Chapter 4 Type Casting and Handling NA Values

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

# Read titanic dataset
tnc = pd.read_csv("./datasets/titanic.csv")

# Print dataframe
tnc.head()
```

Out[1]:

	pclass	survived	name	gender	age	sibsp	parch	ticket	fare	cabin	em
0	1	1	Allen, Miss. Elisabeth Walton	female	29	0	0	24160	211.3375	B5	
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.55	C22 C26	
2	1	0	Allison, Miss. Helen Loraine	female	2	1	2	113781	151.55	C22 C26	
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30	1	2	113781	151.55	C22 C26	
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25	1	2	113781	151.55	C22 C26	

Datatype of a column in a Dataframe

When converting a dataset into a Dataframe, Pandas assumes the datatypes of columns based on the values present in each column.

To determine the datatypes assigned, you can utilize the following methods:

1. **Dataframe.column.dtype**: This method allows you to ascertain the datatype of a specific column.

2. **Dataframe.dtypes**: This method provides the datatypes assigned to all columns in a Dataframe.

3. **Dataframe.info()**: This method gives info about the count of non-null values, the datatypes assigned, and the total memory occupied by the Dataframe.

In [2]:
```
# Print the datatype of age
tnc.age.dtype

# dtype('O') refers to Python Object 'str'
```

Out[2]:
```
dtype('O')
```

In [3]:
```
# Extract datatypes of titanic dataframe
tnc.dtypes
```

Out[3]:
```
pclass        int64
survived      int64
name         object
gender       object
age          object
sibsp         int64
parch         int64
ticket       object
fare         object
cabin        object
embarked     object
boat         object
body         object
home.dest    object
dtype: object
```

In [4]:
```
# Extract info about non-null values, datatypes assigned and total memory occupied
tnc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   pclass     1309 non-null   int64 
 1   survived   1309 non-null   int64 
 2   name       1309 non-null   object
 3   gender     1309 non-null   object
 4   age        1309 non-null   object
 5   sibsp      1309 non-null   int64 
 6   parch      1309 non-null   int64 
 7   ticket     1309 non-null   object
 8   fare       1309 non-null   object
 9   cabin      1309 non-null   object
 10  embarked   1309 non-null   object
 11  boat       1309 non-null   object
 12  body       1309 non-null   object
 13  home.dest  1309 non-null   object
dtypes: int64(4), object(10)
memory usage: 143.3+ KB
```

Converting datatypes (or) Type casting

We can convert the datatype of a column assigned by pandas to a different datatype of our choice by making use of the method:

Dataframe.column.astype(datatype)

Ex: df["age"].astype("float")

In [5]:
```
# Current datatype of age
tnc.age.dtype

# dtype('O') refers to Python Object 'str'
```

Out[5]:
```
dtype('O')
```

In [6]:
```
# Convert datatype of age column from object to float in dataframe
# tnc.age.astype("float")

# The above method fails as python cannot convert the str value '?' into a float
```

In [7]:
```
# Replace the value of '?' in age to None using Dataframe.replace() method, in plac
tnc.age.replace(['?'], [None], inplace=True)
```

In [8]:
```
# Convert datatype of age column from object to float in dataframe
tnc.age.astype("float")

# None values are converted to NaN
# Note that it returns converted values of age but doesn't update the age column
```

Out[8]:
```
0       29.0000
1        0.9167
2        2.0000
3       30.0000
4       25.0000
          ...
1304    14.5000
1305        NaN
1306    26.5000
1307    27.0000
1308    29.0000
Name: age, Length: 1309, dtype: float64
```

In [9]:
```
# Current datatype of age
tnc.age.dtype

# The datatype of age is still Object
```

Out[9]:
```
dtype('O')
```

In [10]:
```
# Set datatype of age as float using type casting
tnc.age = tnc.age.astype("float")

# Current datatype of age
tnc.age.dtype

# Now the datatype of age is converted to float64.
```

Out[10]:
```
dtype('float64')
```

In [11]:
```
# Due to type conversion we can perform EDA
tnc.describe()
```

Out[11]:

	pclass	survived	age	sibsp	parch
count	1309.000000	1309.000000	1046.000000	1309.000000	1309.000000
mean	2.294882	0.381971	29.881135	0.498854	0.385027
std	0.837836	0.486055	14.413500	1.041658	0.865560
min	1.000000	0.000000	0.166700	0.000000	0.000000
25%	2.000000	0.000000	21.000000	0.000000	0.000000
50%	3.000000	0.000000	28.000000	0.000000	0.000000
75%	3.000000	1.000000	39.000000	1.000000	0.000000
max	3.000000	1.000000	80.000000	8.000000	9.000000

Handling missing values

We can handle the missing values in a Dataframe using the following methods:

1. **Dataframe.column.isna()**: Returns a boolean Series indicating True for NaN (missing) values and False otherwise. This can be used for filtering out records with missing values in the specified column.

2. **Dataframe.isna()**: Returns a Dataframe indicating True for NaN (missing) values and False otherwise.

3. **Dataframe.column.notna()**: Returns a boolean Series indicating True for non-NaN values and False for NaN values. This can be used for filtering out records that **do not** have missing values in the specified column.

4. **Dataframe.notna()**: Returns a Dataframe indicating True for non-NaN values and False for NaN values.

5. **Dataframe.column.dropna()**: Drops all the records (rows) that contain NaN values in the specified column.

6. **Dataframe.column.fillna()**: Fills all the NaN values in the specified column with the values passed to the method.

Note that all the methods have inplace set to False.

These methods provide flexibility in how missing values are handled in a Dataframe, allowing users to either filter out, drop, or replace missing values based on their requirements.

In [12]:
```
# Read games dataset
stats = pd.read_csv("./datasets/game_stats.csv")

# Print dataframe
stats
```

Out[12]:

	name	league	points	assists	rebounds
0	bob	nba	22.0	5.0	10.0
1	jessie	NaN	10.0	NaN	2.0
2	stu	euroleague	NaN	NaN	NaN
3	jackson	aba	9.0	NaN	2.0
4	timothee	NaN	8.0	NaN	NaN
5	steph	nba	49.0	8.0	10.0
6	NaN	NaN	NaN	NaN	NaN

Filtering missing values

In [13]:
```
# Generate a dataframe indicating the presence of NaN values in stats dataframe
stats.isna()
```

Out[13]:

	name	league	points	assists	rebounds
0	False	False	False	False	False
1	False	True	False	True	False
2	False	False	True	True	True
3	False	False	False	True	False
4	False	True	False	True	True
5	False	False	False	False	False
6	True	True	True	True	True

In [14]:

```
# Generate a dataframe indicating the presence of non-NaN values in stats dataframe
stats.notna()
```

Out[14]:

	name	league	points	assists	rebounds
0	True	True	True	True	True
1	True	False	True	False	True
2	True	True	False	False	False
3	True	True	True	False	True
4	True	False	True	False	False
5	True	True	True	True	True
6	False	False	False	False	False

In [15]:

```
# Filter out records that have NaN values in league column of stats dataframe
nameNaN = stats.league.isna()

stats[nameNaN]
```

Out[15]:

	name	league	points	assists	rebounds
1	jessie	NaN	10.0	NaN	2.0
4	timothee	NaN	8.0	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN

In [16]:

```
# Filter out records that do not have NaN values in assists column of stats datafra
nameNaN = stats.assists.notna()

stats[nameNaN]
```

Out[16]:

	name	league	points	assists	rebounds
0	bob	nba	22.0	5.0	10.0
5	steph	nba	49.0	8.0	10.0

Dropping missing values

In [17]:

```
# Drop the records that have rebounds as NaN values
stats.rebounds.dropna()

# Returns series of rebounds values that are not NaN
```

Out[17]:

```
0    10.0
1     2.0
3     2.0
5    10.0
Name: rebounds, dtype: float64
```

In [18]:

```
# Drop the records that have atleast one NaN value
stats.dropna()

# Returns a dataframe that contain records having all columns filled or no NaN
```

Out[18]:

	name	league	points	assists	rebounds
0	bob	nba	22.0	5.0	10.0
5	steph	nba	49.0	8.0	10.0

In [19]:

```
# Drop the records that have all values as NaN
stats.dropna(how = "all")

# Returns a dataframe that contain records having atleast one column filled
```

Out[19]:

	name	league	points	assists	rebounds
0	bob	nba	22.0	5.0	10.0
1	jessie	NaN	10.0	NaN	2.0
2	stu	euroleague	NaN	NaN	NaN
3	jackson	aba	9.0	NaN	2.0
4	timothee	NaN	8.0	NaN	NaN
5	steph	nba	49.0	8.0	10.0

Drop the records that have certain columns as NaN

To drop records that have certain columns filled with NaN we can make use of the **subset argument in dropna() method.**

Ex: df.dropna(subset=["age", "country"]), drops all the records that have age or country as NaN.

In [20]:
```
# Drop the records that have league or points as NaN
stats.dropna(subset=["league", "points"])

# Returns a dataframe that contain records whose league and points are filled
```

Out[20]:

	name	league	points	assists	rebounds
0	bob	nba	22.0	5.0	10.0
3	jackson	aba	9.0	NaN	2.0
5	steph	nba	49.0	8.0	10.0

Filling missing values

In [21]:
```
# Fill all the missing values in the dataframe with 0
stats.fillna(0)

# Returns a dataframe with all the missing values replaced with 0
```

Out[21]:

	name	league	points	assists	rebounds
0	bob	nba	22.0	5.0	10.0
1	jessie	0	10.0	0.0	2.0
2	stu	euroleague	0.0	0.0	0.0
3	jackson	aba	9.0	0.0	2.0
4	timothee	0	8.0	0.0	0.0
5	steph	nba	49.0	8.0	10.0
6	0	0	0.0	0.0	0.0

In [22]:
```
# Fill all the missing values in league column with "isl"
stats.league.fillna("isl")

# Returns a series of league values with NaN values replaced with "isl"
```

Out[22]:
```
0           nba
1           isl
2    euroleague
3           aba
4           isl
5           nba
6           isl
Name: league, dtype: object
```

Filling certain missing values

To fill certain missing values in a Dataframe we need to pass a dictionary of columns to be filled along with their values.

In [23]:
```
# Fill the missing values of name with "Unknown" and assists with 0
stats.fillna({"name":"Unknown", "assists": 0})

# Returns a dataframe with missing names filled with "Unknown" and missing assists
```

Out[23]:

	name	league	points	assists	rebounds
0	bob	nba	22.0	5.0	10.0
1	jessie	NaN	10.0	0.0	2.0
2	stu	euroleague	NaN	0.0	NaN
3	jackson	aba	9.0	0.0	2.0
4	timothee	NaN	8.0	0.0	NaN
5	steph	nba	49.0	8.0	10.0
6	Unknown	NaN	NaN	0.0	NaN

Filling missing values with same record-different column values

To fill missing values of a Dataframe with values of some other column in the same record we can pass the column as argument to fillna() method.

In [24]:
```
# Read sales dataset
sales = pd.read_csv("./datasets/sales.csv")

# Print dataframe
sales
```

Out[24]:

	rating	shipping_zip	billing_zip
0	5.0	NaN	81220.0
1	4.5	94931.0	94931.0
2	NaN	92625.0	92625.0
3	4.5	10003.0	10003.0
4	4.0	NaN	92660.0
5	NaN	NaN	NaN
6	NaN	60007.0	60007.0

In [25]:

```
# Replace any missing shipping_zip value with corresponding billing_zip value, in p
sales["shipping_zip"].fillna(sales["billing_zip"], inplace=True)

# Print dataframe
sales

# Records that had NaN as shipping_zip values are replaced with corresponding billi
```

Out[25]:

	rating	shipping_zip	billing_zip
0	5.0	81220.0	81220.0
1	4.5	94931.0	94931.0
2	NaN	92625.0	92625.0
3	4.5	10003.0	10003.0
4	4.0	92660.0	92660.0
5	NaN	NaN	NaN
6	NaN	60007.0	60007.0