Importing Datasets

Table of Contents

Data Acquisition : Introduction

There are various formats for a dataset, *.csv*, *.json*, *.xlsx* etc. The dataset can be stored in different places: our local machine or sometimes online. In this section, we will learn how to load a dataset into our Jupyter Notebook. The *pandas* library is a useful tool that enables us to read various datasets into a *DataFrame*. Jupyter Notebook platforms have a built-in *pandas* library, so all we need to do is import *pandas* without installing.

```
Ex: import pandas as pd
```

Reading Data

We use *pandas.read_csv()* function to read the *csv* file. In the bracket, we put the file path along with a quotation mark. That way *pandas* will read the file into a *DataFrame* from that address. The file path can be either an URL or your local file address.

If the data does not include headers, we can add an argument *header* = *None* inside the *read_csv()* method so that *pandas* will not automatically set the first row as a header. We can also assign the dataset to any variable you create. In our case, the Titanic dataset already contains header. So, we won't use the argument for header.

```
Ex: import pandas as pd
    path = "filepath/file.csv"
    df = pd.read_csv(path)
```

If the file is online, we can provide the URL inside the double quote.

Basic Insights of Dataset : dtypes

After reading data into *pandas DataFrame*, it is time for us to explore the dataset. There are several ways to obtain essential insights of the data to help us better understand our dataset.

Data has a variety of types. The main types stored in a *pandas DataFrame a*re *object*, *float*, *int*, *bool* and *datetime64*. In order to better learn about each attribute, it is always good for us to know the data type of each column. In pandas:

Ex: print(df.dtypes)

```
>>> PassengerId      int64
    Survived         int64
    Pclass           int64
    Name            object
    Sex             object
    Age            float64
    SibSp            int64
    Parch            int64
    Ticket          object
    Fare           float64
    Cabin           object
    Embarked        object
    dtype: object
```

This returns a series with the data type of each column. it is clear to see that the data type of "PassengerID" is *int64*, "Cabin" is *object*, and "Fare" is *float64*, etc.

Showing Data : head()

After reading the dataset, we can use the *dataframe.head(n)* method to check the top n rows of the *DataFrame* where n is an integer. Contrary to *dataframe.head(n)*, *dataframe.tail(n)* will show us the bottom n rows of the *DataFrame*. By default the value of n is 5.

```
Ex: print("The first 5 rows of the dataframe")
    df.head()
    >>>
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Basic Insights of Dataset : describe()

If we would like to get a statistical summary of each column like count, column mean value, column standard deviation, etc, we use the *describe()* method:

Ex: print(df.describe())

>>>

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

This method will provide various summary statistics, excluding *NaN* (Not a Number) values. This shows the statistical summary of all numeric-typed (*int*, *float*) columns.
For example, the attribute "Fare" has 891 counts, the mean value of this column is 0, the standard deviation is 49.69.. and so on.

Basic Insights of Dataset : index, column

We can check the index and name of columns using the following functions:

```
Ex: print(df.index)
    >>> RangeIndex(start=0, stop=891, step=1)

    print(df.columns)
    >>> Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
        'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'], dtype='object')
```

Basic Insights of Dataset : sort_index, sort_values

We can sort the dataframe by an axis, either 0 or 1

Ex: print(df.sort_index(axis=1, ascending=False))

>>>

	Ticket	Survived	SibSp	Sex	Pclass	PassengerId	Parch	Name	Fare	Embarked	Cabin	Age
0	A/5 21171	0	1	male	3	1	0	Braund, Mr. Owen Harris	7.2500	S	NaN	22.0
1	PC 17599	1	1	female	1	2	0	Cumings, Mrs. John Bradley (Florence Briggs Th...	71.2833	C	C85	38.0
2	STON/O2. 3101282	1	0	female	3	3	0	Heikkinen, Miss. Laina	7.9250	S	NaN	26.0
3	113803	1	1	female	1	4	0	Futrelle, Mrs. Jacques Heath (Lily May Peel)	53.1000	S	C123	35.0
4	373450	0	0	male	3	5	0	Allen, Mr. William Henry	8.0500	S	NaN	35.0

We can sort them too by a specific value of a column like the following example:

Ex: print(df.sort_values(by='Age')**)**

>>>

	PassengerId	Survived	Pclass	Name	Sex	Age
803	804	1	3	Thomas, Master. Assad Alexander	male	0.42
755	756	1	2	Hamalainen, Master. Viljo	male	0.67
644	645	1	3	Baclini, Miss. Eugenie	female	0.75
469	470	1	3	Baclini, Miss. Helene Barbara	female	0.75
78	79	1	2	Caldwell, Master. Alden Gates	male	0.83

Basic Insights of Dataset : slice

Like Numpy Array we can do slice in pandas dataframe too :

Ex: print(df[0:3])

>>>

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

Basic Insights of Dataset : selection by label

We can use *loc* to filter data:

```
Ex: print(df.loc[df['Sex']=='male'])
    >>>
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S

Here, we are filtering data of all the male passangers by using the label of column named "Sex" and its value "male"

Similarly, we can add more filter options:

```
Ex: print(df.loc[(df['Sex']=='male') & (df['Embarked']=='S')])
    >>>
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
12	13	0	3	Saundercock, Mr. William Henry	male	20.0	0	0	A/5. 2151	8.0500	NaN	S

Basic Insights of Dataset : selection by position

We can select data via the position of the passed integers instead of label:

```
Ex: print(df.iloc[3])
    >>>  PassengerId                                     4
         Survived                                        1
         Pclass                                          1
         Name           Futrelle, Mrs. Jacques Heath (Lily May Peel)
         Sex                                        female
         Age                                            35
         SibSp                                           1
         Parch                                           0
         Ticket                                     113803
         Fare                                         53.1
         Cabin                                        C123
         Embarked                                        S
         Name: 3, dtype: object
```

Similarly, we can fix row and column limit:

```
Ex: print(df.iloc[3:5, 0:2])
    >>>
```

	PassengerId	Survived
3	4	1
4	5	0

Basic Insights of Dataset : Boolean Indexing

We can also do boolean indexing that is to select column's value to select data:

Ex: print(df[df["Age"] > 54])

>>>

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0	0	248706	16.0000	NaN	S
33	34	0	2	Wheadon, Mr. Edward H	male	66.0	0	0	C.A. 24579	10.5000	NaN	S
54	55	0	1	Ostby, Mr. Engelhart Cornelius	male	65.0	0	1	113509	61.9792	B30	C
94	95	0	3	Coxon, Mr. Daniel	male	59.0	0	0	364500	7.2500	NaN	S

Here, we are only selecting the data of passengers who are above 54 by age

Basic Insights of Dataset : groupby()

groupby() allows to pass functions to apply to each column which produces an aggregated result with a hierarchical index:

Ex: print(df.groupby(["Sex","Survived"]).size())

>>>

```
1 df.groupby(["Sex","Survived"]).size()
```

```
Sex     Survived
female  0            81
        1           233
male    0           468
        1           109
```

This result shows that, 81 female and 468 male passengers could not survive during the sinking of Titanic. On the other hand, 233 female and 109 male passengers survived.

Missing Values : dropna(), fillna(), isna()

dropna() can remove the missing values. First, we just make a copy and then drop the values because we don't want to modify our original data too much.

```
Ex: df1 = df.copy()
    df1.dropna(how="any")
```

If we don't want to drop, we can fill the missing values. For example, in the following example, we are filling the missing values of "Age" column with median Age:

```
Ex: df['Age'] = df['Age'].fillna(df['Age'].median())
```

To get the boolean mask where values are nan, we can use *isna() :*

```
Ex: pd.isna(df1)
    >>>
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False