day23_23_feb_pandas_library_basic-

1 23th feb pandas library basic

read, manupulate, perform feature engineering, EDA operations with pandas

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

1.1 it always work with the structured data like sql

1.2 structured data mean a tabular form of data

2 - reading .csv file

[ ]:
```
pd.read_csv("services.csv")
```

2.0.1 when it reads a file/ tabular data/ dataset

2.0.2 pandas always consider bydefault 1 row as a column schema/name/header

2.0.3 if first row is not header than pass header = None than it'll give indexes [0:] by own

by default give row index from 0 to n and if 'header=None' then only give column index from 0 to n ([0:])

[4]:
```
df= pd.read_csv('services.csv') #stored in variable df
type(df)
```

[4]:
```
pandas.core.frame.DataFrame
```

its type is dataframe

pandas only has 2 data type dataframe and series

mutiple row - multiple column is dataframe

[6]:
```
df.columns #return all the column name written in header, 'to check column name'
```

[6]:
```
Index(['id', 'location_id', 'program_id', 'accepted_payments',
       'alternate_name', 'application_process', 'audience', 'description',
```

```
       'eligibility', 'email', 'fees', 'funding_sources',
       'interpretation_services', 'keywords', 'languages', 'name',
       'required_documents', 'service_areas', 'status', 'wait_time', 'website',
       'taxonomy_ids'],
      dtype='object')
```

[7]:
```
list(df.columns) #converted header column into a list
```

[7]:
```
['id',
 'location_id',
 'program_id',
 'accepted_payments',
 'alternate_name',
 'application_process',
 'audience',
 'description',
 'eligibility',
 'email',
 'fees',
 'funding_sources',
 'interpretation_services',
 'keywords',
 'languages',
 'name',
 'required_documents',
 'service_areas',
 'status',
 'wait_time',
 'website',
 'taxonomy_ids']
```

[ ]:
```
df.head() #if want to select some records from dataframe
# it'll print first 5 records/rows from dataframe, from[0:4]
```

[ ]:
```
df.head(3) # now it'll print first 3 records from top/beginning
```

[ ]:
```
df.tail() # will print by-default last 5 records from bottom/end
# number is pass then it'll print that much record from last
```

[ ]:
```
df.tail(3)
```

[15]:
```
df.dtypes # return datatypes of column inside dataframe
#if want to know which column has null value, string, int, object
```

[15]:
```
id                          int64
location_id                 int64
program_id                float64
```

```
accepted_payments           object
alternate_name              object
application_process         object
audience                    object
description                 object
eligibility                 object
email                       object
fees                        object
funding_sources             object
interpretation_services     object
keywords                    object
languages                   object
name                        object
required_documents          object
service_areas               object
status                      object
wait_time                   object
website                     object
taxonomy_ids                object
dtype: object
```

[16]:
```
df.columns #if want to select couple of column then pass column name
```

[16]:
```
Index(['id', 'location_id', 'program_id', 'accepted_payments',
       'alternate_name', 'application_process', 'audience', 'description',
       'eligibility', 'email', 'fees', 'funding_sources',
       'interpretation_services', 'keywords', 'languages', 'name',
       'required_documents', 'service_areas', 'status', 'wait_time', 'website',
       'taxonomy_ids'],
      dtype='object')
```

[17]:
```
df['location_id'] # selected a single column from dataframe
```

[17]:
```
0      1
1      2
2      3
3      4
4      5
5      6
6      7
7      8
8      9
9     10
10    11
11    12
12    13
13    14
```

```
14    15
15    16
16    17
17    18
18    19
19    20
20    21
21    22
22    22
Name: location_id, dtype: int64
```

[28]:
```
type(df['location_id']) # checking datatype of single column, # passing as a list
```

[28]:
```
pandas.core.series.Series
```

- its type is *series* not dataframe
- dataframe is collection of rows and column, it's table
- but series type is equalent to list, it is also printing row-indexes but list doesn't show indexes,
- a series datatype also show indexes
- a single column selected from dataframe is a *series*
- property of list and series is little bit different

[29]:
```
list(df['location_id']) # see list doesn't show index
```

[29]:
```
[1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
 14,
 15,
 16,
 17,
 18,
 19,
 20,
```

```
 21,
 22,
 22]
```

now passing df[‘location_id’] with extra square bracket as inside a list

[30]:
```
df[['location_id']]
```

[30]:
```
    location_id
0             1
1             2
2             3
3             4
4             5
5             6
6             7
7             8
8             9
9            10
10           11
11           12
12           13
13           14
14           15
15           16
16           17
17           18
18           19
19           20
20           21
21           22
22           22
```

[31]:
```
type(df[['location_id']]) # it's now a dataframe
```

[31]:
```
pandas.core.frame.DataFrame
```

when passed single column (a string) it’s a series datatype and can perform series based operations when passed as list it’ll be dataframe datatype and can perform dataframe based operations

2.0.4 selecting multiple column

[32]:
```
df.columns
```

[32]:
```
Index(['id', 'location_id', 'program_id', 'accepted_payments',
       'alternate_name', 'application_process', 'audience', 'description',
```

```
       'eligibility', 'email', 'fees', 'funding_sources',
       'interpretation_services', 'keywords', 'languages', 'name',
       'required_documents', 'service_areas', 'status', 'wait_time', 'website',
       'taxonomy_ids'],
      dtype='object')
```

[ ]:
```
df['keywords','status'] #cannot pass two column as a string, only single column can be passed
# have to pass it in a list because it's 2 columns/multiple column and its datatype will be a dataframe

## ERROR
```

[34]:
```
df[['keywords','status']]
```

[34]:

	keywords	status
0	ADULT PROTECTION AND CARE SERVICES, Meal Sites…	active
1	EMPLOYMENT/TRAINING SERVICES, Job Development,…	active
2	Geriatric Counseling, Older Adults, Gay, Lesbi…	active
3	INDIVIDUAL AND FAMILY DEVELOPMENT SERVICES, Gr…	active
4	COMMUNITY SERVICES, Speakers, Automobile Loans	active
5	ADULT PROTECTION AND CARE SERVICES, In-Home Su…	active
6	ADULT PROTECTION AND CARE SERVICES, Adult Day …	active
7	ADULT PROTECTION AND CARE SERVICES, Meal Sites…	active
8	EDUCATION SERVICES, Library, Libraries, Public…	active
9	EDUCATION SERVICES, Library, Libraries, Public…	active
10	EDUCATION SERVICES, Library, Libraries, Public…	active
11	EDUCATION SERVICES, Adult, Alternative, Litera…	active
12	EDUCATION SERVICES, Library, Libraries, Public…	active
13	COMMUNITY SERVICES, Interpretation/Translation…	active
14	ALCOHOLISM SERVICES, Residential Care, DRUG AB…	active
15	COMMODITY SERVICES, Clothing/Personal Items, C…	active
16	COMMODITY SERVICES, Clothing/Personal Items, C…	active
17	HEALTH SERVICES, Outpatient Care, Community Cl…	active
18	HEALTH SERVICES, Outpatient Care, Community Cl…	active
19	NaN	defunct
20	NaN	inactive
21	Salud, Medicina	active
22	Ruby on Rails/Postgres/Redis, testing, wic	active

so for passing multiple column, we have to pass it in a list by-default it'll be dataframe datatype

[ ]:
```
df[['interpretation_services', 'keywords','languages','name']]
# order of selection doesn't matter like sql, it'll print whichever columns name we will pass
```

3

4

5 reading excel dataset/file

[ ]:
```
pd.read_excel("LUSID Excel - Setting up your market data.xlsx")
```

5.0.1 pandas.read_excel()

- if it has single sheet then it's ok to use, but excel files can have multiple sheets inside its file

excel has multiple sheet inside its file pandas.read_excel(sheetname:), can give sheetname as filename then it'll fetch that sheet from excel file

[40]:
```
df1 = pd.read_excel("LUSID Excel - Setting up your market data.xlsx") # storing␣
↪in a variable df1
```

[ ]:
```
df1.head # printing first 5 records from dataframe
# head() forget to put paranthesis '()' in head function
```

[ ]:
```
df1.head() # fetched first 5 records,
#head() return first 5 records by-default
```

lets access first six records from df1 dataframe

[ ]:
```
df1.head(6)
```

lets access last three records from df1 dataframe

[ ]:
```
df1.tail(3)
```

6 datatypes of all columns

[53]:
```
type(df1.columns())
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[53], line 1
----> 1 type(df1.columns())

TypeError: 'Index' object is not callable
```

[54]:
```
df1.dtypes # this will return datatypes of all columns

#object is may be string
```

[54]:
```
Unnamed: 0    float64
Unnamed: 1    float64
Unnamed: 2    float64
Unnamed: 3     object
Unnamed: 4     object
Unnamed: 5     object
Unnamed: 6    float64
Unnamed: 7     object
Unnamed: 8     object
Unnamed: 9     object
dtype: object
```

[55]:
```
df1.columns
```

[55]:
```
Index(['Unnamed: 0', 'Unnamed: 1', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4',
       'Unnamed: 5', 'Unnamed: 6', 'Unnamed: 7', 'Unnamed: 8', 'Unnamed: 9'],
      dtype='object')
```

[56]:
```
df1['Unnamed: 0'] # fetching data from single column
```

[56]:
```
0    NaN
1    NaN
2    NaN
3    NaN
4    NaN
5    NaN
6    NaN
7    NaN
8    NaN
9    NaN
10   NaN
11   NaN
12   NaN
13   NaN
14   NaN
15   NaN
16   NaN
17   NaN
18   NaN
19   NaN
20   NaN
21   NaN
22   NaN
23   NaN
24   NaN
25   NaN
26   NaN
```

```
27   NaN
Name: Unnamed: 0, dtype: float64
```

[ ]:
```
# getting data from multiple columns
df1[['Unnamed: 6', 'Unnamed: 7']] # have to inside list to fetch multiple␣
 ↪column data
```

7 read data from link

7.0.1 which is in comma seperated file .csv

7.0.2 online data looks untidy

[ ]:
```
pd.read_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/
 ↪titanic.csv")
```

8 it was in csv format that why read with .read_csv() from link

and its showing 891 rows and 12 columns

[61]:
```
df2 = pd.read_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/
 ↪master/titanic.csv")
# storing in a variable df2
```

[63]:
```
type(df2) #data type is dataframe
```

[63]:
```
pandas.core.frame.DataFrame
```

[65]:
```
df2.dtypes # columns datatype
```

[65]:
```
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
```

[ ]:
```
df2.head(3) # top 3 record
```

[ ]:
```
df2.tail(3) # fetching botton 3 data
```

[68]:
```
# getting all the column names
df2.columns
```

[68]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

fetching/selecting only fare column data

it will be series datatype and have to pass as a string

[ ]:
```
df2['fare'] # it's case sensitive

## ERROR
```

[73]:
```
df2['Fare'] # have to write as it is written in column header/ 'column name'
```

[73]:
```
0       7.2500
1      71.2833
2       7.9250
3      53.1000
4       8.0500
        …
886    13.0000
887    30.0000
888    23.4500
889    30.0000
890     7.7500
Name: Fare, Length: 891, dtype: float64
```

[ ]:
```
# fetching multiple column data, have to pass as list
df2[['Survived', 'Pclass', 'Name']]
```

9 saving data in my local system

10 data is present in my jupyter note book

when will close it we can't see result, manupulation and other operations that we have done

so if want to check data with out opening jupyter notebook then, save this dataframe in local system

[79]:
```
df2.to_csv("titanic_data.csv")
#this will save online-fetched dataset on local system
#can export to any type of format csv, excel etc
```

11 after executing, it's downloaded/saved in our current working directory, because we didn't gave any path

and its file format is csv, whatever file format we want it in, can be saved

after manupulation we can store it in our local system

11.0.1 but it is also saving row indexes which was auto generated, if we don't want it in our exported file we can use ,index=False

index during file creation and saving is True, if we write index=False it won't get printed on to exported file

[80]:
```
df2.to_csv("titanic_data.csv", index=False) # we won't get extra column of auto␣
↪generated index on saved file
```

12 loading dataset from url

pandas only read tabular data so to fetch a tabular data - if we want other type of data we can do it with selenium, BeautifulSoup etc for scraping

in this NBA url reading data which is present in Tabular format only

[ ]:
```
pd.read_html("https://www.basketball-reference.com/leagues/NBA_2015_totals.
↪html")

### ERROR
```

13 because we reading dataset from URL and we haven't import lxml module to read online tabular file

[83]:
```
import lxml
#ModuleNotFoundError because we don't have installed this package in our
#program, now have to pip install package
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[83], line 1
----> 1 import lxml
      2 #ModuleNotFoundError because we don't have installed this package in ou
      3 #program, now have to pip install package

ModuleNotFoundError: No module named 'lxml'
```

[84]:
```
pip install lxml
```

```
Collecting lxml
  Downloading lxml-4.9.2-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_24_x86_64.whl (7.1 MB)
                              7.1/7.1 MB
63.6 MB/s eta 0:00:0000:0100:01
Installing collected packages: lxml
Successfully installed lxml-4.9.2
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
import lxml
import pandas as pd

pd.read_html("https://www.basketball-reference.com/leagues/NBA_2015_totals.
 ↪html")
```

[2]:
```
[      Rk           Player Pos Age   Tm   G  GS    MP   FG  FGA  …   FT%  ORB
\
 0      1      Quincy Acy  PF  24  NYK  68  22  1287  152  331  …  .784   79
 1      2    Jordan Adams  SG  20  MEM  30   0   248   35   86  …  .609    9
 2      3    Steven Adams   C  21  OKC  70  67  1771  217  399  …  .502  199
 3      4     Jeff Adrien  PF  28  MIN  17   0   215   19   44  …  .579   23
 4      5   Arron Afflalo  SG  29  TOT  78  72  2502  375  884  …  .843   27
 ..   …              …  ..  ..  …  ..  ..   …  …  …  …   …  …
 670  490  Thaddeus Young  PF  26  TOT  76  68  2434  451  968  …  .655  127
 671  490  Thaddeus Young  PF  26  MIN  48  48  1605  289  641  …  .682   75
 672  490  Thaddeus Young  PF  26  BRK  28  20   829  162  327  …  .606   52
 673  491     Cody Zeller   C  22  CHO  62  45  1487  172  373  …  .774   97
 674  492    Tyler Zeller   C  25  BOS  82  59  1731  340  619  …  .823  146

      DRB  TRB  AST  STL BLK  TOV   PF   PTS
 0    222  301   68   27  22   60  147   398
 1     19   28   16   16   7   14   24    94
 2    324  523   66   38  86   99  222   537
 3     54   77   15    4   9    9   30    60
 4    220  247  129   41   7  116  167  1035
 ..   …  …  …  …  ..  …  …   …
 670  284  411  173  124  25  117  171  1071
 671  170  245  135   86  17   75  115   685
 672  114  166   38   38   8   42   56   386
 673  265  362  100   34  49   62  156   472
 674  319  465  113   18  52   76  205   833

 [675 rows x 30 columns]]
```

14 now able to read tabular data from url

but it'll not return a dataframe, it'll return data as a list

[3]:
```
url_data = pd.read_html("https://www.basketball-reference.com/leagues/
 ↪NBA_2015_totals.html")
```

[4]:
```
type(url_data)
```

[4]:
```
list
```

15 this tabular data/datatype is returning a list

#because in a url link there's a possiblity that there could be zero table #or thousands of tables, what it does is # pandas.read_html(), it'll read all the table / scrape all the table > 1 data is 1 table - and try to keep all this table in a single list - and one by one we can extract it

we can check how many table are fetched from url/ or in our list just

- use len(url_data)

[6]:
```
len(url_data) # 1 means this url_data had only 1 single table
```

[6]:
```
1
```

16 fetching that single table/dataframe which is in our list in [0] index

[ ]:
```
url_data[0] # now it's properly showing tabular data
# this is our dataframe
```

[8]:
```
df4 = url_data[0] # storing first tabular data fetched from url link
```

[9]:
```
type(df4)
```

[9]:
```
pandas.core.frame.DataFrame
```

[ ]:
```
df4.head() # that '...' is because
#in a console it's not able to project entire/all data
# have to save it and then we can check in a one single file
```

[ ]:
```
df4.tail(4)
```

[15]:
```
df4.dtypes # all columns datatype is object='string'
#object datatype is STRING dataype
```

[15]:
```
Rk        object
Player    object
Pos       object
Age       object
```

```
Tm        object
G         object
GS        object
MP        object
FG        object
FGA       object
FG%       object
3P        object
3PA       object
3P%       object
2P        object
2PA       object
2P%       object
eFG%      object
FT        object
FTA       object
FT%       object
ORB       object
DRB       object
TRB       object
AST       object
STL       object
BLK       object
TOV       object
PF        object
PTS       object
dtype: object
```

[18]:
```
df4.columns # fetching all 30 column names
```

[18]:
```
Index(['Rk', 'Player', 'Pos', 'Age', 'Tm', 'G', 'GS', 'MP', 'FG', 'FGA', 'FG%',
       '3P', '3PA', '3P%', '2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA', 'FT%',
       'ORB', 'DRB', 'TRB', 'AST', 'STL', 'BLK', 'TOV', 'PF', 'PTS'],
      dtype='object')
```

[20]:
```
df4['Rk'] # selected one single column as a series
```

[20]:
```
0        1
1        2
2        3
3        4
4        5
      ...
670    490
671    490
672    490
673    491
```

```
674    492
Name: Rk, Length: 675, dtype: object
```

[ ]:
```
df4[['Pos', "Age", 'Tm', "G"]] # selecting multiple columns
# have to pass in a list
#whichever column name we have written is selected from dataframe
```

[ ]:
```
# now saving in our local system to check data without opening our console
```

17 saving data we fetched from website, because we don't want to fetch

it again and again from website

[24]:
```
df4.to_csv("players.csv", index=False)
#saving it in a .csv format file and removing autogenerated row index in␣
 ↪dataframe
```

[25]:
```
#now saved file is showing in our current file directory
```

[ ]:
```

```

day24_24_feb_pandas_library_advance_1-

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

[2]:
```
df = pd.read_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/
↪master/titanic.csv")
```

[ ]:
```
df.head()
```

[ ]:
```
df.tail()
```

[5]:
```
df.columns
```

[5]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

[6]:
```
df.dtypes
```

[6]:
```
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
```

[ ]:
```
df.head()
```

[8]:
```
df.dtypes #columns ,shape
```

[8]:
```
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
```

[10]:
```
df.describe() # analytical result of numerical value, integer and float datatype, all are not some time useful
# like mean/average of passengerId
```

[10]:
```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

1 describe on string/object dataset

[12]:
```
df.dtypes #checking column datatypes which are string
```

[12]:
```
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
```

[ ]:
```
df[['Name', 'Sex','Ticket','Cabin','Embarked']] #putting string-value column␣
 ↪name manually
```

[18]:
```
df.dtypes.index #when checking datatypes of column,
#column name will be index and printed datatype is value
```

[18]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

[21]:
```
df.dtypes.values # it'll print only datatype of column in dataframe without␣
 ↪indexes
#where 'O' is object/string
```

[21]:
```
array([dtype('int64'), dtype('int64'), dtype('int64'), dtype('O'),
       dtype('O'), dtype('float64'), dtype('int64'), dtype('int64'),
       dtype('O'), dtype('float64'), dtype('O'), dtype('O')], dtype=object)
```

[28]:
```
print(df.dtypes == 'Object')
print(df.dtypes == 'object') #return True and False according to condition
#if conditional value is present True else False
```

```
PassengerId    False
Survived       False
Pclass         False
Name           False
Sex            False
Age            False
SibSp          False
Parch          False
Ticket         False
Fare           False
Cabin          False
Embarked       False
dtype: bool
PassengerId    False
Survived       False
Pclass         False
Name            True
```

```
Sex             True
Age            False
SibSp          False
Parch          False
Ticket          True
Fare           False
Cabin           True
Embarked        True
dtype: bool
```

[29]:
```
(df.dtypes=='object').index
```

[29]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

[ ]:
```
df[df.dtypes=='object']

## ERROR
```

[ ]:
```
df[df.dtypes=='object'].index

##ERROR
```

[34]:
```
df.dtypes[df.dtypes=='object'] #now its printing only string/object-value␣
 ↪columns
## df.columns can't be used because it does;t
```

[34]:
```
Name        object
Sex         object
Ticket      object
Cabin       object
Embarked    object
dtype: object
```

[35]:
```
df.dtypes[df.dtypes=='object'].index
```

[35]:
```
Index(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], dtype='object')
```

[ ]:
```
df[df.dtypes[df.dtypes=='object'].index]
```

[40]:
```
df[df.dtypes[df.dtypes=='object'].index].describe()
#top is first record and frequency will how many times first record data has␣
 ↪been repeated in dataframe
```

[40]:
```
                         Name   Sex  Ticket    Cabin Embarked
count                     891   891     891      204      889
```

```
unique                       891     2     681      147        3
top     Braund, Mr. Owen Harris  male  347082  B96 B98        S
freq                           1   577       7        4      644
```

2 for filtering only integer datatype columns

[ ]:
```
df[df.dtypes[df.dtypes=='int64'].index] #in pandas 'int64' is datatype for␣
↪interger value column
```

[44]:
```
#df[df.dtypes[df.dtypes=='int64'].index].describe()
```

[44]:
```
       PassengerId    Survived      Pclass       SibSp       Parch
count   891.000000  891.000000  891.000000  891.000000  891.000000
mean    446.000000    0.383838    2.308642    0.523008    0.381594
std     257.353842    0.486592    0.836071    1.102743    0.806057
min       1.000000    0.000000    1.000000    0.000000    0.000000
25%     223.500000    0.000000    2.000000    0.000000    0.000000
50%     446.000000    0.000000    3.000000    0.000000    0.000000
75%     668.500000    1.000000    3.000000    1.000000    0.000000
max     891.000000    1.000000    3.000000    8.000000    6.000000
```

[ ]:
```
df[df.dtypes[df.dtypes=='float64']]

## ERROR
```

[ ]:
```
df[df.dtypes[df.dtypes=='float64'].index] #datatype for float value is float64
```

[49]:
```
df.columns
```

[49]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

[ ]:
```
df[['Survived','Pclass']] # selecting some columns
```

[54]:
```
df[['Survived','Pclass']][4:11] #selecting rows 4 to 10
```

[54]:
```
    Survived  Pclass
4          0       3
5          0       3
6          0       1
7          0       3
8          1       3
9          1       2
10         1       3
```

[55]:
```
df[['Survived','Pclass']][4:11:2] #selecting rows 4 to 10 and jumping 2-2␣
↪records between selected record
```

[55]:
```
    Survived  Pclass
4          0       3
6          0       1
8          1       3
10         1       3
```

[57]:
```
df['new_col']=0 # it'll create new column and put 0 in each rows/records
```

[ ]:
```
df.head()
```

[64]:
```
df['pass_sur'] = df['PassengerId']+df['Pclass'] # adding two int column into␣
↪new col
```

[66]:
```
pd.Categorical(df['Pclass']) # fetch no. of category present in selected column
```

[66]:
```
[3, 1, 3, 1, 3, …, 2, 1, 3, 1, 3]
Length: 891
Categories (3, int64): [1, 2, 3]
```

[69]:
```
pd.Categorical(df['Survived']) # 2 category is there 0 and 1
```

[69]:
```
[0, 1, 1, 1, 0, …, 0, 1, 0, 1, 0]
Length: 891
Categories (2, int64): [0, 1]
```

[70]:
```
pd.Categorical(df['Cabin'])
```

[70]:
```
[NaN, 'C85', NaN, 'C123', NaN, …, NaN, 'B42', NaN, 'C148', NaN]
Length: 891
Categories (147, object): ['A10', 'A14', 'A16', 'A19', …, 'F38', 'F4', 'G6',
'T']
```

[71]:
```
pd.unique(df['Cabin'])
```

[71]:
```
array([nan, 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
       'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
       'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101',
       'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4',
       'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
       'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
       'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
       'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40',
       'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44',
       'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14',
```

```
       'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
       'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
       'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48',
       'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63',
       'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30',
       'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36',
       'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42',
       'C148'], dtype=object)
```

[72]:
```
df['Cabin'].unique() # return unique values present in column
```

[72]:
```
array([nan, 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
       'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
       'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101',
       'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4',
       'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
       'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
       'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
       'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40',
       'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44',
       'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14',
       'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
       'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
       'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48',
       'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63',
       'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30',
       'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36',
       'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42',
       'C148'], dtype=object)
```

[ ]:
```

```

3 selecting where Age is greater than 18

[ ]:
```
df['Age']>18 #return boolean data True and False
```

[ ]:
```
df[df['Age']>18] # return dataframe where Age column has Age>18
```

[79]:
```
len(df) - len(df[df["Age"]>18]) # print no. of records whose Age is less than 18
```

[79]:
```
316
```

[ ]:
```
df["Fare"]<32.204208 # checking records who have paid lesser fare than␣
↪average-fare
```

[ ]:
```
df[df["Fare"] < 32.204208]
```

```
[ ]: df[df['Fare']>32.204208] # people who have paid more fare than average
```

```
[ ]: df["Fare"]==0 # who got free ticket on titanic ship
```

```
[ ]: df[df["Fare"]==0] # filtering where fare is 0
```

```
[89]: df[df["Fare"]==0]["Name"] # getting only names of people who paid 0 fare
```

```
[89]: 179                 Leonard, Mr. Lionel
      263               Harrison, Mr. William
      271         Tornquist, Mr. William Henry
      277          Parkes, Mr. Francis "Frank"
      302      Johnson, Mr. William Cahoone Jr
      413       Cunningham, Mr. Alfred Fleming
      466                 Campbell, Mr. William
      481     Frost, Mr. Anthony Wood "Archie"
      597                  Johnson, Mr. Alfred
      633        Parr, Mr. William Henry Marsh
      674           Watson, Mr. Ennis Hastings
      732                 Knight, Mr. Robert J
      806               Andrews, Mr. Thomas Jr
      815                     Fry, Mr. Richard
      822      Reuchlin, Jonkheer. John George
      Name: Name, dtype: object
```

```
[91]: len(df[df["Fare"]==0]) # how many were there with 0 fare amount/free ticket
```

```
[91]: 15
```

```
[ ]: df["Sex"]=="male"
```

```
[ ]: df[df["Sex"]=="male"] # males boarded on ship
```

```
[ ]: df[df['Sex']=="female"] # females boarded on ship
```

```
[ ]: df[df["Pclass"]==1] # people who bought class-1 tickets
```

```
[ ]: df[df["Survived"]==1] # people who survived in accident
     # 1= survived, 2=died
```

```
[ ]: df[df["Survived"]==0] # people who died
```

```
[ ]:
```

4 print females who paid more than average for ticket

[111]:
```
df[df["Sex"]=="female" and df["Fare"]>32.204208] # not and but have to use &␣
↪and put () paranthesis
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[111], line 1
----> 1 df[df["Sex"]=="female" and df["Fare"]>32.204208] # not and but have to␣
↪use & and put () paranthesis

File /opt/conda/lib/python3.10/site-packages/pandas/core/generic.py:1527, in␣
↪NDFrame.__nonzero__(self)
   1525 @final
   1526 def __nonzero__(self) -> NoReturn:
-> 1527     raise ValueError(
   1528         f"The truth value of a {type(self).__name__} is ambiguous. "
   1529         "Use a.empty, a.bool(), a.item(), a.any() or a.all()."
   1530     )

ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bool(), a.
↪item(), a.any() or a.all().
```

[ ]:
```
df["Sex"]=="female"
```

[ ]:
```
df["Fare"]> 32.204208
```

[ ]:
```
df[df["Sex"]=="female" & df["Fare"]>32.204208]
```

[ ]:
```
df[ (df["Sex"]=="female") & (df["Fare"]>32.204208)]
```

[118]:
```
df[ (df["Sex"]=="male") and (df["Fare"] > 32)]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[118], line 1
----> 1 df[ (df["Sex"]=="male") and (df["Fare"] > 32)]

File /opt/conda/lib/python3.10/site-packages/pandas/core/generic.py:1527, in␣
↪NDFrame.__nonzero__(self)
   1525 @final
   1526 def __nonzero__(self) -> NoReturn:
-> 1527     raise ValueError(
   1528         f"The truth value of a {type(self).__name__} is ambiguous. "
   1529         "Use a.empty, a.bool(), a.item(), a.any() or a.all()."
   1530     )
```

```
ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bool(), a.
↪item(), a.any() or a.all().
```

[ ]:
```
df[ (df["Sex"]=="male") & (df["Fare"] > 32)]
```

[ ]:
```
df[ (df["Sex"]=="male") | (df["Fare"] > 32)] # | is or condition either one is␣
↪True will give result
```

[ ]:
```

```

[122]:
```
df["Fare"].max()
```

[122]:
```
512.3292
```

[123]:
```
max(df["Fare"])
```

[123]:
```
512.3292
```

[128]:
```
df[df["Fare"]==max(df["Fare"])]
```

[128]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	new_col	pass_sur
258	259	1	1	Ward, Miss. Anna	female	35.0	0	0	PC 17755	512.3292	NaN	C	0	260
679	680	1	1	Cardeza, Mr. Thomas Drake Martinez	male	36.0	0	1	PC 17755	512.3292	B51 B53 B55	C	0	681
737	738	1	1	Lesurer, Mr. Gustave J	male	35.0	0	0	PC 17755	512.3292	B101	C	0	739

[ ]:
```
df["Fare"]== max(df['Fare'])
```

[ ]:
```
df[df['Fare'] == max(df["Fare"])]
```

[132]:
```
df[df['Fare'] == max(df["Fare"])],["Name"]
```

[132]:
```
(     PassengerId  Survived  Pclass                                Name  \
 258          259         1       1                   Ward, Miss. Anna
 679          680         1       1  Cardeza, Mr. Thomas Drake Martinez
 737          738         1       1              Lesurer, Mr. Gustave J
```

```
        Sex   Age  SibSp  Parch    Ticket      Fare        Cabin Embarked  \
258  female  35.0      0      0  PC 17755  512.3292          NaN        C
679    male  36.0      0      1  PC 17755  512.3292  B51 B53 B55        C
737    male  35.0      0      0  PC 17755  512.3292         B101        C

     new_col  pass_sur
258        0       260
679        0       681
737        0       739  ,
['Name'])
```

[ ]:
```
df[df[df['Fare'] == max(df["Fare"])],["Name"]]

## ERROR
```

[135]:
```
df[df['Fare'] == max(df["Fare"])]['Name'] #printing name of max fare
```

[135]:
```
258                     Ward, Miss. Anna
679    Cardeza, Mr. Thomas Drake Martinez
737                Lesurer, Mr. Gustave J
Name: Name, dtype: object
```

5

[ ]:
```
df[0:100:2] #selecting rows form 0 to 100 with even number indexes
```

[ ]:
```

```

6 loc and iloc is for selecting rows and columns

[158]:
```
df.iloc[0:2] #i for inbuilt/integer location or default indexes given by␣
 ↪pandas, value for selection,
# exclude last given index in range
```

[158]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1

   Parch     Ticket     Fare Cabin Embarked  new_col  pass_sur
0      0  A/5 21171   7.2500   NaN        S        0         4
1      0   PC 17599  71.2833   C85        C        0         3
```

[143]:
```
df.iloc[0:2]['Fare']
```

[143]:
```
0     7.2500
1    71.2833
Name: Fare, dtype: float64
```

[153]:
```
df.loc[0:2] # location function
    #loc consider named indexes where iloc consider inbuilt indexes
    # it doesn't exclude last row in given range
```

[153]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0

   Parch            Ticket     Fare Cabin Embarked  new_col  pass_sur
0      0         A/5 21171   7.2500   NaN        S        0         4
1      0          PC 17599  71.2833   C85        C        0         3
2      0  STON/O2. 3101282   7.9250   NaN        S        0         6
```

[152]:
```
df.loc[0:2]['Fare']
```

[152]:
```
0     7.2500
1    71.2833
2     7.9250
Name: Fare, dtype: float64
```

6.0.1 iloc select rows and column with internal indexes or by-default indexes given by pandas

[ ]:
```
df.iloc[0:2,['PassengerId', 'Survived', 'Pclass']]

## ERROR
```

6.0.2 loc select rows and column with named indexes or indexes given by human/table

[147]:
```
df.loc[0:2,['PassengerId', 'Survived', 'Pclass']]
```

[147]:
```
   PassengerId  Survived  Pclass
0            1         0       3
1            2         1       1
```

```
2          3          1          3
```

[1]:
```
df.iloc[0:2,0:3] #iloc exclude last index in given range
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 df.iloc[0:2,0:3] #iloc exclude last index in given range

NameError: name 'df' is not defined
```

[ ]:
```
df.iloc[0:2,[0,1,2]]
```

[ ]:
```
df.loc[0:2,0:3]

## ERROR
```

[ ]:
```

```

day25_25_feb_pandas_library_part_3_advance_2-

1

2 day25_25th_feb_pandas_library_part_4_advance_2

[14]:
```
import pandas as pd
```

[13]:
```
data = {"a":[1,2,3,4],
        "b":[4,5,6,7],
        "c":["mohan","sohan","rohan","gohan"]}
```

[14]:
```
data
```

[14]:
```
{'a': [1, 2, 3, 4],
 'b': [4, 5, 6, 7],
 'c': ['mohan', 'sohan', 'rohan', 'gohan']}
```

[15]:
```
df = pd.DataFrame(data)
```

[16]:
```
df
```

[16]:

	a	b	c
0	1	4	mohan
1	2	5	sohan
2	3	6	rohan
3	4	7	gohan

-

2.0.1 set_index() and reset_index()

[17]:
```
df.set_index('c') # inplace=True will update main dataframe, now it's just␣
↪showing operation result
```

[17]:

c	a	b
mohan	1	4
sohan	2	5

```
rohan  3  6
gohan  4  7
```

[18]:
```
df
```

[18]:
```
    a  b      c
0   1  4  mohan
1   2  5  sohan
2   3  6  rohan
3   4  7  gohan
```

[19]:
```
df.set_index('c', inplace=True) # inplace=True will change main dataframe
#set 'c' column as row indexes for dataframe
```

[20]:
```
df
```

[20]:
```
        a  b
c
mohan   1  4
sohan   2  5
rohan   3  6
gohan   4  7
```

[21]:
```
df.reset_index() # reseting  row indexes to default state
```

[21]:
```
        c  a  b
0   mohan  1  4
1   sohan  2  5
2   rohan  3  6
3   gohan  4  7
```

[22]:
```
df
```

[22]:
```
        a  b
c
mohan   1  4
sohan   2  5
rohan   3  6
gohan   4  7
```

[23]:
```
df.reset_index(inplace=True)
```

[24]:
```
df
```

[24]:
```
        c  a  b
0   mohan  1  4
1   sohan  2  5
```

```
2  rohan  3  6
3  gohan  4  7
```

-

2.0.2 passing manual row index during dataframe creation

[25]:
```
data = {"a":[1,2,3,4],
        "b":[4,5,6,7],
        "c":["mohan","sohan","rohan","gohan"]}

df1= pd.DataFrame(data, index=['a','b','c','d'])
```

[27]:
```
df1 #now changed default indexes to our own indexes
```

[27]:
```
   a  b      c
a  1  4  mohan
b  2  5  sohan
c  3  6  rohan
d  4  7  gohan
```

-

2.0.3 reindex() to change order of row/column using indexes

[28]:
```
df1.reindex(['b','c','a','d']) # have to write as it is index name, otherwise␣
 ↪it'll create Nan value rows/columns with new-given name index
#changed order of rows
```

[28]:
```
   a  b      c
b  2  5  sohan
c  3  6  rohan
a  1  4  mohan
d  4  7  gohan
```

-

2.0.4 iterrows()

[30]:
```
df1
```

[30]:
```
   a  b      c
a  1  4  mohan
b  2  5  sohan
c  3  6  rohan
d  4  7  gohan
```

[32]:
```
df1.iterrows()
```

[32]:
```
<generator object DataFrame.iterrows at 0x7fb7baff3d80>
```

[35]:
```
for i in df1.iterrows(): #it iterate row by row, one by one it iterate rows and␣
 ↪extract dataset
    print(i)
# it'll iterate row wise
```

```
('a', a       1
b       4
c    mohan
Name: a, dtype: object)
('b', a       2
b       5
c    sohan
Name: b, dtype: object)
('c', a       3
b       6
c    rohan
Name: c, dtype: object)
('d', a       4
b       7
c    gohan
Name: d, dtype: object)
```

[36]:
```
for i in df1.iteritems(): # it'll iterate column wise
    print(i)
```

```
('a', a    1
b    2
c    3
d    4
Name: a, dtype: int64)
('b', a    4
b    5
c    6
d    7
Name: b, dtype: int64)
('c', a    mohan
b    sohan
c    rohan
d    gohan
Name: c, dtype: object)
```

```
/tmp/ipykernel_6593/2038616457.py:1: FutureWarning: iteritems is deprecated and
will be removed in a future version. Use .items instead.
  for i in df1.iteritems(): # it'll iterate column wise
```

2.0.5 summation on all columns one by one, 1 column summation then next column

[37]:
```
def test(x):
    return x.sum()

df1.apply(test, axis=0) #axis = 0 means add rows of a column one by one
```

[37]:
```
a                       10
b                       22
c    mohansohanrohangohan
dtype: object
```

2.0.6 subset of our dataframe

[38]:
```
df2 = df1[['a','b']] #subsetting dataframe with only numerical column
```

[39]:
```
df2
```

[39]:

	a	b
a	1	4
b	2	5
c	3	6
d	4	7

2.0.7 .apply() for a single column to perform some operation

2.0.8 .applymap for whole dataset or whole column, to perform some operation

[41]:
```
df2.applymap(lambda x :x**2) #squaring all columns in dataset with applymap()
```

[41]:

	a	b
a	1	16
b	4	25
c	9	36
d	16	49

[42]:
```
df1
```

[42]:

	a	b	c
a	1	4	mohan
b	2	5	sohan
c	3	6	rohan
d	4	7	gohan

2.0.9 sorting dataframe with respect to a column

2.0.10 sort_values()

[44]:
```
df1.sort_values('c')
```

[44]:
```
   a  b      c
d  4  7  gohan
a  1  4  mohan
c  3  6  rohan
b  2  5  sohan
```

2.0.11 sorting based on indexes

2.0.12 sort_index()

[48]:
```
df1.sort_index(ascending=False) # sorting in descending order, bydefault it's␣
↪in ascending order
```

[48]:
```
   a  b      c
d  4  7  gohan
c  3  6  rohan
b  2  5  sohan
a  1  4  mohan
```

[ ]:
```

```

2.0.13 how to see complete data if we have large amount of data

[51]:
```
df3 = pd.DataFrame({"desc":["Data Science Masters course is highly curated and␣
↪uniquely designed according to the latest industry standards. This program␣
↪instills students the skills essential to knowledge discovery efforts to␣
↪identify standard, novel, and truly differentiated solutions and␣
↪decision-making, including skills in managing, querying, analyzing,␣
↪visualizing, and extracting meaning from extremely large data sets. This␣
↪trending program provides students with the statistical, mathematical and␣
↪computational skills needed to meet the large-scale data science challenges␣
↪of today's professional world. You will learn all the stack required to work␣
↪in data science industry including cloud infrastructure and real-time␣
↪industry projects. This course will be taught in Hindi language."]})
```

[52]:
```
df3
```

[52]:
```
                                                desc
0  Data Science Masters course is highly curated …
```

[55]:
```
pd.set_option("display.max_colwidth",500)  # it'll show 500 character or size␣
 ↪of data
# we can set display limit to our own need
```

[56]:
```
df3
```

[56]:
```
                    desc
0  Data Science Masters course is highly curated and uniquely designed according
to the latest industry standards. This program instills students the skills
essential to knowledge discovery efforts to identify standard, novel, and truly
differentiated solutions and decision-making, including skills in managing,
querying, analyzing, visualizing, and extracting meaning from extremely large
data sets. This trending program provides students with the statistical,
mathematical and computational skil…
```

[57]:
```
pd.set_option("display.max_colwidth",1000)
```

[58]:
```
df3
```

[58]:
```
                                                     desc
0  Data Science Masters course is highly curated and uniquely designed according
to the latest industry standards. This program instills students the skills
essential to knowledge discovery efforts to identify standard, novel, and truly
differentiated solutions and decision-making, including skills in managing,
querying, analyzing, visualizing, and extracting meaning from extremely large
data sets. This trending program provides students with the statistical,
mathematical and computational skills needed to meet the large-scale data
science challenges of today's professional world. You will learn all the stack
required to work in data science industry including cloud infrastructure and
real-time industry projects. This course will be taught in Hindi language.
```

[ ]:
```

```

[59]:
```
df3 = pd.DataFrame({"desc":["Data Science Masters course is highly curated and␣
 ↪uniquely designed according to the latest industry standards. This program␣
 ↪instills students the skills essential to knowledge discovery efforts to␣
 ↪identify standard, novel, and truly differentiated solutions and␣
 ↪decision-making, including skills in managing, querying, analyzing,␣
 ↪visualizing, and extracting meaning from extremely large data sets. This␣
 ↪trending program provides students with the statistical, mathematical and␣
 ↪computational skills needed to meet the large-scale data science challenges␣
 ↪of today's professional world. You will learn all the stack required to work␣
 ↪in data science industry including cloud infrastructure and real-time␣
 ↪industry projects. This course will be taught in Hindi language.","my name␣
 ↪is lakhan","i use to take data science master class"]})
```

[61]:
```
df3 # added more data to our dataset
```

[61]:
```
                                                desc
0  Data Science Masters course is highly curated and uniquely designed according
to the latest industry standards. This program instills students the skills
essential to knowledge discovery efforts to identify standard, novel, and truly
differentiated solutions and decision-making, including skills in managing,
querying, analyzing, visualizing, and extracting meaning from extremely large
data sets. This trending program provides students with the statistical,
mathematical and computational skills needed to meet the large-scale data
science challenges of today's professional world. You will learn all the stack
required to work in data science industry including cloud infrastructure and
real-time industry projects. This course will be taught in Hindi language.
1
my name is lakhan
2
i use to take data science master class
```

2.0.14 calculating the character length string in each rows of 'desc' column and saving it in new column

[63]:
```
df3['char_len_data'] = df3['desc'].apply(len)
```

[64]:
```
df3
```

[64]:
```
                                                desc  \
0  Data Science Masters course is highly curated and uniquely designed according
to the latest industry standards. This program instills students the skills
essential to knowledge discovery efforts to identify standard, novel, and truly
differentiated solutions and decision-making, including skills in managing,
querying, analyzing, visualizing, and extracting meaning from extremely large
data sets. This trending program provides students with the statistical,
mathematical and computational skills needed to meet the large-scale data
science challenges of today's professional world. You will learn all the stack
required to work in data science industry including cloud infrastructure and
real-time industry projects. This course will be taught in Hindi language.
1
my name is lakhan
2
i use to take data science master class

   char_len_data
0            765
1             17
2             39
```

2.0.15 creating new column with no of words (word count)

[65]:
```
t = "i use to take data science master class"
t.split() #return a list where a string is seperated by space
```

[65]:
```
['i', 'use', 'to', 'take', 'data', 'science', 'master', 'class']
```

[67]:
```
len(t.split()) # total no.of words
```

[67]:
```
8
```

[68]:
```
df3['word_count'] = df3['desc'].apply(lambda x: len(x.split()))
```

[69]:
```
df3
```

[69]:
```
                                                desc  \
0  Data Science Masters course is highly curated and uniquely designed according
to the latest industry standards. This program instills students the skills
essential to knowledge discovery efforts to identify standard, novel, and truly
differentiated solutions and decision-making, including skills in managing,
querying, analyzing, visualizing, and extracting meaning from extremely large
data sets. This trending program provides students with the statistical,
mathematical and computational skills needed to meet the large-scale data
science challenges of today's professional world. You will learn all the stack
required to work in data science industry including cloud infrastructure and
real-time industry projects. This course will be taught in Hindi language.
1
my name is lakhan
2
i use to take data science master class

   char_len_data  word_count
0            765         104
1             17           4
2             39           8
```

[ ]:
```

```

[ ]:
```

```

3 performing mathematical operation

[70]:
```
df1
```

[70]:
```
   a  b      c
a  1  4  mohan
```

```
b  2  5  sohan
c  3  6  rohan
d  4  7  gohan
```

3.0.1 Average of 'a' column, using mean()

[71]:
```
df1['a'].mean()
```

[71]:
```
2.5
```

3.0.2 median of column 'a', middle value of data

[73]:
```
df1['a'].median()
```

[73]:
```
2.5
```

3.0.3 mode (frequency) of column 'a' data

[74]:
```
df1['a'].mode()
```

[74]:
```
0    1
1    2
2    3
3    4
Name: a, dtype: int64
```

3.0.4 standard deviation of column 'a' data

dispersion from the mean or distance from the mean

[75]:
```
df1['a'].std()
```

[75]:
```
1.2909944487358056
```

3.0.5 minimum value in column 'a'

[76]:
```
df1['a'].min()
```

[76]:
```
1
```

3.0.6 maximum value in column 'a'

[77]:
```
df1['a'].max()
```

[77]:
```
4
```

3.0.7 summation of entire column 'a' in dataset

[78]:
```
df1['a'].sum()
```

[78]:
```
10
```

3.0.8 variance in column

variance is spread of data

[79]:
```
df1['a'].var()
```

[79]:
```
1.6666666666666667
```

4 performing all these mathematical operation in a Series datatype

5 because we are selection a column to perform all these mathematical operation

[ ]:
```

```

[ ]:
```

```

6 windowing function, rolling windows

[3]:
```
df4 = pd.DataFrame({'a':[1,2,3,4,5,6,7,8,9]})
```

[4]:
```
df4 # small simple dataframe
```

[4]:

	a
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9

6.0.1 rolling window concept

[5]:
```
df4.rolling(window=1).mean()
```

[5]:
```
     a
0  1.0
1  2.0
2  3.0
3  4.0
4  5.0
5  6.0
6  7.0
7  8.0
8  9.0
```

6.0.2 scaler topic example

[2]:
```
import pandas as pd
df_sc = pd.DataFrame({"students_score":[35,38,39,30,20,26,29]})
df_sc
```

[2]:
```
   students_score
0              35
1              38
2              39
3              30
4              20
5              26
6              29
```

[ ]:
```
df_sc.rolling(window=1).apply(lambda x : x.iloc[0]-x.iloc[1])

## ERROR
```

[13]:
```
df_sc.rolling(window=2).apply(lambda x : x.iloc[0]-x.iloc[1])
```

[13]:
```
   students_score
0             NaN
1            -3.0
2            -1.0
3             9.0
4            10.0
5            -6.0
6            -3.0
```

[19]:
```
df_sc.rolling(window=2).apply(lambda x : x.iloc[1]-x.iloc[0])
```

[19]:
```
   students_score
0             NaN
1             3.0
2             1.0
```

```
3            -9.0
4           -10.0
5             6.0
6             3.0
```

[17]:
```
df_sc.rolling(window=2).sum()
```

[17]:
```
   students_score
0             NaN
1            73.0
2            77.0
3            69.0
4            50.0
5            46.0
6            55.0
```

[ ]:
```

```

7

[6]:
```
df4.rolling(window=2).mean() # return mean of 2 rows because window=2
```

[6]:
```
     a
0  NaN
1  1.5
2  2.5
3  3.5
4  4.5
5  5.5
6  6.5
7  7.5
8  8.5
```

[7]:
```
df4.rolling(window=3).mean()
```

[7]:
```
     a
0  NaN
1  NaN
2  2.0
3  3.0
4  4.0
5  5.0
6  6.0
7  7.0
8  8.0
```

8 a window is set of dataset we take to perform some operation

8.0.1 kind of cummulitive take data in rows, according to windows size

8.0.2 rolling: keep rolling to next row

[9]:
```
df4.rolling(window=3).sum() # sum of three rows data one after another
```

[9]:
```
     a
0  NaN
1  NaN
2  6.0
3  9.0
4 12.0
5 15.0
6 18.0
7 21.0
8 24.0
```

[11]:
```
df4.rolling(window=3).min()
```

[11]:
```
     a
0  NaN
1  NaN
2  1.0
3  2.0
4  3.0
5  4.0
6  5.0
7  6.0
8  7.0
```

9 cumulative sum: cumsum()

[13]:
```
df4.cumsum() #row by row add all above rows data/number
```

[13]:
```
    a
0   1
1   3
2   6
3  10
4  15
5  21
6  28
7  36
8  45
```

10 python pandas - Date Functionality

- pandas.date_range(),
- pandas.to_datetime(),
- pandas.Timedelta()

10.0.1 when try to read date based data and perform date wise operaion

[17]:
```
data = pd.date_range(start='2022-03-22',end = '2022-06-22')
# created list of dates, from start to end
```

[18]:
```
data #print date from start to end given date
```

[18]:
```
DatetimeIndex(['2022-03-22', '2022-03-23', '2022-03-24', '2022-03-25',
               '2022-03-26', '2022-03-27', '2022-03-28', '2022-03-29',
               '2022-03-30', '2022-03-31', '2022-04-01', '2022-04-02',
               '2022-04-03', '2022-04-04', '2022-04-05', '2022-04-06',
               '2022-04-07', '2022-04-08', '2022-04-09', '2022-04-10',
               '2022-04-11', '2022-04-12', '2022-04-13', '2022-04-14',
               '2022-04-15', '2022-04-16', '2022-04-17', '2022-04-18',
               '2022-04-19', '2022-04-20', '2022-04-21', '2022-04-22',
               '2022-04-23', '2022-04-24', '2022-04-25', '2022-04-26',
               '2022-04-27', '2022-04-28', '2022-04-29', '2022-04-30',
               '2022-05-01', '2022-05-02', '2022-05-03', '2022-05-04',
               '2022-05-05', '2022-05-06', '2022-05-07', '2022-05-08',
               '2022-05-09', '2022-05-10', '2022-05-11', '2022-05-12',
               '2022-05-13', '2022-05-14', '2022-05-15', '2022-05-16',
               '2022-05-17', '2022-05-18', '2022-05-19', '2022-05-20',
               '2022-05-21', '2022-05-22', '2022-05-23', '2022-05-24',
               '2022-05-25', '2022-05-26', '2022-05-27', '2022-05-28',
               '2022-05-29', '2022-05-30', '2022-05-31', '2022-06-01',
               '2022-06-02', '2022-06-03', '2022-06-04', '2022-06-05',
               '2022-06-06', '2022-06-07', '2022-06-08', '2022-06-09',
               '2022-06-10', '2022-06-11', '2022-06-12', '2022-06-13',
               '2022-06-14', '2022-06-15', '2022-06-16', '2022-06-17',
               '2022-06-18', '2022-06-19', '2022-06-20', '2022-06-21',
               '2022-06-22'],
              dtype='datetime64[ns]', freq='D')
```

[21]:
```
df_date = pd.DataFrame({"date" :data})
#create dataframe of date column
```

[ ]:
```
df_date ## display all data
```

[24]:
```
df_date.dtypes #datatype of date column will be 'datetime'
```

[24]:
```
date    datetime64[ns]
dtype: object
```

[25]:
```
df7 = pd.DataFrame({"date":['2022-06-22','2022-06-20','2022-06-19']})
```

creating dataframe with directly giving dates list as a 'string'

passing dates as a string so datatype will be object not datetime format

mostly in fetched data data-time will be in string format so have to convert it datetime

10.1 to perform datetime operation column should be in datetime datatype, then only we can fetch only year or day or date

[26]:
```
df7
```

[26]:
```
         date
0  2022-06-22
1  2022-06-20
2  2022-06-19
```

[27]:
```
df7.dtypes
```

[27]:
```
date    object
dtype: object
```

11 convert datetime string datatype to proper datetime format

12 pandas.to_datetime()

[34]:
```
df7['update_data']= pd.to_datetime(df7['date'])
#creating new column with updated date in datetime format datatype
```

[29]:
```
df7
```

[29]:
```
         date update_data update_date
0  2022-06-22  2022-06-22  2022-06-22
1  2022-06-20  2022-06-20  2022-06-20
2  2022-06-19  2022-06-19  2022-06-19
```

[35]:
```
df7['update_date']= pd.to_datetime(df7['date'])
```

[36]:
```
df7
```

[36]:
```
         date update_data update_date
0  2022-06-22  2022-06-22  2022-06-22
1  2022-06-20  2022-06-20  2022-06-20
2  2022-06-19  2022-06-19  2022-06-19
```

[38]:
```
df7.drop('update_data', axis =1)
```

[38]:
```
         date update_date
0  2022-06-22  2022-06-22
1  2022-06-20  2022-06-20
2  2022-06-19  2022-06-19
```

[42]:
```
df7
```

[42]:
```
         date update_data update_date  year
0  2022-06-22  2022-06-22  2022-06-22  2022
1  2022-06-20  2022-06-20  2022-06-20  2022
2  2022-06-19  2022-06-19  2022-06-19  2022
```

[43]:
```
df7.drop('update_data', axis =1, inplace=True)
```

[44]:
```
df7
```

[44]:
```
         date update_date  year
0  2022-06-22  2022-06-22  2022
1  2022-06-20  2022-06-20  2022
2  2022-06-19  2022-06-19  2022
```

13 both looks same but their datatype is different

[30]:
```
df7.dtypes
```

[30]:
```
date                   object
update_data    datetime64[ns]
update_date    datetime64[ns]
dtype: object
```

when column in datetime datatype format we can perform many datetime operation on it

fetching only years on present dataset

[45]:
```
df7['year'] = df7['update_date'].dt.year
#creating new column 'year' and storing years only, after fetching data from␣
 ↪'update_date'
```

[46]:
```
df7
```

[46]:

	date	update_date	year
0	2022-06-22	2022-06-22	2022
1	2022-06-20	2022-06-20	2022
2	2022-06-19	2022-06-19	2022

extracting only month from column and storing in new column named 'month'

[47]:
```
df7['month'] = df7['update_date'].dt.month
```

[48]:
```
df7
```

[48]:

	date	update_date	year	month
0	2022-06-22	2022-06-22	2022	6
1	2022-06-20	2022-06-20	2022	6
2	2022-06-19	2022-06-19	2022	6

extracting only day from column and storing in new column named 'day'

[51]:
```
df7['day'] = df7['update_date'].dt.day
```

[52]:
```
df7
```

[52]:

	date	update_date	year	month	day
0	2022-06-22	2022-06-22	2022	6	22
1	2022-06-20	2022-06-20	2022	6	20
2	2022-06-19	2022-06-19	2022	6	19

14 python pandas : Time Delta- pandas.Timedelta()

it just time difference it'll create time difference

[54]:
```
pd.Timedelta(days=1) # means difference of one day
```

[54]:
```
Timedelta('1 days 00:00:00')
```

[56]:
```
pd.Timedelta(days=1, hours = 5) # one day and 5 hours
```

[56]:
```
Timedelta('1 days 05:00:00')
```

[57]:
```
pd.Timedelta(days=1, hours=5, minutes=55) # giving time difference of 1 day 5␣
↪hours and 55 minutes
```

[57]:
```
Timedelta('1 days 05:55:00')
```

[59]:
```
time = pd.Timedelta(days=1, hours=5, minutes=55)# storing in a variable
```

[62]:
```
time
```

[62]:
```
Timedelta('1 days 05:55:00')
```

[60]:
```
dt = pd.to_datetime('2022-06-20') # creating a datetime variable
```

[61]:
```
dt
```

[61]:
```
Timestamp('2022-06-20 00:00:00')
```

[63]:
```
dt+time # date has chaged to 21 and, hours and minutes has been added because␣
↪our Timedelta had 1day,5hours and 55minutes
```

[63]:
```
Timestamp('2022-06-21 05:55:00')
```

15 python pandas- Categorical Data

[66]:
```
data = ["rohan", "mohan", "sohan", "gohan", "rohan","rohan"]
```

[70]:
```
pd.Categorical(data) # tells category present in data
#return distinct category present in data
```

[70]:
```
['rohan', 'mohan', 'sohan', 'gohan', 'rohan', 'rohan']
Categories (4, object): ['gohan', 'mohan', 'rohan', 'sohan']
```

[71]:
```
cat = pd.Categorical(data)
```

[72]:
```
cat.value_counts() # gives counts of group-by element present
# group-by count, kind of a operation
```

[72]:
```
gohan    1
mohan    1
rohan    3
sohan    1
dtype: int64
```

[ ]:
```

```

16 python Pandas - Visualization

matplotlib, seaborn, plotly, bokeh and other library has visuality function

- but pandas has its own function to visualize data

[73]:
```
d = pd.Series([1,2,3,4,5,6,7,8])
```

[75]:
```
d
```

[75]:
```
0    1
1    2
2    3
3    4
4    5
5    6
6    7
7    8
dtype: int64
```

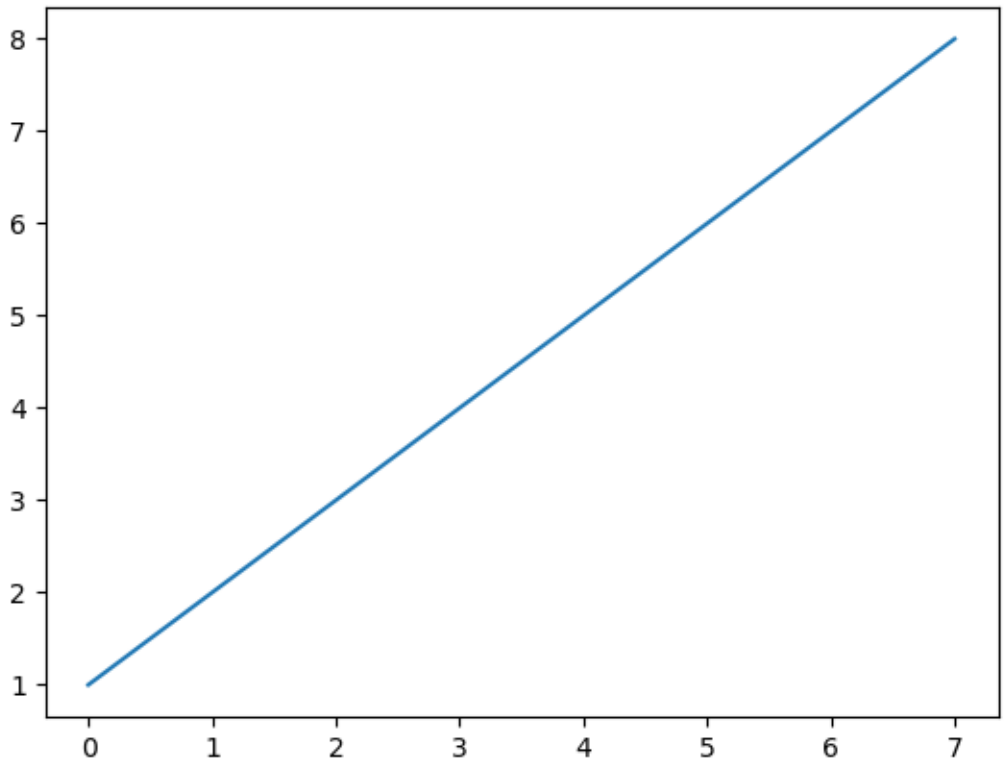
17 let's see how our data in moving visually

18 representation of values in 2-D graph

19 to get insight of data

[77]:
```
d.plot() #give a graph of data
```

[77]:
```
<AxesSubplot: >
```



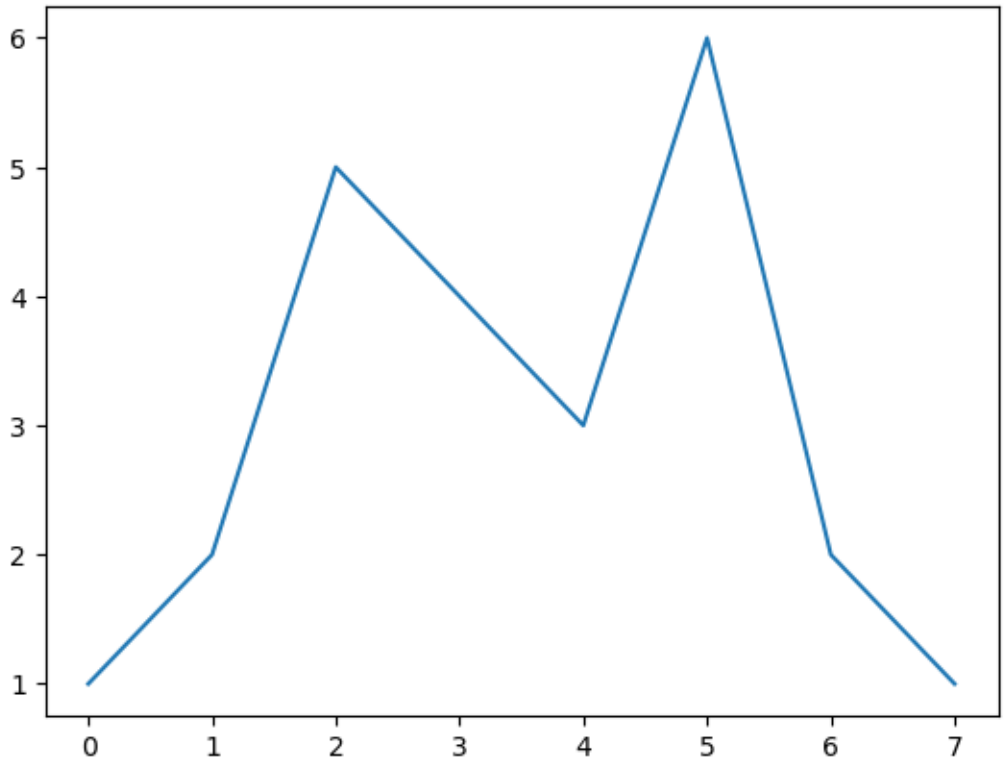
[78]:
```
d2 = pd.Series([1,2,5,4,3,6,2,1])
```

[80]:
```
d2
```

[80]:
```
0    1
1    2
2    5
3    4
4    3
5    6
6    2
7    1
dtype: int64
```

[82]:
```
d2.plot() # indexes are available on x-axis and values are available on y-axis
# then it's trying to create pair of cordinates( two numbers in index and value)
```

[82]:
```
<AxesSubplot: >
```



[ ]:
```

```