.000000
mean	1.785507	2002.917596	2.638161	264.069282	2009.070531
std	1.240976	26014.728304	3.818617	733.116493	3.972567
min	1.000000	0.090706	0.003600	1.350000	1989.000000
25%	1.000000	5.442540	0.229000	32.560000	2007.000000
50%	1.000000	39.979500	1.260000	55.250000	2010.000000
75%	2.000000	526.005000	3.040000	178.500000	2012.000000
max	7.000000	730000.000000	25.000000	8500.000000	2014.000000

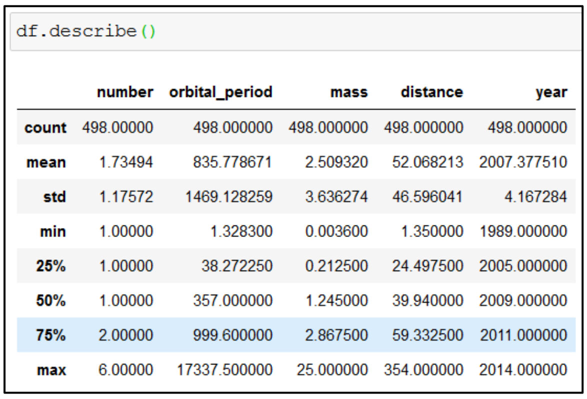
- It is a very common requirement to drop those rows from the dataset where any attribute value is missing. It can be done in the variety of ways but efficiency is the key. So let us review these methods and also measure their timings.
- The first method is to use the convectional method of iterating each row and each column of this row. If any attribute is missing then drop the entire row.



```
%%time
for i in df.index:
    for j in df.columns:
        if pd.isnull(df.loc[i,j]):
            df.drop(i, axis=0, inplace=True)
            break;
```

```
Wall time: 899 ms
```

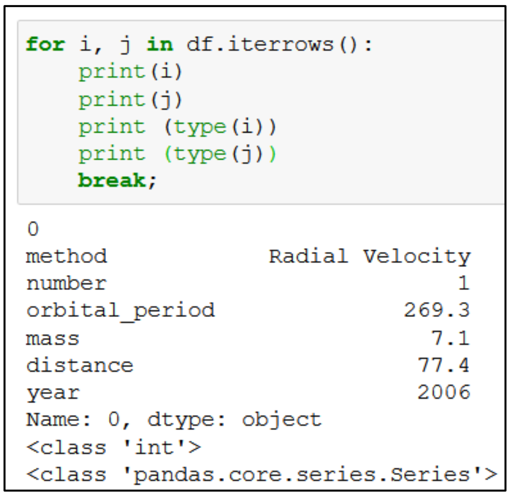
- It takes 899 ms. Let us also see what it does to the data frame. Notice that there are only 498 rows where there is no missing value for any column.



```
df.describe()
```

	number	orbital_period	mass	distance	year
count	498.00000	498.000000	498.000000	498.000000	498.000000
mean	1.73494	835.778671	2.509320	52.068213	2007.377510
std	1.17572	1469.128259	3.636274	46.596041	4.167284
min	1.00000	1.328300	0.003600	1.350000	1989.000000
25%	1.00000	38.272250	0.212500	24.497500	2005.000000
50%	1.00000	357.000000	1.245000	39.940000	2009.000000
75%	2.00000	999.600000	2.867500	59.332500	2011.000000
max	6.00000	17337.500000	25.000000	354.000000	2014.000000

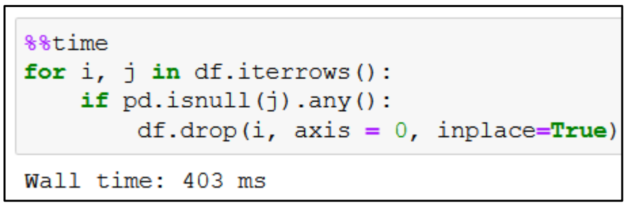
- The second method is to use the function ***iterrows().*** The function iteratively returns two values the row and the attributes of that row for the entire data frame. To understand the return values better, let us review the following code:



```
for i, j in df.iterrows():
    print(i)
    print(j)
    print (type(i))
    print (type(j))
    break;
```

```
0
method            Radial Velocity
number                          1
orbital_period              269.3
mass                          7.1
distance                     77.4
year                         2006
Name: 0, dtype: object
<class 'int'>
<class 'pandas.core.series.Series'>
```

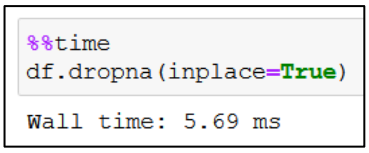
- In the above code, only one iteration return values are printed for the data frame and then there is a break. The value for i is 0 which is of integer type and the value of j is the different columns for this i which in turn is a Pandas Series object.
- Now a compact code to drop the rows can be written where any attribute / column value is missing.



```
%%time
for i, j in df.iterrows():
    if pd.isnull(j).any():
        df.drop(i, axis = 0, inplace=True)
```

```
Wall time: 403 ms
```

- Notice the usage of function ***any ()*** along with ***isnull ();*** that means any null value for the jth Series. Also note that it is over 50% more time efficient from the previous method.
- Pandas DataFrame supports even a direct, faster and more efficient function ***dropna ()*** that is optimized for the purpose.



```
%%time
df.dropna(inplace=True)
```

```
Wall time: 5.69 ms
```

- Observe that the last method is the most compact one and also 70 to 150 times faster than the previous methods achieving the same result. For the large datasets it would be very time efficient.

Exercise:

Download Seaborn ***penguins.csv*** dataset through Pandas dataframe and attempt the following:

a) Find out its rows, columns and descriptive statistics.
b) Attempt different methods of dropping the rows of missing attribute values.