Working on Data with Pandas

Analysing Data

- **df.head(num)** and **df.tail(num)** return the first and last rows of the dataframe. By default it returns 5 rows, we can give num to have the required number of rows.
- **df.info()** method allows us to learn the shape of object types of our data. The information contains the below:

  1. **RangeIndex:** Number of rows
  2. **Data columns:** Number of columns
  3. **column labels:**, Name of each column
  4. **column data types:** could be object, int64, int32 etc.
  5. **Non-Null Count:** the number of cells in each column (non-null values).
  6. **memory usage:**, Total memory usage
- **df.describe()** method gives us summary statistics for all numerical columns seperately (**8 points summary**) in our DataFrame which are:

  1. count
  2. mean
  3. standard deviation
  4. minimum and maximum values
  5. value at 25%, 50%(median) and 75%th position in the particular column
  - The default setting of "describe" skips variables of type object. We can apply the method "describe" on the variables of type 'object' as follows: **df.describe(include=['object'])**

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
titanic = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/r
#print("\ndatadescframe from read file:\n", titanic)
```

In [2]:
```
titanic.head(3)
```

Out[2]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	c
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	

In [3]:
```
titanic.tail(3)
```

Out[3]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	c
888	0	3	female	NaN	1	2	23.45	S	Third	woman	False	
889	1	1	male	26.0	0	0	30.00	C	First	man	True	
890	0	3	male	32.0	0	0	7.75	Q	Third	man	True	

In [4]:
```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   survived     891 non-null    int64  
 1   pclass       891 non-null    int64  
 2   sex          891 non-null    object 
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64  
 5   parch        891 non-null    int64  
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object 
 8   class        891 non-null    object 
 9   who          891 non-null    object 
 10  adult_male   891 non-null    bool   
 11  deck         203 non-null    object 
 12  embark_town  889 non-null    object 
 13  alive        891 non-null    object 
 14  alone        891 non-null    bool   
dtypes: bool(2), float64(2), int64(4), object(7)
memory usage: 92.4+ KB
```

In [5]:
```
titanic.describe()
```

Out[5]:

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Cleaning Data

Data cleaning means fixing bad data in your data set. Bad data could be:

1. Empty cells
2. Data in wrong format
3. Wrong data
4. Duplicates

Empty Cells

- Can potentially give you a wrong result when you analyze data.
- We have different ways to clean the data with empty cells. They are:
    1. **Remove rows:** Generally we have large datasets so it is okay to remove a few rows with no data.
        - **df.dropna()** or **df[col].dropna** is used to drop/remove rows with empty cells.
        - By default, the dropna() method returns a new DataFrame, and will not change the original.
        - Use **inplace=True** to make changes in same dataframe
    2. **Replace Empty Values:** helps in updating empty cells without removing the rows and having major impact on data
        - The **df.fillna(value, inplace = true/false)** or **df[col].fillna(value, inplace = true/false)** method allows us to replace empty cells with a value
        - We can replace empty cells with mean, median or mode depending on the type of data

In [6]:
```
titanic_dropped = titanic.dropna()
#titanic.dropna(inplace = True) #Use this is you want to change titanic DF dire
titanic_dropped #titanic contains 891 rows and titanic_dropped contains 182 rov
```

Out[6]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
6	0	1	male	54.0	0	0	51.8625	S	First	man	True
10	1	3	female	4.0	1	1	16.7000	S	Third	child	False
11	1	1	female	58.0	0	0	26.5500	S	First	woman	False
...	...	...	...	...	...	...	...	...	...	...	...
871	1	1	female	47.0	1	1	52.5542	S	First	woman	False
872	0	1	male	33.0	0	0	5.0000	S	First	man	True
879	1	1	female	56.0	0	1	83.1583	C	First	woman	False
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True

182 rows × 15 columns

In [7]:
```python
x = titanic["sex"].mode() #Filling empty rows of column sex with mode of the co
titanic_filled = titanic["sex"].fillna(x)
titanic_filled.head()
```

Out[7]:
```
0      male
1    female
2    female
3    female
4      male
Name: sex, dtype: object
```

Data in Wrong Format

- Cells with data of wrong format can make it difficult, or even impossible to analyze data and can also give wrong results
- We have two options to fix it:
  1. Remove the rows
  2. Convert all cells in the columns into the same format
     - Eg: Changing rows in date column to standard format. Pandas has a to_datetime() method
     - df['Date_col'] = pd.to_datetime(df['Date_col'])

Wrong Data

- Can be checked if there are any outliers or unusual data
- This can be fixed as follows:
  1. Replacing values
     - Replace the values with extremes/mean/median/mode
     - For small data sets you might be able to replace the wrong data one by one, but not for big data sets.
     - To replace wrong data for larger data sets you can create some rules, e.g. set some boundaries for legal values, and replace any values that are outside of the boundaries.
  2. Removing rows
     - remove the rows if the value is an outlier or visibly wrong using **df.drop(row_index)**

In [8]:
```python
#Change all fare values to 120 if it is more than 120. Considering 120 as the l
for x in titanic.index:
  if titanic.loc[x, "fare"] > 120:
     titanic.loc[x, "fare"] = 120
```

In [9]:
```python
# Remove all rows with fare >120
for x in titanic.index:
  if titanic.loc[x, "fare"] > 120:
    titanic.drop(x, inplace = True)
```

Removing Duplicates

- To discover duplicates, we can use the **df.duplicated()** method. It returns a Boolean values for each row and returns true if it finds duplicate row
- **df.drop_duplicates()** is used to drop/remove duplicate rows. Use **inplace = True** if changes need to be made in original DF.

In [10]:
```
print(titanic.duplicated())
```

```
0      False
1      False
2      False
3      False
4      False
       ...
886     True
887    False
888    False
889    False
890    False
Length: 891, dtype: bool
```

In [11]:
```
titanic_dupdrop = titanic.drop_duplicates()
titanic_dupdrop
```

Out[11]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True
...	...	...	...	...	...	...	...	...	...	...	...
885	0	3	female	39.0	0	5	29.1250	Q	Third	woman	False
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True

784 rows × 15 columns

Correlations and Finding Relationships between Variables

- **df.corr()** is used to find correlation between each column
  - It ignores non-numeric columns
  - The value of correlation ranges from -1 to 1

- If value is near 1 --> strong direct correlation, if it is near -1 --> strong indirect correlation.
- Whereas, values near 0 means no or weak correlation

In [12]:
```
titanic.corr()
```

```
C:\Users\srbhk\AppData\Local\Temp\ipykernel_2428\2964377706.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  titanic.corr()
```

Out[12]:

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
survived	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.312741	-0.557080	-0.203367
pclass	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.687877	0.094035	0.135207
age	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.133674	0.280328	0.198270
sibsp	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.247533	-0.253586	-0.584471
parch	0.081629	0.018443	-0.189119	0.414838	1.000000	0.261243	-0.349943	-0.583398
fare	0.312741	-0.687877	0.133674	0.247533	0.261243	1.000000	-0.249450	-0.388331
adult_male	-0.557080	0.094035	0.280328	-0.253586	-0.349943	-0.249450	1.000000	0.404744
alone	-0.203367	0.135207	0.198270	-0.584471	-0.583398	-0.388331	0.404744	1.000000

Plotting in Pandas

- Pandas uses the **df.plot(kind='some_type', x= 'some_column', y='some_colum', color='somecolor')** method to create diagrams
  - kind= scatter, hist, line or bar
- We can use Pyplot, a submodule of the Matplotlib library to visualize the diagram on the screen. Function used it **plt.show()**
- **plt.title("Title")** is used to give title to the plot

In [13]:
```
titanic.plot(kind = 'scatter', x = 'fare', y = 'pclass')
plt.title("Scatter plot for fare and pclass")
plt.show()
```