Learning_Pandas_Part_9_MoreInPandas-2

June 20, 2021

0.0.1 Prepared by Abhishek Kumar

0.0.2 https://www.linkedin.com/in/abhishekkumar-0311/

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sys
```

[2]:
```
# To get multiple outputs in the same cell

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

%matplotlib inline
```

1 Data Preparation

[3]:
```
# Setup : DataFrame creation

salary = [['1','Abhishek Kumar','AIML', 'Machine Learning Engineer','M', 'Y',␣
 ↪'04051990', 1121000],
          ['2','Arjun Kumar','DM', 'Tech Lead','M', 'Y', '09031992', 109000],
          ['3','Vivek Raj','DM', 'Devops Engineer','M', 'N', np.NaN , 827000],
          ['4','Mika Singh','DM', 'Data Analyst','F', 'Y', '15101991',  np.NaN],
          ['5','Anusha Yenduri','AIML', 'Data Scientist','F', 'Y', '01011989', ␣
 ↪921000],
          ['6','Ritesh Srivastava','AIML', 'Data Engineer','M', 'Y', np.NaN,␣
 ↪785000]]

columns_name=['Emp_Id','Emp_Name','Department','Role','Gender', 'WFH Status',␣
 ↪'DOB', 'Salary']

emp_df = pd.DataFrame(salary,columns=columns_name)
emp_df
```

[3]:

	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	Y	04051990	1121000.0
1	2	Arjun Kumar	DM	Tech Lead	M	Y	09031992	109000.0
2	3	Vivek Raj	DM	Devops Engineer	M	N	NaN	827000.0
3	4	Mika Singh	DM	Data Analyst	F	Y	15101991	NaN
4	5	Anusha Yenduri	AIML	Data Scientist	F	Y	01011989	921000.0
5	6	Ritesh Srivastava	AIML	Data Engineer	M	Y	NaN	785000.0

[4]:
```
import numpy as np
import pandas as pd
sample = {
'col_a':['Houston,TX', 'Dallas,TX', 'Chicago,IL', 'Phoenix,AZ',      'San␣
 ↪Diego,CA'],
'col_b':['62K-70K', '62K-70K', '69K-76K', '62K-72K', '71K-78K' ],
'col_c':['A','B','A','a','c'],
'col_d':['  1x', ' 1y', '2x  ', '1x', '1y  ']
}
df_sample = pd.DataFrame(sample)
df_sample
```

[4]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	Dallas,TX	62K-70K	B	1y
2	Chicago,IL	69K-76K	A	2x
3	Phoenix,AZ	62K-72K	a	1x
4	San Diego,CA	71K-78K	c	1y

2 Functions discussed in this Notebook - Part 2

Function	Description	Part
apply()	Apply a function along an axis of the DataFrame.	1
applymap()	Apply a function to a Dataframe elementwise.	1
map()	map() is used to substitute each value in a Series with another value.	1

Function	Description	Part
transform()	Call func on self producing a DataFrame with transformed values.	1

Function	Description	Part
df.assign()	Assign new columns to a DataFrame.	2
pipe()	Apply func(self, *args, **kwargs).	2
df.update()	Modify in place using non-NA values from another DataFrame.	2
df.take	Return the elements in the given positional indices along an axis.	2
df.truncate	Truncate a Series or DataFrame before and after some index value.	2

Function	Description	Part
df.items	Iterates over the DataFrame columns, returning a tuple with the column name and the content as a Series.	3
df.iteritems	Iterates over the DataFrame columns, returning a tuple with the column name and the content as a Series.	3
df.iterrows	Iterate over DataFrame rows as (index, Series) pairs.	3
df.itertuples	Iterate over DataFrame rows as namedtuples.	3

[ ]:

[5]:
```
# Data prep

df = df_sample.copy();
df
```

[5]:
```
          col_a    col_b col_c col_d
0    Houston,TX  62K-70K     A    1x
1     Dallas,TX  62K-70K     B    1y
2    Chicago,IL  69K-76K     A  2x
3    Phoenix,AZ  62K-72K     a    1x
4  San Diego,CA  71K-78K     c  1y
```

[6]:
```
df.values
```

[6]:
```
array([['Houston,TX', '62K-70K', 'A', '  1x'],
       ['Dallas,TX', '62K-70K', 'B', ' 1y'],
       ['Chicago,IL', '69K-76K', 'A', '2x  '],
       ['Phoenix,AZ', '62K-72K', 'a', '1x'],
       ['San Diego,CA', '71K-78K', 'c', '1y  ']], dtype=object)
```

[7]:
```
df.col_a.tolist()
```

[7]:
```
['Houston,TX', 'Dallas,TX', 'Chicago,IL', 'Phoenix,AZ', 'San Diego,CA']
```

[ ]:
```

```

2.1 Assign()

- DataFrame.assign(**kwargs)[source]
  - Assign new columns to a DataFrame.
- Parameters
  - **kwargsdict of {str: callable or Series}
    - The column names are keywords. If the values are callable, they are computed on the DataFrame and assigned to the new columns. The callable must not change input DataFrame (though pandas doesn't check it). If the values are not callable, (e.g. a Series, scalar, or array), they are simply assigned.
- Returns : DataFrame
  - A **new DataFrame** with the **new columns in addition to all the existing columns**.
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.assign.html
- https://towardsdatascience.com/using-pandas-method-chaining-to-improve-code-readability-d8517c5626ac

[8]:
```
# emp_df.dtypes
```

[9]:
```
empdf = emp_df.copy()
empdf
```

[9]:
```
   Emp_Id           Emp_Name Department                        Role Gender  \
0       1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1       2        Arjun Kumar         DM                   Tech Lead      M
2       3          Vivek Raj         DM           Devops Engineer      M
3       4         Mika Singh         DM              Data Analyst      F
```

```
4      5     Anusha Yenduri       AIML             Data Scientist      F
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary
0          Y  04051990  1121000.0
1          Y  09031992   109000.0
2          N       NaN   827000.0
3          Y  15101991        NaN
4          Y  01011989   921000.0
5          Y       NaN   785000.0
```

[10]:
```
empdf = empdf.assign(NewSal1 = lambda x : x.Salary*1.1)
empdf
```

[10]:
```
  Emp_Id           Emp_Name Department                       Role Gender  \
0      1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1      2        Arjun Kumar         DM                  Tech Lead      M
2      3          Vivek Raj         DM           Devops Engineer      M
3      4         Mika Singh         DM              Data Analyst      F
4      5     Anusha Yenduri       AIML             Data Scientist      F
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary    NewSal1
0          Y  04051990  1121000.0  1233100.0
1          Y  09031992   109000.0   119900.0
2          N       NaN   827000.0   909700.0
3          Y  15101991        NaN        NaN
4          Y  01011989   921000.0  1013100.0
5          Y       NaN   785000.0   863500.0
```

[11]:
```
empdf = empdf.assign(NewSal2 = empdf.Salary*1.1)
empdf
```

[11]:
```
  Emp_Id           Emp_Name Department                       Role Gender  \
0      1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1      2        Arjun Kumar         DM                  Tech Lead      M
2      3          Vivek Raj         DM           Devops Engineer      M
3      4         Mika Singh         DM              Data Analyst      F
4      5     Anusha Yenduri       AIML             Data Scientist      F
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary    NewSal1    NewSal2
0          Y  04051990  1121000.0  1233100.0  1233100.0
1          Y  09031992   109000.0   119900.0   119900.0
2          N       NaN   827000.0   909700.0   909700.0
3          Y  15101991        NaN        NaN        NaN
4          Y  01011989   921000.0  1013100.0  1013100.0
```

```
5          Y       NaN   785000.0   863500.0   863500.0
```

2.1.1 We can create multiple columns within the same assign where one of the columns depends on another one defined within the same assign:

[12]:
```
# empdf.assign(NewSal3 = empdf.NewSal1 + empdf.NewSal2,
#              NewSal4 = empdf.NewSal3 + empdf.NewSal3)
```

[ ]:
```
# ![image.png](attachment:image.png)
```

2.1.2 But it does not work like this. We need to create the SECOND NEW Column using LAMBDA function.

- i.e, the dependent columns should be created with lambda expressions

[13]:
```
empdf.assign(NewSal3 = lambda x : x.NewSal1 + x.NewSal2,
             NewSal4 = lambda x : x.NewSal3 + x.NewSal3)
```

[13]:
```
  Emp_Id           Emp_Name Department                      Role Gender  \
0      1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1      2        Arjun Kumar         DM                  Tech Lead      M
2      3          Vivek Raj         DM           Devops Engineer       M
3      4         Mika Singh         DM              Data Analyst       F
4      5     Anusha Yenduri       AIML             Data Scientist      F
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary    NewSal1    NewSal2    NewSal3    NewSal4
0          Y  04051990  1121000.0  1233100.0  1233100.0  2466200.0  4932400.0
1          Y  09031992   109000.0   119900.0   119900.0   239800.0   479600.0
2          N       NaN   827000.0   909700.0   909700.0  1819400.0  3638800.0
3          Y  15101991        NaN        NaN        NaN        NaN        NaN
4          Y  01011989   921000.0  1013100.0  1013100.0  2026200.0  4052400.0
5          Y       NaN   785000.0   863500.0   863500.0  1727000.0  3454000.0
```

[14]:
```
empdf.assign(NewSal33 = empdf.NewSal1 + empdf.NewSal2,
             NewSal44 = lambda x : x.NewSal33 + x.NewSal33)
```

[14]:
```
  Emp_Id           Emp_Name Department                      Role Gender  \
0      1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1      2        Arjun Kumar         DM                  Tech Lead      M
2      3          Vivek Raj         DM           Devops Engineer       M
3      4         Mika Singh         DM              Data Analyst       F
4      5     Anusha Yenduri       AIML             Data Scientist      F
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary    NewSal1    NewSal2   NewSal33   NewSal44
0          Y  04051990  1121000.0  1233100.0  1233100.0  2466200.0  4932400.0
```

```
1          Y  09031992   109000.0   119900.0   119900.0   239800.0   479600.0
2          N       NaN   827000.0   909700.0   909700.0  1819400.0  3638800.0
3          Y  15101991        NaN        NaN        NaN        NaN        NaN
4          Y  01011989   921000.0  1013100.0  1013100.0  2026200.0  4052400.0
5          Y       NaN   785000.0   863500.0   863500.0  1727000.0  3454000.0
```

[15]:
```
empdf.assign(NewSal33 = empdf.NewSal1 + empdf.NewSal2,
             NewSal44 = lambda x : x.NewSal33 + x.NewSal33, NewSal55 = lambda x␣
↪: x.NewSal44 + x.NewSal44)
```

[15]:
```
  Emp_Id           Emp_Name Department                        Role Gender  \
0      1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1      2        Arjun Kumar         DM                   Tech Lead      M
2      3          Vivek Raj         DM            Devops Engineer      M
3      4         Mika Singh         DM               Data Analyst      F
4      5     Anusha Yenduri       AIML             Data Scientist      F
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary    NewSal1    NewSal2   NewSal33   NewSal44  \
0          Y  04051990  1121000.0  1233100.0  1233100.0  2466200.0  4932400.0
1          Y  09031992   109000.0   119900.0   119900.0   239800.0   479600.0
2          N       NaN   827000.0   909700.0   909700.0  1819400.0  3638800.0
3          Y  15101991        NaN        NaN        NaN        NaN        NaN
4          Y  01011989   921000.0  1013100.0  1013100.0  2026200.0  4052400.0
5          Y       NaN   785000.0   863500.0   863500.0  1727000.0  3454000.0

    NewSal55
0  9864800.0
1   959200.0
2  7277600.0
3        NaN
4  8104800.0
5  6908000.0
```

2.2 PIPE

- DataFrame.pipe(func, *args, **kwargs)[source]
  - Apply func(self, *args, **kwargs).
- Parameters :
  - func function
    - Function to apply to the Series/DataFrame. args, and kwargs are passed into func. Alternatively a (callable, data_keyword) tuple where data_keyword is a string indicating the keyword of callable that expects the Series/DataFrame.
  - args iterable, optional
    - Positional arguments passed into func.
  - kwargs mapping, optional

    - A dictionary of keyword arguments passed into func.

- Returns : objectthe return type of func.
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pipe.html
- https://towardsdatascience.com/using-pandas-method-chaining-to-improve-code-readability-d8517c5626ac
- https://towardsdatascience.com/using-pandas-pipe-function-to-improve-code-readability-96d66abfaf8

[16]:
```
# I have taken up the titanic dataset for this case

def load_data():
    return pd.read_csv('./titanic.csv')
df = load_data()
df.head()
```

[16]:

	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

2.2.1 Tasks

Suppose we have been asked to work on the following tasks

1. Split Name into first name and second name
2. For Sex, substitute value male with M and female with F
3. Replace the missing Age with some form of imputation
4. Convert ages to groups of age ranges: 12, Teen (18), Adult (60), and Older (>60).

- Let's go ahead and use `pipe()` to accomplish them step by step,

2.2.2 1. Split Name into first name and second name

[17]:
```
def split_name(x_df):
    def split_name_series(string):
        firstName, secondName=string.split(', ')
#         print(type(firstName))
        c =  pd.Series(
            (firstName, secondName),
            index='firstName secondName'.split()
        )
#         print(c)
#         print(type(c))
        return c
    # Select the Name column and apply a function
    res=x_df['Name'].apply(split_name_series)
#     print(type(res))
#     print(res)
    x_df[res.columns]=res
#     print(x_df)
    return x_df
```

[18]:
```
res=(
    load_data()
    .pipe(split_name)
)
res.head()
```

[18]:

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

```
                                secondName
0                            Mr. Owen Harris
1  Mrs. John Bradley (Florence Briggs Thayer)
2                                Miss. Laina
3         Mrs. Jacques Heath (Lily May Peel)
4                          Mr. William Henry
```

2.2.3 2. For Sex, substitute value male with M and female with F

[19]:
```
def substitute_sex(x_df):
    mapping={'male':'M','female':'F'}
    x_df['Sex']=df['Sex'].map(mapping)
    return x_df
```

[20]:
```
res=(
    load_data()
    .pipe(split_name)
    .pipe(substitute_sex)
)
res.head()
```

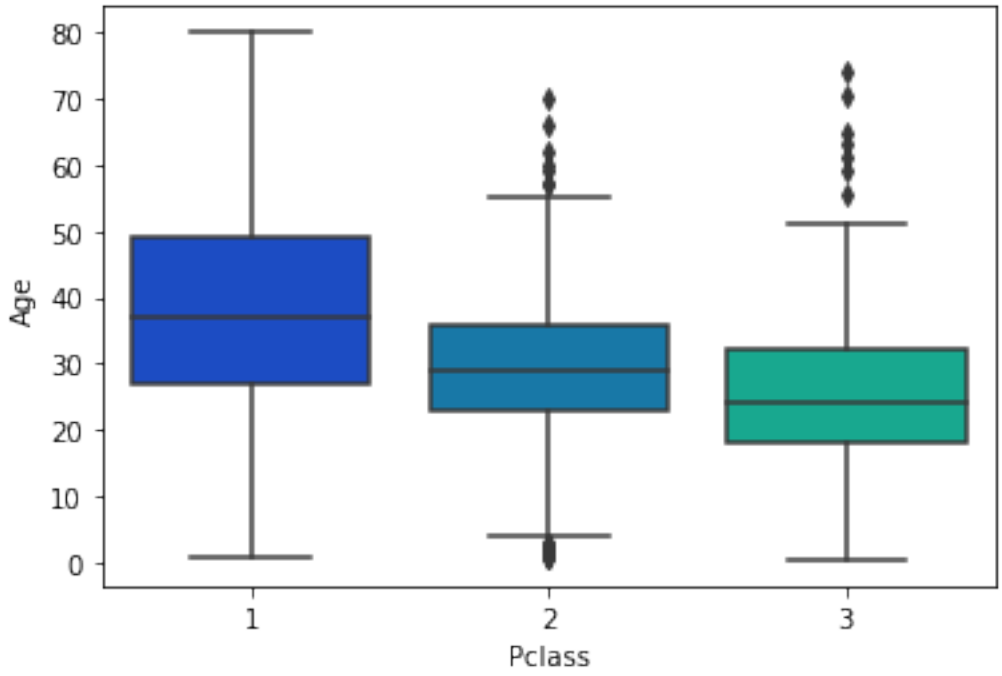
[20]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name Sex   Age  SibSp  Parch  \
0                            Braund, Mr. Owen Harris   M  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th…   F  38.0      1      0
2                             Heikkinen, Miss. Laina   F  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)   F  35.0      1      0
4                           Allen, Mr. William Henry   M  35.0      0      0

             Ticket     Fare Cabin Embarked  firstName  \
0         A/5 21171   7.2500   NaN        S     Braund
1          PC 17599  71.2833   C85        C    Cumings
2  STON/O2. 3101282   7.9250   NaN        S  Heikkinen
3            113803  53.1000  C123        S   Futrelle
4            373450   8.0500   NaN        S      Allen

                                secondName
0                            Mr. Owen Harris
1  Mrs. John Bradley (Florence Briggs Thayer)
2                                Miss. Laina
```

```
3          Mrs. Jacques Heath (Lily May Peel)
4                           Mr. William Henry
```

2.2.4 3. Replace the missing Age with some form of imputation

[21]:
```
sns.boxplot(x='Pclass',
            y='Age',
            data=df,
            palette='winter')
```

[21]:
```
<AxesSubplot:xlabel='Pclass', ylabel='Age'>
```



[22]:
```
pclass_age_map = {
  1: 37,
  2: 29,
  3: 24,
}
def replace_age_na(x_df, fill_map):
    cond=x_df['Age'].isna()
    res=x_df.loc[cond,'Pclass'].map(fill_map)
    x_df.loc[cond,'Age']=res
    return x_df
```

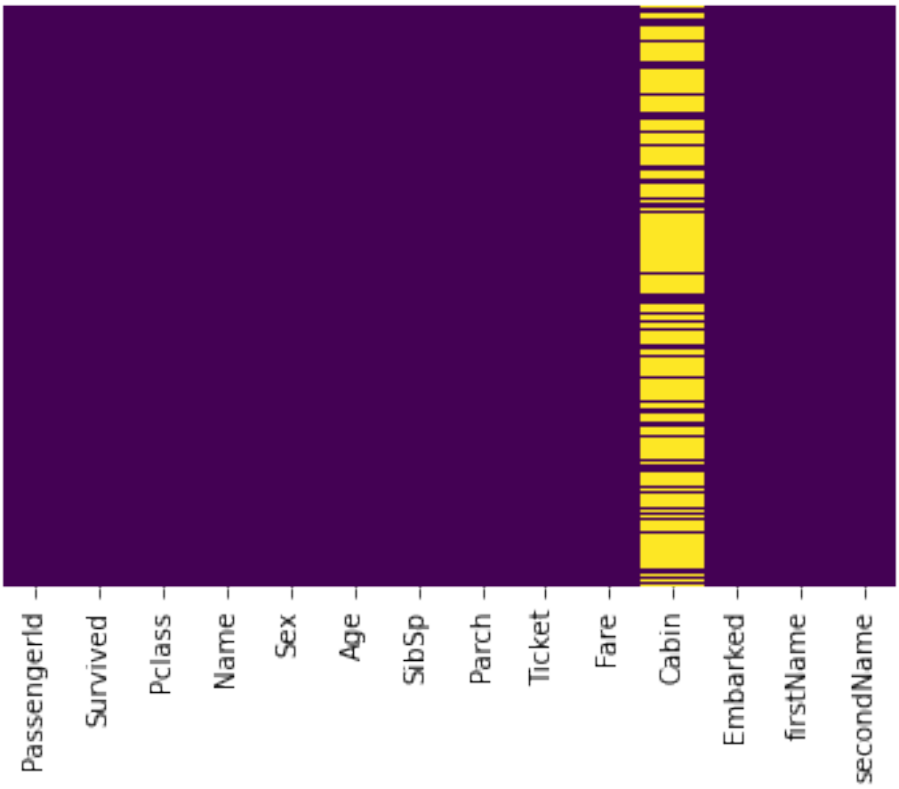
[23]:
```
res=(
    load_data()
    .pipe(split_name)
    .pipe(substitute_sex)
    .pipe(replace_age_na, pclass_age_map)
)
res.head()
```

[23]:

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

[24]:
```
sns.heatmap(res.isnull(),
            yticklabels=False,
            cbar=False,
            cmap='viridis')
```

[24]:
```
<AxesSubplot:>
```



2.2.5 4. Convert ages to groups of age ranges: 12, Teen (18), Adult (60), and Older (>60)

[25]:
```
def create_age_group(x_df):
    bins=[0, 13, 19, 61, sys.maxsize]
    labels=['<12', 'Teen', 'Adult', 'Older']
    ageGroup=pd.cut(x_df['Age'], bins=bins, labels=labels)
    x_df['ageGroup']=ageGroup
    return x_df
```

[26]:
```
res=(
    load_data()
    .pipe(split_name)
    .pipe(substitute_sex)
    .pipe(replace_age_na, pclass_age_map)
    .pipe(create_age_group)
)
res.head()
```

[26]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
```

```
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3
```

```
                                                 Name Sex   Age  SibSp  Parch  \
0                             Braund, Mr. Owen Harris   M  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th…   F  38.0      1      0
2                              Heikkinen, Miss. Laina   F  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)   F  35.0      1      0
4                            Allen, Mr. William Henry   M  35.0      0      0

            Ticket     Fare Cabin Embarked  firstName  \
0        A/5 21171   7.2500   NaN        S     Braund
1         PC 17599  71.2833   C85        C    Cumings
2  STON/O2. 3101282   7.9250   NaN        S  Heikkinen
3           113803  53.1000  C123        S   Futrelle
4           373450   8.0500   NaN        S      Allen

                                secondName ageGroup
0                            Mr. Owen Harris    Adult
1  Mrs. John Bradley (Florence Briggs Thayer)    Adult
2                               Miss. Laina    Adult
3         Mrs. Jacques Heath (Lily May Peel)    Adult
4                           Mr. William Henry    Adult
```

2.2.6 More examples :

2.2.7 https://www.kdnuggets.com/2021/01/cleaner-data-analysis-pandas-pipes.html

2.3 Update()

- DataFrame.update(other, join='left', overwrite=True, filter_func=None, errors='ignore')[source]
  - Modify **in place using non-NA values** from another DataFrame.
  - **Aligns on indices.** There is no return value.
- Parameters :
  - other DataFrame, or object coercible into a DataFrame
    - Should have **at least one matching index/column label with the original DataFrame**. If a Series is passed, its name attribute must be set, and that will be used as the column name to align with the original DataFrame.
  - join{'left'}, default 'left'
    - **Only left join is implemented, keeping the index and columns of the original object.**
  - overwrite bool, default True
    - How to handle non-NA values for overlapping keys:
    - **True: overwrite original DataFrame's values with values from other.**
    - **False: only update values that are NA in the original DataFrame.**

- filter_func callable(1d-array) -> bool 1d-array, optional
  - Can choose to replace values other than NA. Return True for values that should be updated.
- errors {'raise', 'ignore'}, default 'ignore'
  - If 'raise', will raise a ValueError if the DataFrame and other both contain non-NA data in the same place.
  - Changed in version 0.24.0: Changed from raise_conflict=False|True to errors='ignore'|'raise'.

- Returns : None method directly changes calling object
- Raises : ValueError
  - When errors='raise' and there's overlapping non-NA data.
  - When errors is not either 'ignore' or 'raise'
  - NotImplementedError : If join != 'left'

[29]:
```
df = pd.DataFrame({'A': ['a', 'b', 'c'],
                   'B': ['x', 'y', 'z']})
df
new_df = pd.DataFrame({'B': ['d', 'e']}, index=[1, 2])
new_df
df.update(new_df)
df
```

[29]:

	A	B
0	a	x
1	b	y
2	c	z

[29]:

	B
1	d
2	e

[29]:

	A	B
0	a	x
1	b	d
2	c	e

2.3.1 overwrite (bool) : default True : Defines How to handle non-NA values for overlapping keys :

- **False: only update values that are NA in the original DataFrame.**

[30]:
```
df = pd.DataFrame({'A': ['a', 'b', 'c'],
                   'B': ['x', 'y', 'z']})
df
new_df = pd.DataFrame({'B': ['d', 'e']}, index=[1, 2])
new_df
df.update(new_df, overwrite=False)
df
```

```
# Here the values in the overlpping indexes of original dataframe IS NOT updated.
# With the option overwrite=false, it only updates when the original dataframe has NaNs in overlapping indexes.
```

[30]:
```
   A  B
0  a  x
1  b  y
2  c  z
```

[30]:
```
   B
1  d
2  e
```

[30]:
```
   A  B
0  a  x
1  b  y
2  c  z
```

[31]:
```
df = pd.DataFrame({'A': ['a', 'b', 'c'],
                   'B': ['x', np.NaN, 'z']})
df
new_df = pd.DataFrame({'B': ['d', 'e']}, index=[1, 2])
new_df
df.update(new_df, overwrite=False)
df

# Here the values in the overlpping indexes of original dataframe IS Updated.
# With the option overwrite=false, it only updates when the original dataframe has NaNs in overlapping indexes.
```

[31]:
```
   A    B
0  a    x
1  b  NaN
2  c    z
```

[31]:
```
   B
1  d
2  e
```

[31]:
```
   A  B
0  a  x
1  b  d
2  c  z
```

2.3.2 If other contains NaNs the corresponding values are not updated in the original dataframe.

[33]:
```
df = pd.DataFrame({'A': ['a', 'b', 'c'],
                   'B': ['x','y', 'z']})
df
new_df = pd.DataFrame({'B': ['d', np.NaN]}, index=[1, 2])
new_df
df.update(new_df)
df
```

[33]:
```
   A  B
0  a  x
1  b  y
2  c  z
```

[33]:
```
     B
1    d
2  NaN
```

[33]:
```
   A  B
0  a  x
1  b  d
2  c  z
```

2.4 take

- `DataFrame.take(indices, axis=0, is_copy=None, ****kwargs)`[source]
  - Return the elements in the given positional indices along an axis.
  - This means that we are not indexing according to actual values in the index attribute of the object. We are indexing according to the actual position of the element in the object.
- Parameters :
  - indices array-like : An array of ints indicating which positions to take.
  - axis {0 or 'index', 1 or 'columns', None}, default 0
    - The axis on which to select elements. 0 means that we are selecting rows, 1 means that we are selecting columns.
- **https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.take.html**

[34]:
```
df = pd.DataFrame([('falcon', 'bird', 389.0),
                   ('parrot', 'bird', 24.0),
                   ('lion', 'mammal', 80.5),
                   ('monkey', 'mammal', np.nan)],
                  columns=['name', 'class', 'max_speed'],
                  index=[0, 2, 3, 1])
df
```

[34]:
```
     name   class  max_speed
0  falcon    bird      389.0
2  parrot    bird       24.0
3    lion  mammal       80.5
1  monkey  mammal        NaN
```

[35]:
```
# the actual indices selected (0 and 1) do not correspond to our selected␣
↪indices 0 and 3

df.take([0, 3])
```

[35]:
```
     name   class  max_speed
0  falcon    bird      389.0
1  monkey  mammal        NaN
```

Take elements at indices 1 and 2 along the axis 1 (column selection).

[36]:
```
df.take([1, 2], axis=1)
```

[36]:
```
    class  max_speed
0    bird      389.0
2    bird       24.0
3  mammal       80.5
1  mammal        NaN
```

Take elements using negative integers for positive indices, starting from the end of the object, just like with Python lists.

[37]:
```
df.take([-1, -2])
```

[37]:
```
     name   class  max_speed
1  monkey  mammal        NaN
3    lion  mammal       80.5
```

2.4.1 Showing the behaviour of `.iloc()` and `.loc()`

[43]:
```
df
```

[43]:
```
     name   class  max_speed
0  falcon    bird      389.0
2  parrot    bird       24.0
3    lion  mammal       80.5
1  monkey  mammal        NaN
```

[38]:
```
df.iloc[[0,3]]
```

[38]:
```
     name   class  max_speed
0  falcon    bird      389.0
1  monkey  mammal        NaN
```

[41]:
```
df.loc[[0,3]]
```

[41]:
```
     name   class  max_speed
0  falcon    bird      389.0
3    lion  mammal       80.5
```

[45]:
```
df.iloc[df.index[[0,3]]]
df.index
df.index[[0,3]]
```

[45]:
```
     name class  max_speed
0  falcon  bird      389.0
2  parrot  bird       24.0
```

[45]:
```
Int64Index([0, 2, 3, 1], dtype='int64')
```

[45]:
```
Int64Index([0, 1], dtype='int64')
```

2.5 `truncate()`

`DataFrame.truncate(before=None, after=None, axis=None, copy=True)`[source] - Truncate a Series or DataFrame **before and after some index value.**

`- This is a useful shorthand for boolean indexing based on index values above or below certain`

- Parameters :
  - before - date, str, int : Truncate all rows before this index value.
  - after - date, str, int : Truncate all rows after this index value.
  - axis - {0 or 'index', 1 or 'columns'}, optional : Axis to truncate. Truncates the index (rows) by default.
  - copy - bool, default is True, Return a copy of the truncated section.
- Returns : type of caller : The truncated Series or DataFrame.

[46]:
```
df2 = df.copy()
df2
```

[46]:
```
     name   class  max_speed
0  falcon    bird      389.0
2  parrot    bird       24.0
3    lion  mammal       80.5
1  monkey  mammal        NaN
```

[48]:
```
# df2.truncate(before=2, after =3)
```

```
# Truncate requires a sorted index
```

[ ]:
```
# ![image.png](attachment:image.png)
```

[57]:
```
# So we will sort the dataframe based on index

# By default, axis= 0  / 'rows'
df2.sort_index(axis= 'rows', inplace=True)
df2
```

[57]:

	name	class	max_speed
0	falcon	bird	389.0
1	monkey	mammal	NaN
2	parrot	bird	24.0
3	lion	mammal	80.5

[58]:
```
df2.truncate(before=2, after =3)
```

[58]:

	name	class	max_speed
2	parrot	bird	24.0
3	lion	mammal	80.5

[ ]:
```

```