Pandas

1. Series

- Generate Series from list
- Generate Series from dict
- Generate Series from csv

2. DataFrame

- Generate Dataframe from list
- Generate Dataframe from dict
- Generate Dataframe from csv

3. Reindexing
4. attributes of Series and Dataframe

- T, dtype, shape, index, columns, axis,values, empty(Series), size(Series), is_uniqe(Series)

5. Indexing, Selecting, Filtering

- at, iat, loc, iloc

6. Summarizing and Computing Descriptive Statistics(Unlike NumPy arrays, Pandas descriptive statistics automatically exclude missing data. NaN values are excluded unless the entire row or column is NA)

- head(), tail(), info(),describe()

7. Handling Missing Value

- find missing value
- fill the value or drop the value

8. Handle duplicate

- Identifying Duplicate Records in a Pandas DataFrame
- Removing Duplicate Data in a Pandas DataFrame¶

9. Handle inconsistent data: Identify and correct any inconsistent data in the DataFrame using the replace() function or by manually changing the data.
10. Handling outliers: Outliers can be detected and handled using various statistical methods. The z-score and interquartile range (IQR) methods are commonly used for this purpose.
11. Handle data types: Convert columns to the correct data type using the astype() function
12. Handle categorical data: Convert categorical data to numeric using the get_dummies() function.

- some other step require

13. remove outliers
14. crosstab
15. corr(),parallel_cordinates

Vishal Acharya

What is Pandas

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

pandas Series

- A Pandas Series is like a column in a table. It is a 1-D array holding data of any type.
- A Series is a one-dimensional array-like object containing an array of data and an associated array of data labels. The data can be any NumPy data type and the labels are the Series' index.

pandas dataframe

Data frame: A Pandas Data Frame is a 2-dimensional data structure, like a 2-dimensional array, or a table with rows and columns.

IMPORT PANDAS LIBRARY

In [19]:
```
import numpy as np  #(it's beneficial)
import pandas as pd
```

Vishal Acharya

class pandas.Series(data=None, index=None, dtype=None, name=None, copy=None, fastpath=False)

```
One-dimensional ndarray with axis labels (including time series).
Labels need not be unique but must be a hashable type. The object
supports both integer- and label-based indexing and provides a host of
methods for performing operations involving the index. Statistical
methods from ndarray have been overridden to automatically exclude
missing data (currently represented as NaN).
Operations between Series (+, -, /, \\*, \\*\\*) align values based on their
associated index values-- they need not be the same length. The result
index will be the sorted union of the two indexes.
    Parameters
----------
data : array-like, Iterable, dict, or scalar value
    Contains data stored in Series. If data is a dict, argument order is
    maintained.
index : array-like or Index (1d)
    Values must be hashable and have the same length as `data`.
    Non-unique index values are allowed. Will default to
    RangeIndex (0, 1, 2, ..., n) if not provided. If data is dict-like
    and index is None, then the keys in the data are used as the index. If the
    index is not None, the resulting Series is reindexed with the index values.
dtype : str, numpy.dtype, or ExtensionDtype, optional
    Data type for the output Series. If not specified, this will be
    inferred from `data`.
    See the :ref:`user guide <basics.dtypes>` for more usages.
name : Hashable, default None
    The name to give to the Series.
copy : bool, default False
    Copy input data. Only affects Series or 1d ndarray input.
```

Generate Series from list

In [20]:
```
# string
Friends = ['vishal','milan','tushar','kavit','manish']
(pd.Series(Friends))
```

Out[20]:
```
0    vishal
1     milan
2    tushar
3     kavit
4    manish
dtype: object
```

In [21]:
```
# integers
marks = [13,24,56,78,100]

marks_ser = pd.Series(marks)
print(marks_ser)
```

```
0     13
1     24
2     56
3     78
4    100
dtype: int64
```

In [22]:
```
# custom index
marks = [97,97,99,100]
subjects = ['python2','fsd2','toc','dm']

pd.Series(marks,index=subjects)
```

Out[22]:
```
python2     97
fsd2        97
toc         99
dm         100
dtype: int64
```

In [23]:
```
# setting a name
marks = pd.Series(marks,index=subjects,name='vishal Score')
marks
```

Out[23]:
```
python2     97
fsd2        97
toc         99
dm         100
Name: vishal Score, dtype: int64
```

Vishal Acharya

Generate series from dict

In [24]:
```
marks = {
    'fsd':97,
    'python':77,
    'DM':80,
    'TOC':100
}
marks_series = pd.Series(marks,name='vishal score')
print(type(marks_series))
print(marks_series)
```

```
<class 'pandas.core.series.Series'>
fsd        97
python     77
DM         80
TOC       100
Name: vishal score, dtype: int64
```

In [25]:
```
d = {'a': 1, 'b': 2, 'c': 3}
ser = pd.Series(data=d, index=['a', 'b', 'c'])
ser

```

Out[25]:
```
a    1
b    2
c    3
dtype: int64
```

In [26]:
```
d = {'a': 1, 'b': 2, 'c': 3}
ser = pd.Series(data=d, index=['x', 'y', 'z'])
ser
```

Out[26]:
```
x   NaN
y   NaN
z   NaN
dtype: float64
```

In [27]:
```
r = [1, 2]
ser = pd.Series(r, copy=False)
print(r)
print("***")
print(ser)
ser.iloc[0] = 999
print("***")
print(ser)
print("***")
print(r,"r")
```

```
[1, 2]
***
0    1
1    2
dtype: int64
***
0    999
1      2
dtype: int64
***
[1, 2] r
```

Vishal Acharya

In [28]:

```
r = np.array([1, 2])
ser = pd.Series(r, copy=False)
print(r)
print("***")

ser.iloc[0] = 999
print(ser)
print("***")
print(ser)
print("***")
print(r)
```

```
[1 2]
***
0    999
1      2
dtype: int32
***
0    999
1      2
dtype: int32
***
[999   2]
```

In [29]:

```
r = np.array([1, 2])
ser = pd.Series(r, copy=True)
print(ser)
ser.iloc[0] = 999
print("***")
print(ser)
print("***")
print(r)
```

```
0    1
1    2
dtype: int32
***
0    999
1      2
dtype: int32
***
[1 2]
```

Vishal Acharya

Generate series from CSV

In [30]:
```
# with one col
subs = pd.read_csv('vha.csv')
print(type(subs))
print(subs)
```

```
<class 'pandas.core.frame.DataFrame'>
     cgpa
0    6.89
1    5.12
2    7.82
3    7.42
4    6.94
..    ...
195  6.93
196  5.89
197  7.21
198  7.63
199  6.22

[200 rows x 1 columns]
```

In [31]:
```
# with one col
vha_ser = pd.read_csv('vha.csv')
ds=vha_ser["cgpa"]
print(type(ds))
print(ds)
```

```
<class 'pandas.core.series.Series'>
0      6.89
1      5.12
2      7.82
3      7.42
4      6.94
       ...
195    6.93
196    5.89
197    7.21
198    7.63
199    6.22
Name: cgpa, Length: 200, dtype: float64
```

Vishal Acharya

In [32]:
```
movie=pd.read_csv('movie.csv')
print(type(movie))
movie
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[32]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	a
0	Color	James Cameron	723.0	178.0	0.0	855.0	
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	
2	Color	Sam Mendes	602.0	148.0	0.0	161.0	
3	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	
4	NaN	Doug Walker	NaN	NaN	131.0	NaN	
...	...	...	...	...	...	...	
4911	Color	Scott Smith	1.0	87.0	2.0	318.0	
4912	Color	NaN	43.0	43.0	NaN	319.0	
4913	Color	Benjamin Roberds	13.0	76.0	0.0	0.0	
4914	Color	Daniel Hsia	14.0	100.0	0.0	489.0	
4915	Color	Jon Gunn	43.0	90.0	16.0	16.0	

4916 rows × 28 columns

In [33]:
```
director=movie["director_name"]
print(type(director))
director
```

```
<class 'pandas.core.series.Series'>
```

Out[33]:

```
0           James Cameron
1          Gore Verbinski
2              Sam Mendes
3       Christopher Nolan
4             Doug Walker
                 ...
4911          Scott Smith
4912                  NaN
4913     Benjamin Roberds
4914          Daniel Hsia
4915             Jon Gunn
Name: director_name, Length: 4916, dtype: object
```

Vishal Acharya

Creating DataFrame using list

In [34]:
```
# using lists
cse_data = [
    [100,80,15],
    [60,70,17],
    [120,80,14],
    [80,50,20]
]

df_1=pd.DataFrame(cse_data,columns=['iq','marks','package'])
df_1
```

Out[34]:

	iq	marks	package
0	100	80	15
1	60	70	17
2	120	80	14
3	80	50	20

Creating DataFrame using dict

In [35]:
```
cse_dict = {
    'name':['vishal','kavit','manish','milan','amit','tushar'],
    'iq':[100,50,120,80,0,0],
    'marks':[80,60,100,70,10,0],
    'package':[10,7,14,20,0,0]
}

cse = pd.DataFrame(cse_dict)
cse.set_index('name',inplace=True)
cse
```

Out[35]:

name	iq	marks	package
vishal	100	80	10
kavit	50	60	7
manish	120	100	14
milan	80	70	20
amit	0	10	0
tushar	0	0	0

Vishal Acharya

Use the read_csv function to read in the movie dataset

In [36]:
```
# using read_csv
movie = pd.read_csv('movie.csv')
movie
```

Out[36]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	a
0	Color	James Cameron	723.0	178.0	0.0	855.0	
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	
2	Color	Sam Mendes	602.0	148.0	0.0	161.0	
3	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	
4	NaN	Doug Walker	NaN	NaN	131.0	NaN	
...	...	...	...	...	...	...	
4911	Color	Scott Smith	1.0	87.0	2.0	318.0	
4912	Color	NaN	43.0	43.0	NaN	319.0	
4913	Color	Benjamin Roberds	13.0	76.0	0.0	0.0	
4914	Color	Daniel Hsia	14.0	100.0	0.0	489.0	
4915	Color	Jon Gunn	43.0	90.0	16.0	16.0	

4916 rows × 28 columns

Vishal Acharya

Reindexing

- Create a new object with the data conformed to a new index. Any missing values are set to NaN.

In [37]:
```
data_1 = {'state' : ['VA', 'VA', 'VA', 'MD', 'MD'],
          'year' : [2012, 2013, 2014, 2014, 2015],
          'pop' : [5.0, 5.1, 5.2, 4.0, 4.1]}
df_1 = pd.DataFrame(data_1)
df_1
```

Out[37]:

	state	year	pop
0	VA	2012	5.0
1	VA	2013	5.1
2	VA	2014	5.2
3	MD	2014	4.0
4	MD	2015	4.1

In [38]:
```
df_1=df_1.reindex(list(reversed(range(0, 6))))
df_1
```

Out[38]:

	state	year	pop
5	NaN	NaN	NaN
4	MD	2015.0	4.1
3	MD	2014.0	4.0
2	VA	2014.0	5.2
1	VA	2013.0	5.1
0	VA	2012.0	5.0

Both Series and DataFrames have a tremendous amount of power. We can use the dir function to uncover all the attributes and methods of a Series. Additionally, we can find the number of attributes and methods common to both Series and DataFrames. Both of these objects share the vast majority of attribute and method name

Vishal Acharya

In [39]:
```
ser_attr_methods = set(dir(pd.Series))
print("ser_attr_methods",len(ser_attr_methods))
#print(ser_attr_methods)
```

```
ser_attr_methods 417
```

In [40]:
```
df_attr_methods = set(dir(pd.DataFrame))
print("df_attr_methods",len(df_attr_methods))
```

```
df_attr_methods 435
```

In [41]:
```
print("common_attr_methods",len(ser_attr_methods & df_attr_methods))
```

```
common_attr_methods 361
```

attributes of Series and Dataframe

- T, dtype, shape, index, columns, axis,values, empty, ndim. size, is_uniqe(Series)

T

The transpose of the DataFrame.

In [42]:
```
1 movie.T
```

Out[42]:

	0	1	
color	Color	Color	
director_name	James Cameron	Gore Verbinski	
num_critic_for_reviews	723.0	302.0	
duration	178.0	169.0	
director_facebook_likes	0.0	563.0	
actor_3_facebook_likes	855.0	1000.0	
actor_2_name	Joel David Moore	Orlando Bloom	
actor_1_facebook_likes	1000.0	40000.0	
gross	760505847.0	309404152.0	
genres	Action|Adventure|Fantasy|Sci-Fi	Action|Adventure|Fantasy	Actio
actor_1_name	CCH Pounder	Johnny Depp	
movie_title	Avatar	Pirates of the Caribbean: At World's End	
num_voted_users	886204	471220	
cast_total_facebook_likes	4834	48350	
actor_3_name	Wes Studi	Jack Davenport	
facenumber_in_poster	0.0	0.0	
plot_keywords	avatar|future|marine|native|paraplegic	goddess|marriage ceremony|marriage proposal|pi...	bomb|espionage
movie_imdb_link	http://www.imdb.com/title/tt0499549/?ref_=fn_t...	http://www.imdb.com/title/tt0449088/?ref_=fn_t...	http://www.imdb.c
num_user_for_reviews	3054.0	1238.0	
language	English	English	
country	USA	USA	
content_rating	PG-13	PG-13	
budget	237000000.0	300000000.0	
title_year	2009.0	2007.0	
actor_2_facebook_likes	936.0	5000.0	
imdb_score	7.9	7.1	
aspect_ratio	1.78	2.35	
movie_facebook_likes	33000	0	

28 rows × 4916 columns

Vishal Acharya

In [43]:
```
1 df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
2 df
```

Out[43]:

	col1	col2
0	1	3
1	2	4

In [44]:
```
df.T
```

Out[44]:

	0	1
col1	1	2
col2	3	4

axes

Return a list representing the axes of the DataFrame.

In [45]:
```
df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
print(df)
print("#"*58)
print(df.axes)
```

```
   col1  col2
0     1     3
1     2     4
##########################################################
[RangeIndex(start=0, stop=2, step=1), Index(['col1', 'col2'], dtype='object')]
```

In [46]:
```
print(movie.axes)
```

```
[RangeIndex(start=0, stop=4916, step=1), Index(['color', 'director_name', 'num_critic_fo
r_reviews', 'duration',
       'director_facebook_likes', 'actor_3_facebook_likes', 'actor_2_name',
       'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
       'movie_title', 'num_voted_users', 'cast_total_facebook_likes',
       'actor_3_name', 'facenumber_in_poster', 'plot_keywords',
       'movie_imdb_link', 'num_user_for_reviews', 'language', 'country',
       'content_rating', 'budget', 'title_year', 'actor_2_facebook_likes',
       'imdb_score', 'aspect_ratio', 'movie_facebook_likes'],
      dtype='object')]
```

columns

The column labels of the DataFrame.

Vishal Acharya

In [47]:
```
cse_dict = {
    'name':['vishal','kavit','manish','milan','amit','tushar'],
    'iq':[100,50,120,80,0,0],
    'marks':[80,60,100,70,10,0],
    'package':[10,7,14,20,0,0]
}

cse = pd.DataFrame(cse_dict)
cse.set_index('name',inplace=True)
print(cse)
print("*"*50)
print(cse.columns)
```

```
         iq  marks  package
name
vishal  100     80       10
kavit    50     60        7
manish  120    100       14
milan    80     70       20
amit      0     10        0
tushar    0      0        0
**************************************************
Index(['iq', 'marks', 'package'], dtype='object')
```

In [48]:
```
print(movie.columns)
```

```
Index(['color', 'director_name', 'num_critic_for_reviews', 'duration',
       'director_facebook_likes', 'actor_3_facebook_likes', 'actor_2_name',
       'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
       'movie_title', 'num_voted_users', 'cast_total_facebook_likes',
       'actor_3_name', 'facenumber_in_poster', 'plot_keywords',
       'movie_imdb_link', 'num_user_for_reviews', 'language', 'country',
       'content_rating', 'budget', 'title_year', 'actor_2_facebook_likes',
       'imdb_score', 'aspect_ratio', 'movie_facebook_likes'],
      dtype='object')
```

dtypes

Return the dtypes in the DataFrame.

Vishal Acharya

In [49]:
```
print(movie.dtypes)
```

```
color                       object
director_name               object
num_critic_for_reviews     float64
duration                   float64
director_facebook_likes    float64
actor_3_facebook_likes     float64
actor_2_name                object
actor_1_facebook_likes     float64
gross                      float64
genres                      object
actor_1_name                object
movie_title                 object
num_voted_users              int64
cast_total_facebook_likes    int64
actor_3_name                object
facenumber_in_poster       float64
plot_keywords               object
movie_imdb_link             object
num_user_for_reviews       float64
language                    object
country                     object
content_rating              object
budget                     float64
title_year                 float64
actor_2_facebook_likes     float64
imdb_score                 float64
aspect_ratio               float64
movie_facebook_likes         int64
dtype: object
```

In [50]:
```
print(cse.dtypes)
```

```
iq         int64
marks      int64
package    int64
dtype: object
```

In [51]:
```
print(ds.dtypes)
```

```
float64
```

empty

Indicator whether Series/DataFrame is empty.

In [52]:
```
df_empty = pd.DataFrame({'A' : []})
df_empty
```

Out[52]:

A

In [53]:
```
df_empty.empty
```

Out[53]:
```
True
```

Vishal Acharya

In [54]:
```
movie.empty
```

Out[54]:
```
False
```

In [55]:
```
df = pd.DataFrame({'A' : [np.nan]})
df
```

Out[55]:

	A
0	NaN

In [56]:
```
df.empty
```

Out[56]:
```
False
```

index

The index (row labels) of the DataFrame.

In [57]:
```
movie.index
```

Out[57]:
```
RangeIndex(start=0, stop=4916, step=1)
```

In [58]:
```
ds.index
```

Out[58]:
```
RangeIndex(start=0, stop=200, step=1)
```

In [59]:
```
cse.index
```

Out[59]:
```
Index(['vishal', 'kavit', 'manish', 'milan', 'amit', 'tushar'], dtype='object', name='name')
```

ndim

Return an int representing the number of axes / array dimensions.

In [60]:
```
s = pd.Series({'a': 1, 'b': 2, 'c': 3})
s.ndim
```

Out[60]:
```
1
```

In [61]:
```
df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
df.ndim
```

Out[61]:
```
2
```

In [62]:
```
movie.ndim
```

Out[62]:
```
2
```

Vishal Acharya

In [63]:
```
cse
```

Out[63]:

name	iq	marks	package
vishal	100	80	10
kavit	50	60	7
manish	120	100	14
milan	80	70	20
amit	0	10	0
tushar	0	0	0

In [64]:
```
cse.ndim
```

Out[64]:
```
2
```

In [65]:
```
ds
```

Out[65]:
```
0      6.89
1      5.12
2      7.82
3      7.42
4      6.94
       ...
195    6.93
196    5.89
197    7.21
198    7.63
199    6.22
Name: cgpa, Length: 200, dtype: float64
```

In [66]:
```
ds.ndim
```

Out[66]:
```
1
```

Vishal Acharya

shape

Return a tuple representing the dimensionality of the DataFrame.

In [67]:
```
df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
df.shape
```

Out[67]:
```
(2, 2)
```

In [68]:
```
movie.shape
```

Out[68]:
```
(4916, 28)
```

In [69]:
```
df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4],
                   'col3': [5, 6]})
df.shape
```

Out[69]:
```
(2, 3)
```

size

Return an int representing the number of elements in this object.

In [70]:
```
s = pd.Series({'a': 1, 'b': 2, 'c': 3 ,"D": None})
s.size
```

Out[70]: 4

In [71]:
```
df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
df.size
```

Out[71]: 4

In [72]:
```
movie.size
```

Out[72]: 137648

values

Return a Numpy representation of the DataFrame.

In [73]:
```
movie.values
```

Out[73]:
```
array([['Color', 'James Cameron', 723.0, ..., 7.9, 1.78, 33000],
       ['Color', 'Gore Verbinski', 302.0, ..., 7.1, 2.35, 0],
       ['Color', 'Sam Mendes', 602.0, ..., 6.8, 2.35, 85000],
       ...,
       ['Color', 'Benjamin Roberds', 13.0, ..., 6.3, nan, 16],
       ['Color', 'Daniel Hsia', 14.0, ..., 6.3, 2.35, 660],
       ['Color', 'Jon Gunn', 43.0, ..., 6.6, 1.85, 456]], dtype=object)
```

In [74]:
```
movie.dtypes.values
```

Out[74]:
```
array([dtype('O'), dtype('O'), dtype('float64'), dtype('float64'),
       dtype('float64'), dtype('float64'), dtype('O'), dtype('float64'),
       dtype('float64'), dtype('O'), dtype('O'), dtype('O'),
       dtype('int64'), dtype('int64'), dtype('O'), dtype('float64'),
       dtype('O'), dtype('O'), dtype('float64'), dtype('O'), dtype('O'),
       dtype('O'), dtype('float64'), dtype('float64'), dtype('float64'),
       dtype('float64'), dtype('float64'), dtype('int64')], dtype=object)
```

In [75]:
```
movie.iloc[:,0].value_counts()
```

Out[75]:
```
color
Color              4693
Black and White     204
Name: count, dtype: int64
```

Vishal Acharya

In [76]:
```
1 movie
```

Out[76]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	a
0	Color	James Cameron	723.0	178.0	0.0	855.0	
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	
2	Color	Sam Mendes	602.0	148.0	0.0	161.0	
3	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	
4	NaN	Doug Walker	NaN	NaN	131.0	NaN	
...	...	...	...	...	...	...	
4911	Color	Scott Smith	1.0	87.0	2.0	318.0	
4912	Color	NaN	43.0	43.0	NaN	319.0	
4913	Color	Benjamin Roberds	13.0	76.0	0.0	0.0	
4914	Color	Daniel Hsia	14.0	100.0	0.0	489.0	
4915	Color	Jon Gunn	43.0	90.0	16.0	16.0	

4916 rows × 28 columns

In [77]:
```
1 movie["director_name"]
```

Out[77]:
```
0           James Cameron
1          Gore Verbinski
2              Sam Mendes
3       Christopher Nolan
4             Doug Walker
                 ...
4911          Scott Smith
4912                  NaN
4913     Benjamin Roberds
4914          Daniel Hsia
4915             Jon Gunn
Name: director_name, Length: 4916, dtype: object
```

In [78]:
```
1 movie.loc[0:,"director_name"]
```

Out[78]:
```
0           James Cameron
1          Gore Verbinski
2              Sam Mendes
3       Christopher Nolan
4             Doug Walker
                 ...
4911          Scott Smith
4912                  NaN
4913     Benjamin Roberds
4914          Daniel Hsia
4915             Jon Gunn
Name: director_name, Length: 4916, dtype: object
```

Vishal Acharya

In [79]:
```
type(movie.loc[:,"director_name"])
```

Out[79]:
```
pandas.core.series.Series
```

In [80]:
```
movie.loc[:,"director_name"].values
```

Out[80]:
```
array(['James Cameron', 'Gore Verbinski', 'Sam Mendes', ...,
       'Benjamin Roberds', 'Daniel Hsia', 'Jon Gunn'], dtype=object)
```

series attr

In [81]:
```
ds.is_unique
```

Out[81]:
```
False
```

In [82]:
```
movie.loc[:,"director_name"].name
```

Out[82]:
```
'director_name'
```

In [83]:
```
ds.name
```

Out[83]:
```
'cgpa'
```

5. Indexing, Selecting, Filtering

- at, iat, loc, iloc

at

Access a single value for a row/column label pair.

In [84]:
```
df = pd.DataFrame([[0, 2, 3], [0, 4, 1], [10, 20, 30]],index=[4, 5, 6], columns=['A',
df
```

Out[84]:

	A	B	C
4	0	2	3
5	0	4	1
6	10	20	30

In [85]:
```
df.at[4, 'B']
```

Out[85]:
```
2
```

In [86]:
```
df.at[4, 'B'] = 10
df.at[4, 'B']
```

Out[86]:
```
10
```

Vishal Acharya

In [87]:
```
df.loc[5].at['B']
```

Out[87]: 4

In [88]:
```
movie.at[5,"director_name"]
```

Out[88]: 'Andrew Stanton'

iat

Access a single value for a row/column pair by integer position

In [89]:
```
df = pd.DataFrame([[0, 2, 3], [0, 4, 1], [10, 20, 30]],
                  columns=['A', 'B', 'C'])
df
```

Out[89]:

	A	B	C
0	0	2	3
1	0	4	1
2	10	20	30

In [90]:
```
df.iat[1, 2]
```

Out[90]: 1

In [91]:
```
df.iat[2, 2]
```

Out[91]: 30

In [92]:
```
movie.iat[5,5]
```

Out[92]: 530.0

In [94]:
```
movie.iat[4,5]
```

Out[94]: nan

Selecting rows/colums from a DataFrame

- iloc - searches using index positions
- loc - searches using index labels

iloc

Purely integer-location based indexing for selection by position.

.iloc[] is primarily integer position based (from 0 to length-1 of the axis), but may also be used with a boolean array.

Allowed inputs are:

An integer, e.g. 5.

Vishal Acharya

A list or array of integers, e.g. [4, 3, 0].

A slice object with ints, e.g. 1:7.

A boolean array.

A callable function with one argument (the calling Series or DataFrame) and that returns valid output for indexing (one of the above). This is useful in method chains, when you don't have a reference to the calling object, but would like to base your selection on some value.

A tuple of row and column indexes. The tuple elements consist of one of the above inputs, e.g. (0, 1).

In [96]:
```
mydict = [{'a': 1, 'b': 2, 'c': 3, 'd': 4},
          {'a': 100, 'b': 200, 'c': 300, 'd': 400},
          {'a': 1000, 'b': 2000, 'c': 3000, 'd': 4000 }]
df = pd.DataFrame(mydict)
df
```

Out[96]:

	a	b	c	d
0	1	2	3	4
1	100	200	300	400
2	1000	2000	3000	4000

Indexing just the rows

With a scalar integer

In [97]:
```
type(df.iloc[0])
```

Out[97]:
```
pandas.core.series.Series
```

In [98]:
```
df.iloc[0]
```

Out[98]:
```
a    1
b    2
c    3
d    4
Name: 0, dtype: int64
```

In [99]:
```
df.iloc[[0, 1]]
```

Out[99]:

	a	b	c	d
0	1	2	3	4
1	100	200	300	400

With a slice object.

Vishal Acharya

In [101]:
```
df.iloc[:3]
```

Out[101]:

	a	b	c	d
0	1	2	3	4
1	100	200	300	400
2	1000	2000	3000	4000

With a boolean mask the same length as the index.

In [103]:
```
df.iloc[[True, False, True]]
```

Out[103]:

	a	b	c	d
0	1	2	3	4
2	1000	2000	3000	4000

With a callable, useful in method chains. The x passed to the lambda is the DataFrame being sliced. This selects the rows whose index label even

In [104]:
```
df.iloc[lambda x: x.index % 2 == 0]
```

Out[104]:

	a	b	c	d
0	1	2	3	4
2	1000	2000	3000	4000

Indexing both axes

You can mix the indexer types for the index and columns. Use : to select the entire axis. With scalar integers.

In [105]:
```
df.iloc[0, 1]
```

Out[105]: 2

In [106]:
```
df.iloc[[0, 2], [1, 3]]
```

Out[106]:

	b	d
0	2	4
2	2000	4000

Vishal Acharya

In [107]:
```
df.iloc[1:3, 0:3]
```

Out[107]:

	a	b	c
1	100	200	300
2	1000	2000	3000

In [108]:
```
df.iloc[:, [True, False, True, False]]
```

Out[108]:

	a	c
0	1	3
1	100	300
2	1000	3000

In [116]:
```
movie
```

Out[116]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes
0	Color	James Cameron	723.0	178.0	0.0	855.0
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0
2	Color	Sam Mendes	602.0	148.0	0.0	161.0
3	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0
4	NaN	Doug Walker	NaN	NaN	131.0	NaN
...	...	...	...	...	...	...
4911	Color	Scott Smith	1.0	87.0	2.0	318.0
4912	Color	NaN	43.0	43.0	NaN	319.0
4913	Color	Benjamin Roberds	13.0	76.0	0.0	0.0
4914	Color	Daniel Hsia	14.0	100.0	0.0	489.0
4915	Color	Jon Gunn	43.0	90.0	16.0	16.0

4916 rows × 28 columns

In [117]:
```
movie.iloc[1:3,0:5]
```

Out[117]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes
1	Color	Gore Verbinski	302.0	169.0	563.0
2	Color	Sam Mendes	602.0	148.0	0.0

Vishal Acharya

In [118]:
```
1 movie.iloc[0:15:3,0:5]
```

Out[118]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes
0	Color	James Cameron	723.0	178.0	0.0
3	Color	Christopher Nolan	813.0	164.0	22000.0
6	Color	Sam Raimi	392.0	156.0	0.0
9	Color	David Yates	375.0	153.0	282.0
12	Color	Marc Forster	403.0	106.0	395.0

In [119]:
```
1 ds
```

Out[119]:
```
0      6.89
1      5.12
2      7.82
3      7.42
4      6.94
       ...
195    6.93
196    5.89
197    7.21
198    7.63
199    6.22
Name: cgpa, Length: 200, dtype: float64
```

In [120]:
```
1 ds.iloc[0:5]
```

Out[120]:
```
0    6.89
1    5.12
2    7.82
3    7.42
4    6.94
Name: cgpa, dtype: float64
```

Vishal Acharya

loc

Access a group of rows and columns by label(s) or a boolean array.

.loc[] is primarily label based, but may also be used with a boolean array.

Allowed inputs are:

A single label, e.g. 5 or 'a', (note that 5 is interpreted as a label of the index, and never as an integer position along the index).

A list or array of labels, e.g. ['a', 'b', 'c'].

A slice object with labels, e.g. 'a':'f'.

Warning

Note that contrary to usual python slices, both the start and the stop are included

A boolean array of the same length as the axis being sliced, e.g. [True, False, True].

An alignable boolean Series. The index of the key will be aligned before masking.

An alignable Index. The Index of the returned selection will be the input.

A callable function with one argument (the calling Series or DataFrame) and that returns valid output for indexing (one of the above)

In [121]:
```
df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
     index=['cobra', 'viper', 'sidewinder'],
     columns=['max_speed', 'shield'])
df
```

Out[121]:

	max_speed	shield
cobra	1	2
viper	4	5
sidewinder	7	8

Single label. Note this returns the row as a Series.

In [123]:
```
df.loc['viper']
```

Out[123]:
```
max_speed    4
shield       5
Name: viper, dtype: int64
```

List of labels. Note using [[]] returns a DataFrame.

In [125]:
```
df.loc[['viper', 'sidewinder']]
```

Out[125]:

	max_speed	shield
viper	4	5
sidewinder	7	8

In [126]:
```
df.loc['cobra', 'shield']
```

Out[126]:
```
2
```

In [127]:
```
df.loc['cobra':'viper', 'max_speed']
```

Out[127]:
```
cobra    1
viper    4
Name: max_speed, dtype: int64
```

Boolean list with the same length as the row axis

In [129]:
```
df.loc[[False, False, True]]
```

Out[129]:

	max_speed	shield
sidewinder	7	8

Vishal Acharya

In [130]:
```
df.loc[pd.Series([False, True, False],index=['viper', 'sidewinder', 'cobra'])]
```

Out[130]:

	max_speed	shield
sidewinder	7	8

In [131]:
```
df.loc[pd.Index(["cobra", "viper"], name="foo")]
```

Out[131]:

foo	max_speed	shield
cobra	1	2
viper	4	5

Conditional that returns a boolean Series¶

In [132]:
```
df.loc[df['shield'] > 6]
```

Out[132]:

	max_speed	shield
sidewinder	7	8

Conditional that returns a boolean Series with column labels specified

In [133]:
```
df.loc[df['shield'] > 6, ['max_speed']]
```

Out[133]:

	max_speed
sidewinder	7

Callable that returns a boolean Series

In [134]:
```
df.loc[lambda df: df['shield'] == 8]
```

Out[134]:

	max_speed	shield
sidewinder	7	8

Setting values

Set value for all items matching the list of labels

Vishal Acharya

In [135]:
```
df.loc[['viper', 'sidewinder'], ['shield']] = 50
df
```

Out[135]:

	max_speed	shield
cobra	1	2
viper	4	50
sidewinder	7	50

Set value for an entire row¶

In [136]:
```
df.loc['cobra'] = 10
df
```

Out[136]:

	max_speed	shield
cobra	10	10
viper	4	50
sidewinder	7	50

Set value for rows matching callable condition

In [137]:
```
df.loc[df['shield'] > 35] = 0
df
```

Out[137]:

	max_speed	shield
cobra	10	10
viper	0	0
sidewinder	0	0

Getting values on a DataFrame with an index that has integer labels Another example using integers for the index

In [138]:
```
df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
     index=[7, 8, 9], columns=['max_speed', 'shield'])
df
```

Out[138]:

	max_speed	shield
7	1	2
8	4	5
9	7	8

Slice with integer labels for rows. As mentioned above, note that both the start and stop of the slice are

Vishal Acharya

included¶

In [139]:
```
df.loc[7:9]
```

Out[139]:

	max_speed	shield
7	1	2
8	4	5
9	7	8

Getting values with a MultiIndex

A number of examples using a DataFrame with a MultiIndex

In [141]:
```
tuples = [
   ('cobra', 'mark i'), ('cobra', 'mark ii'),
   ('sidewinder', 'mark i'), ('sidewinder', 'mark ii'),
   ('viper', 'mark ii'), ('viper', 'mark iii')
]
index = pd.MultiIndex.from_tuples(tuples)
values = [[12, 2], [0, 4], [10, 20],
        [1, 4], [7, 1], [16, 36]]
df = pd.DataFrame(values, columns=['max_speed', 'shield'], index=index)
df
```

Out[141]:

		max_speed	shield
cobra	**mark i**	12	2
	mark ii	0	4
sidewinder	**mark i**	10	20
	mark ii	1	4
viper	**mark ii**	7	1
	mark iii	16	36

In [142]:
```
df.loc['cobra']
```

Out[142]:

	max_speed	shield
mark i	12	2
mark ii	0	4

In [143]:
```
df.loc[('cobra', 'mark ii')]
```

Out[143]:
```
max_speed    0
shield       4
Name: (cobra, mark ii), dtype: int64
```

Vishal Acharya

In [144]:
```
df.loc[('cobra', 'mark i'):('viper', 'mark ii')]
```

Out[144]:

		max_speed	shield
cobra	**mark i**	12	2
	mark ii	0	4
sidewinder	**mark i**	10	20
	mark ii	1	4
viper	**mark ii**	7	1

Summarizing and Computing Descriptive Statistics(Unlike NumPy arrays, Pandas descriptive statistics automatically exclude missing data. NaN values are excluded unless the entire row or column is NA)

- head(), tail(), info(),describe()

The head() function is a built-in function in pandas for the dataframe used to display the rows of the dataset. We can specify the number of rows by giving the number within the parenthesis. By default, it displays the first five rows of the dataset. If we want to see the last five rows of the dataset, we use the tail()function of the dataframe

In [146]:
```
movie.head()
```

Out[146]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	acto
0	Color	James Cameron	723.0	178.0	0.0	855.0	
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orla
2	Color	Sam Mendes	602.0	148.0	0.0	161.0	Rc
3	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	Ch
4	NaN	Doug Walker	NaN	NaN	131.0	NaN	F

5 rows × 28 columns

Vishal Acharya

In [147]:
```
1 movie.head(7)
```

Out[147]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	acto
0	Color	James Cameron	723.0	178.0	0.0	855.0	
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orla
2	Color	Sam Mendes	602.0	148.0	0.0	161.0	Rc
3	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	Ch
4	NaN	Doug Walker	NaN	NaN	131.0	NaN	F
5	Color	Andrew Stanton	462.0	132.0	475.0	530.0	
6	Color	Sam Raimi	392.0	156.0	0.0	4000.0	Jan

7 rows × 28 columns

In [148]:
```
1 movie.tail()
```

Out[148]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	a
4911	Color	Scott Smith	1.0	87.0	2.0	318.0	
4912	Color	NaN	43.0	43.0	NaN	319.0	
4913	Color	Benjamin Roberds	13.0	76.0	0.0	0.0	
4914	Color	Daniel Hsia	14.0	100.0	0.0	489.0	
4915	Color	Jon Gunn	43.0	90.0	16.0	16.0	

5 rows × 28 columns

In [149]:
```
1 movie.tail(3)
```

Out[149]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	a
4913	Color	Benjamin Roberds	13.0	76.0	0.0	0.0	
4914	Color	Daniel Hsia	14.0	100.0	0.0	489.0	
4915	Color	Jon Gunn	43.0	90.0	16.0	16.0	

3 rows × 28 columns

Vishal Acharya

In [150]:
```
1 movie.sample(5)
```

Out[150]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	a
1089	Color	Ron Shelton	48.0	135.0	41.0	808.0	
4745	Color	Bill Plympton	19.0	75.0	45.0	0.0	
2365	Color	Larry Charles	343.0	82.0	119.0	6.0	
806	Color	Rob Letterman	218.0	103.0	11.0	543.0	
3437	Color	Phillip Noyce	74.0	94.0	176.0	46.0	

5 rows × 28 columns

df.info()

Print a concise summary of a DataFrame. This method prints information about a DataFrame including the index dtype and columns, non-null values and memory usage.

In [151]:
```
1 movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4916 entries, 0 to 4915
Data columns (total 28 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   color                      4897 non-null   object 
 1   director_name              4814 non-null   object 
 2   num_critic_for_reviews     4867 non-null   float64
 3   duration                   4901 non-null   float64
 4   director_facebook_likes    4814 non-null   float64
 5   actor_3_facebook_likes     4893 non-null   float64
 6   actor_2_name               4903 non-null   object 
 7   actor_1_facebook_likes     4909 non-null   float64
 8   gross                      4054 non-null   float64
 9   genres                     4916 non-null   object 
 10  actor_1_name               4909 non-null   object 
 11  movie_title                4916 non-null   object 
 12  num_voted_users            4916 non-null   int64  
 13  cast_total_facebook_likes  4916 non-null   int64  
 14  actor_3_name               4893 non-null   object 
 15  facenumber_in_poster       4903 non-null   float64
 16  plot_keywords              4764 non-null   object 
 17  movie_imdb_link            4916 non-null   object 
 18  num_user_for_reviews       4895 non-null   float64
 19  language                   4902 non-null   object 
 20  country                    4911 non-null   object 
 21  content_rating             4616 non-null   object 
 22  budget                     4432 non-null   float64
 23  title_year                 4810 non-null   float64
 24  actor_2_facebook_likes     4903 non-null   float64
 25  imdb_score                 4916 non-null   float64
 26  aspect_ratio               4590 non-null   float64
 27  movie_facebook_likes       4916 non-null   int64  
dtypes: float64(13), int64(3), object(12)
memory usage: 1.1+ MB
```

Vishal Acharya

df.describe(percentiles=None, include=None, exclude=None)

Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values. Analyzes both numeric and object series, as well as DataFrame column sets of mixed data types. The output will vary depending on what is provided.

percentileslist-like of numbers, optional The percentiles to include in the output. All should fall between 0 and 1. The default is [.25, .5, .75], which returns the 25th, 50th, and 75th percentiles.

include'all', list-like of dtypes or None (default), optional

A white list of data types to include in the result. Ignored for Series. Here are the options:

'all' : All columns of the input will be included in the output.

A list-like of dtypes : Limits the results to the provided data types. To limit the result to numeric types submit numpy.number. To limit it instead to object columns submit the numpy.object data type. Strings can also be used in the style of select_dtypes (e.g. df.describe(include=['O'])). To select pandas categorical columns, use 'category'

None (default) : The result will include all numeric columns. excludelist-like of dtypes or None (default), optional,

A black list of data types to omit from the result. Ignored for Series. Here are the options:

A list-like of dtypes : Excludes the provided data types from the result. To exclude numeric types submit numpy.number. To exclude object columns submit the data type numpy.object. Strings can also be used in the style of select_dtypes (e.g. df.describe(exclude=['O'])). To exclude pandas categorical columns, use 'category'

None (default) : The result will exclude nothing.

Vishal Acharya

In [152]:
```
1 movie.describe()
```

Out[152]:

facebook_likes	facenumber_in_poster	num_user_for_reviews	budget	title_year	actor_2_facebook_likes
4916.000000	4903.000000	4895.000000	4.432000e+03	4810.000000	4903.000000
9579.815907	1.377320	267.668846	3.654749e+07	2002.447609	1621.923516
18164.316990	2.023826	372.934839	1.002427e+08	12.453977	4011.299523
0.000000	0.000000	1.000000	2.180000e+02	1916.000000	0.000000
1394.750000	0.000000	64.000000	6.000000e+06	1999.000000	277.000000
3049.000000	1.000000	153.000000	1.985000e+07	2005.000000	593.000000
13616.750000	2.000000	320.500000	4.300000e+07	2011.000000	912.000000
656730.000000	43.000000	5060.000000	4.200000e+09	2016.000000	137000.000000

In [153]:
```
1 movie.describe(include="all")
```

Out[153]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_li
count	4897	4814	4867.000000	4901.000000	4814.000000	4893.000
unique	2	2397	NaN	NaN	NaN	N
top	Color	Steven Spielberg	NaN	NaN	NaN	N
freq	4693	26	NaN	NaN	NaN	N
mean	NaN	NaN	137.988905	107.090798	691.014541	631.276
std	NaN	NaN	120.239379	25.286015	2832.954125	1625.874
min	NaN	NaN	1.000000	7.000000	0.000000	0.000
25%	NaN	NaN	49.000000	93.000000	7.000000	132.000
50%	NaN	NaN	108.000000	103.000000	48.000000	366.000
75%	NaN	NaN	191.000000	118.000000	189.750000	633.000
max	NaN	NaN	813.000000	511.000000	23000.000000	23000.000

11 rows × 28 columns

In [158]:
```
1 movie.describe(include="object")
```

Out[158]:

	color	director_name	actor_2_name	genres	actor_1_name	movie_title	actor_3_name	plot_keywords
count	4897	4814	4903	4916	4909	4916	4893	4764
unique	2	2397	3030	914	2095	4916	3519	4756
top	Color	Steven Spielberg	Morgan Freeman	Drama	Robert De Niro	Avatar	Steve Coogan	based on novel
freq	4693	26	18	233	48	1	8	4

In [159]:
```
1 movie.describe(include="number")
```

Out[159]:

	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_1_facebook_l
count	4867.000000	4901.000000	4814.000000	4893.000000	4909.00
mean	137.988905	107.090798	691.014541	631.276313	6494.48
std	120.239379	25.286015	2832.954125	1625.874802	15106.98
min	1.000000	7.000000	0.000000	0.000000	0.00
25%	49.000000	93.000000	7.000000	132.000000	607.00
50%	108.000000	103.000000	48.000000	366.000000	982.00
75%	191.000000	118.000000	189.750000	633.000000	11000.00
max	813.000000	511.000000	23000.000000	23000.000000	640000.00

Vishal Acharya

In [161]:
```
movie.describe(exclude='category')
```

Out[161]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_li
count	4897	4814	4867.000000	4901.000000	4814.000000	4893.000
unique	2	2397	NaN	NaN	NaN	N
top	Color	Steven Spielberg	NaN	NaN	NaN	N
freq	4693	26	NaN	NaN	NaN	N
mean	NaN	NaN	137.988905	107.090798	691.014541	631.276
std	NaN	NaN	120.239379	25.286015	2832.954125	1625.874
min	NaN	NaN	1.000000	7.000000	0.000000	0.000
25%	NaN	NaN	49.000000	93.000000	7.000000	132.000
50%	NaN	NaN	108.000000	103.000000	48.000000	366.000
75%	NaN	NaN	191.000000	118.000000	189.750000	633.000
max	NaN	NaN	813.000000	511.000000	23000.000000	23000.000

11 rows × 28 columns

In [162]:
```
movie.describe(exclude='number')
```

Out[162]:

	color	director_name	actor_2_name	genres	actor_1_name	movie_title	actor_3_name	plot_keywords
count	4897	4814	4903	4916	4909	4916	4893	4764
unique	2	2397	3030	914	2095	4916	3519	4756
top	Color	Steven Spielberg	Morgan Freeman	Drama	Robert De Niro	Avatar	Steve Coogan	based on novel
freq	4693	26	18	233	48	1	8	4

In [164]:
```
movie.describe(percentiles=[0.20,0.40])
```

Out[164]:

	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_1_facebook_l
count	4867.000000	4901.000000	4814.000000	4893.000000	4909.00
mean	137.988905	107.090798	691.014541	631.276313	6494.48
std	120.239379	25.286015	2832.954125	1625.874802	15106.98
min	1.000000	7.000000	0.000000	0.000000	0.00
20%	39.000000	91.000000	3.000000	94.400000	510.00
40%	82.000000	99.000000	25.000000	260.000000	854.00
50%	108.000000	103.000000	48.000000	366.000000	982.00
max	813.000000	511.000000	23000.000000	23000.000000	640000.00

Vishal Acharya

3.Handle missing data: Identify any missing data in the DataFrame using the isna() function and handle it appropriately using the fillna() or dropna() function.

- find missing value
- fill the value or drop the value

find the missing value

- Using isnull() function: This function provides the boolean value for the complete dataset to know if any null value is present or not. Using isna() function:

df.isnull()

- isna function: This is the same as the isnull() function. Ans provides the same output.

df.isna()

- Using isna().any() his function also gives a boolean value if any null value is present or not, but it gives results column-wise, not in tabular format.

df.isna().any()

- Using isna(). sum(): This function gives the sum of the null values preset in the dataset column-wise.

df.isna().sum()

- Using isna().any().sum():This function gives output in a single value if any null is present or not.

data.isna().any().sum()

- Dropping Missing Data in a Pandas DataFrame When working with missing data, it's often good to do one of two things: either drop the records or find ways to fill the data. In this section, you'll learn how to take on the former of the two. Pandas provides a method, .dropna(), which is used to drop missing data. Exploring the Pandas .dropna() method

df.dropna(

axis=0, # Whether to drop rows or columns

how='any', # Whether to drop records if 'all' or 'any' records are missing

thresh=None, # How many columns/rows must be missing to drop

subset=None, # Which rows/columns to consider

inplace=False # Whether to drop in place (i.e., without needing to re- assign)

)

- The drop() method removes the specified row or column.

Vishal Acharya

Vishal Acharya

df.drop(labels, axis, index, columns, level, inplace., errors)

Parameter	Value	Description
labels		Optional, The labels or indexes to drop. If more than one, specify them in a list.

axis	`0`
	`1`
	`'index'`
	`'columns'`
		Optional, Which axis to check, default 0.

index	*String* *List*	Optional, Specifies the name of the rows to drop. Can be used instead of the `labels` parameter.
columns	*String* *List*	Optional, Specifies the name of the columns to drop. Can be used instead of the `labels` parameter.
level	*Number* *level name*	Optional, default None. Specifies which level (in a hierarchical multi index) to check along
inplace	`True` `False`	Optional, default False. If True: the removing is done on the current DataFrame. If False: returns a copy where the removing is done.
errors	`'ignore'` `'raise'`	Optional, default 'ignore'. Specifies whether to ignore errors or not

- The fillna() method replaces the NULL values with a specified value.
- The fillna() method returns a new DataFrame object unless the inplace parameter is set to True, in that case the fillna() method does the replacing in the original DataFrame instead.

df.fillna(value, method, axis, inplace, limit, downcast)

Parameter	Value	Description
value	*Number* *String* *Dictionary* *Series* *DataFrame*	Required, Specifies the value to replace the NULL values with. This can also be values for the entire row or column.

method	`'backfill'`
	`'bfill'`
	`'pad'`
	`'ffill'`
	`None`
		Optional, default None'. Specifies the method to use when replacing

axis	`0`
	`1`
	`'index'`
	`'columns'`
		Optional, default 0. The axis to fill the NULL values along

inplace	`True` `False`	Optional, default False. If True: the replacing is done on the current DataFrame. If False: returns a copy where the replacing is done.
limit	*Number* `None`	Optional, default None. Specifies the maximum number of NULL values to fill (if method is specified)
downcast	*Dictionary* `None`	Optional, a dictionary of values to fill for specific data types

image-2.png

- Different ways to fill the missing values

1. Mean/Median, Mode
2. bfill,ffill
3. interpolate
4. Mean/Median, Mode

- Numerical Data →Mean/Median
- Categorical Data →Mode

In columns having numerical data, we can fill the missing values by mean/median.

Mean — When the data has no outliers. Mean is the average value. Mean will be affected by outliers.

[Example. If we are calculating, mean salary of the employees in a room and if the company CEO walks in, the mean will tend to be higher. It can't be the representative amount. In that scenario, we can choose median]

Median — When the data has more outliers, it's best to replace them with the median value. Median is the middle value (50%)

In columns having categorical data, we can fill the missing values by mode

Mode — Most common value.

In [165]:
```
1 movie.isnull()
```

Out[165]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	a
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	
3	False	False	False	False	False	False	
4	True	False	True	True	False	True	
...	...	...	...	...	...	...	
4911	False	False	False	False	False	False	
4912	False	True	False	False	True	False	
4913	False	False	False	False	False	False	
4914	False	False	False	False	False	False	
4915	False	False	False	False	False	False	

4916 rows × 28 columns

Vishal Acharya

In [166]:
```
1 movie.isnull().sum()
```

Out[166]:
```
color                      19
director_name             102
num_critic_for_reviews     49
duration                   15
director_facebook_likes   102
actor_3_facebook_likes     23
actor_2_name               13
actor_1_facebook_likes      7
gross                     862
genres                      0
actor_1_name                7
movie_title                 0
num_voted_users             0
cast_total_facebook_likes   0
actor_3_name               23
facenumber_in_poster       13
plot_keywords             152
movie_imdb_link             0
num_user_for_reviews       21
language                   14
country                     5
content_rating            300
budget                    484
title_year                106
actor_2_facebook_likes     13
imdb_score                  0
aspect_ratio              326
movie_facebook_likes        0
dtype: int64
```

In [167]:
```
1 movie.isna()
```

Out[167]:

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	a
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	
3	False	False	False	False	False	False	
4	True	False	True	True	False	True	
...	...	...	...	...	...	...	
4911	False	False	False	False	False	False	
4912	False	True	False	False	True	False	
4913	False	False	False	False	False	False	
4914	False	False	False	False	False	False	
4915	False	False	False	False	False	False	

4916 rows × 28 columns

Vishal Acharya

In [169]:
```
1 movie.isna().sum()
```

Out[169]:
```
color                      19
director_name             102
num_critic_for_reviews     49
duration                   15
director_facebook_likes   102
actor_3_facebook_likes     23
actor_2_name               13
actor_1_facebook_likes      7
gross                     862
genres                      0
actor_1_name                7
movie_title                 0
num_voted_users             0
cast_total_facebook_likes   0
actor_3_name               23
facenumber_in_poster       13
plot_keywords             152
movie_imdb_link             0
num_user_for_reviews       21
language                   14
country                     5
content_rating            300
budget                    484
title_year                106
actor_2_facebook_likes     13
imdb_score                  0
aspect_ratio              326
movie_facebook_likes        0
dtype: int64
```

In [170]:
```
1 movie.isna().any()
```

Out[170]:
```
color                      True
director_name              True
num_critic_for_reviews     True
duration                   True
director_facebook_likes    True
actor_3_facebook_likes     True
actor_2_name               True
actor_1_facebook_likes     True
gross                      True
genres                    False
actor_1_name               True
movie_title               False
num_voted_users           False
cast_total_facebook_likes False
actor_3_name               True
facenumber_in_poster       True
plot_keywords              True
movie_imdb_link           False
num_user_for_reviews       True
language                   True
country                    True
content_rating             True
budget                     True
title_year                 True
actor_2_facebook_likes     True
imdb_score                False
aspect_ratio               True
movie_facebook_likes      False
dtype: bool
```

Vishal Acharya

in pandas, the fillna method is used to fill missing or NaN values in a DataFrame or Series.

The ffill and bfill options of the fillna method can be used to forward fill or backward fill missing values respectively.¶

In [171]:
```
import pandas as pd
import numpy as np
# create a sample dataframe with missing values
df = pd.DataFrame({'A': [1, 2, np.nan, 4, np.nan],
'B': [np.nan, 6, 7, np.nan, 9],
'C': [10, 11, np.nan, np.nan, 14]})
print(df)
# forward fill missing values
df_ffill = df.fillna(method='ffill')
print("DataFrame after forward filling:")
print(df_ffill)
# backward fill missing values
df_bfill = df.fillna(method='bfill')
print("\nDataFrame after backward filling:")
print(df_bfill)
```

```
     A    B     C
0  1.0  NaN  10.0
1  2.0  6.0  11.0
2  NaN  7.0   NaN
3  4.0  NaN   NaN
4  NaN  9.0  14.0
DataFrame after forward filling:
     A    B     C
0  1.0  NaN  10.0
1  2.0  6.0  11.0
2  2.0  7.0  11.0
3  4.0  7.0  11.0
4  4.0  9.0  14.0

DataFrame after backward filling:
     A    B     C
0  1.0  6.0  10.0
1  2.0  6.0  11.0
2  4.0  7.0  14.0
3  4.0  9.0  14.0
4  NaN  9.0  14.0

C:\Users\VISHAL\AppData\Local\Temp\ipykernel_13304\1489739334.py:9: FutureWarning: DataF
rame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffil
l() or obj.bfill() instead.
  df_ffill = df.fillna(method='ffill')
C:\Users\VISHAL\AppData\Local\Temp\ipykernel_13304\1489739334.py:13: FutureWarning: Data
Frame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffi
ll() or obj.bfill() instead.
  df_bfill = df.fillna(method='bfill')
```

Vishal Acharya

In pandas, the fillna method with the interpolate option can be used to fill missing values using interpolation. Interpolation is a method of estimating missing values by using the values of adjacent data points.

In [172]:

```
import pandas as pd
import numpy as np
# create a sample dataframe with missing values
df = pd.DataFrame({'A': [1, 2, np.nan, 4, np.nan],
'B': [np.nan, 6, 7, np.nan, 9],
'C': [10, 11, np.nan, np.nan, 14]})
# interpolate missing values
df_interpolate = df.interpolate()
print("DataFrame after interpolating missing values:")
print(df_interpolate)
```

```
DataFrame after interpolating missing values:
     A    B     C
0  1.0  NaN  10.0
1  2.0  6.0  11.0
2  3.0  7.0  12.0
3  4.0  8.0  13.0
4  4.0  9.0  14.0
```

Vishal Acharya

In [179]:

```
df = pd.DataFrame({'A': [1, 2, np.nan, 4, np.nan],
'B': [np.nan, 6, 7, np.nan, 9],
'C': ["10", "11", np.nan, np.nan, "14"]})
print(df)
# fill with the mean,median,mode
print("mean")
df["A"]=df["A"].fillna(df["A"].mean())
print(df)
print("median")
df["B"]=df["B"].fillna(df["B"].median())
print(df)
print("mode")
df["C"]=df["C"].fillna(df["C"].mode())
print(df)

```

```
     A    B    C
0  1.0  NaN   10
1  2.0  6.0   11
2  NaN  7.0  NaN
3  4.0  NaN  NaN
4  NaN  9.0   14
mean
          A    B    C
0  1.000000  NaN   10
1  2.000000  6.0   11
2  2.333333  7.0  NaN
3  4.000000  NaN  NaN
4  2.333333  9.0   14
median
          A    B    C
0  1.000000  7.0   10
1  2.000000  6.0   11
2  2.333333  7.0  NaN
3  4.000000  7.0  NaN
4  2.333333  9.0   14
mode
          A    B    C
0  1.000000  7.0   10
1  2.000000  6.0   11
2  2.333333  7.0   14
3  4.000000  7.0  NaN
4  2.333333  9.0   14
```

Vishal Acharya

in pandas, the dropna method is used to remove missing or NaN values from a DataFrame or Series. There are several options available in dropna method to control the behavior of dropping missing values.

- Drop rows with missing values

In [180]:

```
import pandas as pd
import numpy as np
# create a sample dataframe with missing values
df = pd.DataFrame({'A': [1, 2, np.nan, 4, np.nan],
'B': [np.nan, 6, 7, np.nan, 9],
'C': [10, 11, np.nan, np.nan, 14]})
print(df)
# drop rows with missing values
df_dropped = df.dropna()
print("DataFrame after dropping rows with missing values:")
print(df_dropped)
```

```
     A    B     C
0  1.0  NaN  10.0
1  2.0  6.0  11.0
2  NaN  7.0   NaN
3  4.0  NaN   NaN
4  NaN  9.0  14.0
DataFrame after dropping rows with missing values:
     A    B     C
1  2.0  6.0  11.0
```

- Drop columns with missing values

In [181]:

```
import pandas as pd
import numpy as np
# create a sample dataframe with missing values
df = pd.DataFrame({'A': [1, 2, np.nan, 4, np.nan],
'B': [np.nan, 6, 7, np.nan, 9],
'C': [10, 11, np.nan, np.nan, 14]})
print(df)
# drop columns with missing values
df_dropped = df.dropna(axis=1)
print("DataFrame after dropping columns with missing values:")
print(df_dropped)
```

```
     A    B     C
0  1.0  NaN  10.0
1  2.0  6.0  11.0
2  NaN  7.0   NaN
3  4.0  NaN   NaN
4  NaN  9.0  14.0
DataFrame after dropping columns with missing values:
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4]
```

Vishal Acharya

Drop rows with missing values only in specific columns

In [182]:
```
# import pandas as pd
import numpy as np
# create a sample dataframe with missing values
df = pd.DataFrame({'A': [1, 2, np.nan, 4, np.nan],
'B': [np.nan, 6, 7, np.nan, 9],
'C': [10, 11, np.nan, np.nan, 14]})
# drop rows with missing values in column B only
df_dropped = df.dropna(subset=['B'])
print("DataFrame after dropping rows with missing values in column B only:")
print(df_dropped)
```

```
DataFrame after dropping rows with missing values in column B only:
     A    B     C
1  2.0  6.0  11.0
2  NaN  7.0   NaN
4  NaN  9.0  14.0
```

Vishal Acharya

the thresh parameter with dropna method to drop rows with less than 2 non-missing values

In [183]:
```
import pandas as pd
import numpy as np
# create a sample dataframe with missing values
df = pd.DataFrame({'A': [1, 2, np.nan, 4, np.nan],
'B': [np.nan, 6, 7, np.nan, 9],
'C': [10, 11, np.nan, np.nan, 14]})
# drop rows with less than 2 non-missing values
df_dropped = df.dropna(thresh=2)
print("DataFrame after dropping rows with less than 2 non-missing values:")
print(df_dropped)
```

```
DataFrame after dropping rows with less than 2 non-missing values:
     A    B     C
0  1.0  NaN  10.0
1  2.0  6.0  11.0
4  NaN  9.0  14.0
```

use drop for removing specify col/row

In [190]:
```
import pandas as pd
import numpy as np
# create a sample dataframe with missing values
df = pd.DataFrame({'A': [1, 2, np.nan, 4, np.nan],
'B': [np.nan, 6, 7, np.nan, 9],
'C': [10, 11, np.nan, np.nan, 14]})

df_dropped = df.drop(labels="A" ,axis=1)

print(df_dropped)
```

```
     B     C
0  NaN  10.0
1  6.0  11.0
2  7.0   NaN
3  NaN   NaN
4  9.0  14.0
```

In [192]:

```
import pandas as pd
import numpy as np
# create a sample dataframe with missing values
df = pd.DataFrame({'A': [1, 2, np.nan, 4, np.nan],
'B': [np.nan, 6, 7, np.nan, 9],
'C': [10, 11, np.nan, np.nan, 14]})

df_dropped = df.drop(2 ,axis=0)

print(df_dropped)
```

```
     A    B     C
0  1.0  NaN  10.0
1  2.0  6.0  11.0
3  4.0  NaN   NaN
4  NaN  9.0  14.0
```

Vishal Acharya

Handle duplicates: Identify and remove any duplicate records in the DataFrame using the duplicated() and drop_duplicates() functions.

Identifying Duplicate Records in a Pandas DataFrame

Pandas provides a helpful method, .duplicated(), which allows you to identify duplicate records in a dataset. The method, similar to the .isnull() method you learned above, returns boolean values when duplicate records exist. This method returns a single Series if records are duplicated:

- print(df.duplicated()) Counting Duplicate Records in a DataFrame
- print(df.duplicated().sum()) Removing Duplicate Data in a Pandas DataFrame

Pandas makes it easy to remove duplicate records using the .drop_duplicates() method. Let's take a look at what parameters the method has available:

- The Pandas .drop_duplicates() method

df.drop_duplicates(

subset=None, # Which columns to consider

keep='first', # Which duplicate record to keep

inplace=False, # Whether to drop in place

ignore_index=False # Whether to relabel the index)

The drop_duplicates method in pandas can be used to remove duplicate rows from a DataFrame or Series. It can be used with various options to specify which columns to consider for duplicates, which row to keep, and whether to keep the first or last occurrence of a duplicate row.

- Removing duplicate rows based on all columns

In [194]:

```
import pandas as pd
# create a sample dataframe with duplicate rows
df = pd.DataFrame({'A': [1, 1, 2, 3, 4, 4],
'B': [5, 5, 6, 7, 8, 8]})
print(df)
print(" Counting Duplicate Records in a DataFrame")
print(df.duplicated())
print(df.duplicated().sum())
# drop duplicate rows based on all columns
df_dropped = df.drop_duplicates()
print("DataFrame after dropping all duplicate rows:")
print(df_dropped)
```

```
   A  B
0  1  5
1  1  5
2  2  6
3  3  7
4  4  8
5  4  8
 Counting Duplicate Records in a DataFrame
0    False
1     True
2    False
3    False
4    False
5     True
dtype: bool
2
DataFrame after dropping all duplicate rows:
   A  B
0  1  5
2  2  6
3  3  7
4  4  8
```

Removing duplicate rows based on a subset of columns

In [195]:

```
import pandas as pd
# create a sample dataframe with duplicate rows
df = pd.DataFrame({'A': [1, 1, 2, 3, 4, 4],
'B': [5, 5, 6, 7, 8, 8],
'C': ['x', 'x', 'y', 'z', 'w', 'w']})
print(df)
# drop duplicate rows based on a subset of columns
df_dropped = df.drop_duplicates(subset=['A', 'B'])
print("DataFrame after dropping duplicate rows based on columns A and B:")
print(df_dropped)
```

```
   A  B  C
0  1  5  x
1  1  5  x
2  2  6  y
3  3  7  z
4  4  8  w
5  4  8  w
DataFrame after dropping duplicate rows based on columns A and B:
   A  B  C
0  1  5  x
2  2  6  y
3  3  7  z
4  4  8  w
```

Vishal Acharya

Keeping the first or last occurrence of a duplicate row:

In [196]:
```
import pandas as pd
# create a sample dataframe with duplicate rows
df = pd.DataFrame({'A': [1, 1, 2, 3, 4, 4],
'B': [5, 5, 6, 7, 8, 8],
'C': ['x', 'x', 'y', 'z', 'w', 'w']})
print(df)
# drop duplicate rows based on a subset of columns
df_dropped = df.drop_duplicates(subset=['A', 'B'],keep="last")
print("DataFrame after dropping duplicate rows based on columns A and B:")
print(df_dropped)
```

```
   A  B  C
0  1  5  x
1  1  5  x
2  2  6  y
3  3  7  z
4  4  8  w
5  4  8  w
DataFrame after dropping duplicate rows based on columns A and B:
   A  B  C
1  1  5  x
2  2  6  y
3  3  7  z
5  4  8  w
```

Vishal Acharya

how to use the drop method to remove a row from a DataFrame

In [197]:
```
import pandas as pd

data = {'name': ['John', 'Emily', 'Michael', 'Jessica'],
        'age': [23, 25, 32, 28],
        'country': ['USA', 'Canada', 'UK', 'Australia']}

df = pd.DataFrame(data)

# remove the second row (index 1)
new_df = df.drop(1)

print(df)
print("*"*50)
print(new_df)


```

```
      name  age    country
0     John   23        USA
1    Emily   25     Canada
2  Michael   32         UK
3  Jessica   28  Australia
**************************************************
      name  age    country
0     John   23        USA
2  Michael   32         UK
3  Jessica   28  Australia
```

how to use the drop method to remove a column from a DataFrame

In [274]:
```
import pandas as pd

data = {'name': ['John', 'Emily', 'Michael', 'Jessica'],
        'age': [23, 25, 32, 28],
        'country': ['USA', 'Canada', 'UK', 'Australia']}

df = pd.DataFrame(data)

# remove the 'country' column
new_df = df.drop('country', axis=1)

print(df)
print("*"*50)
print(new_df)

```

```
      name  age    country
0     John   23        USA
1    Emily   25     Canada
2  Michael   32         UK
3  Jessica   28  Australia
**************************************************
      name  age
0     John   23
1    Emily   25
2  Michael   32
3  Jessica   28
```

Vishal Acharya

Handle inconsistent data: Identify and correct any inconsistent data in the DataFrame using the replace() function or by manually changing the data.

- Non-standard Missing Values: Sometimes missing values will be entered like .., __, — , missing, na, @,??,***, etc.
- Unexpected Missing Values: example in marksheet the mark value grater than 100 is unexpectd
- this type value find df["column_name"].is_unique

Replace Missing Values

- df.replace(old_value, new_value) → old_value will be replaced by new_value¶

In [204]:
```
df
```

Out[204]:

	name	age	country
0	John	23	USA
1	Emily	25	Canada
2	Michael	32	UK
3	Jessica	28	Australia

In [205]:
```
df.nunique()
```

Out[205]:
```
name       4
age        4
country    4
dtype: int64
```

In [206]:
```
df.values
```

Out[206]:
```
array([['John', 23, 'USA'],
       ['Emily', 25, 'Canada'],
       ['Michael', 32, 'UK'],
       ['Jessica', 28, 'Australia']], dtype=object)
```

In [207]:
```
df.replace('UK', 'VIRGINIA', inplace=True)
df
```

Out[207]:

	name	age	country
0	John	23	USA
1	Emily	25	Canada
2	Michael	32	VIRGINIA
3	Jessica	28	Australia

In [208]:
```
df = df.set_index('country')
df.head()
```

Out[208]:

country	name	age
USA	John	23
Canada	Emily	25
VIRGINIA	Michael	32
Australia	Jessica	28

In [209]:
```
#Renaming column of a DataFrame
new_col = {'name':'listing_name', 'age':'reviews'}

df.rename(columns=new_col, inplace=True)
df.head()
```

Out[209]:

country	listing_name	reviews
USA	John	23
Canada	Emily	25
VIRGINIA	Michael	32
Australia	Jessica	28

Vishal Acharya

In [210]:
```
#Converting data type to reduce memory
df['reviews'] = df["reviews"].astype('int8')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, USA to Australia
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   listing_name  4 non-null      object
 1   reviews       4 non-null      int8
dtypes: int8(1), object(1)
memory usage: 68.0+ bytes
```

pandas.crosstab(index, columns, values=None, rownames=None, colnames=None, aggfunc=None, margins=False, margins_name='All', dropna=True, normalize=False)[source]

- Compute a simple cross tabulation of two (or more) factors.
- By default, computes a frequency table of the factors unless an array of values and an aggregation function are passed.
- Parameters indexarray-like, Series, or list of arrays/Series Values to group by in the rows.

columnsarray-like, Series, or list of arrays/Series Values to group by in the columns.

valuesarray-like, optional Array of values to aggregate according to the factors. Requires aggfunc be specified.

rownamessequence, default None If passed, must match number of row arrays passed.

colnamessequence, default None If passed, must match number of column arrays passed.

aggfuncfunction, optional If specified, requires values be specified as well.

marginsbool, default False Add row/column margins (subtotals).

margins_namestr, default ‘All’ Name of the row/column that will contain the totals when margins is True.

dropnabool, default True Do not include columns whose entries are all NaN.

normalizebool, {‘all’, ‘index’, ‘columns’}, or {0,1}, default False Normalize by dividing all values by the sum of values.

If passed ‘all’ or True, will normalize over all values.

If passed ‘index’ will normalize over each row.

If passed ‘columns’ will normalize over each column.

If margins is True, will also normalize margin values.

Vishal Acharya

In [294]:
```
df_1=pd.read_csv("chile.csv")
```

In [295]:
```
1 df_1
```

Out[295]:

	region	population	sex	age	education	income	statusquo	vote
0	N	175000	M	65.0	P	35000.0	1.00820	Y
1	N	175000	M	29.0	PS	7500.0	-1.29617	N
2	N	175000	F	38.0	P	15000.0	1.23072	Y
3	N	175000	F	49.0	P	35000.0	-1.03163	N
4	N	175000	F	23.0	S	35000.0	-1.10496	N
...	...	...	...	...	...	...	...	...
2695	M	15000	M	42.0	P	15000.0	-1.26247	N
2696	M	15000	F	28.0	P	15000.0	1.32950	Y
2697	M	15000	F	44.0	P	75000.0	1.42045	Y
2698	M	15000	M	21.0	S	75000.0	0.18315	NaN
2699	M	15000	M	20.0	PS	35000.0	1.38179	Y

2700 rows × 8 columns

In [296]:
```
1 df_1.shape
```

Out[296]:
```
(2700, 8)
```

In [297]:
```
1 df_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2700 entries, 0 to 2699
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   region      2700 non-null   object 
 1   population  2700 non-null   int64  
 2   sex         2700 non-null   object 
 3   age         2699 non-null   float64
 4   education   2689 non-null   object 
 5   income      2602 non-null   float64
 6   statusquo   2683 non-null   float64
 7   vote        2532 non-null   object 
dtypes: float64(3), int64(1), object(4)
memory usage: 168.9+ KB
```

In [298]:
```
1 df=df_1.dropna()
```

Vishal Acharya

In [299]:
```
df
```

Out[299]:

	region	population	sex	age	education	income	statusquo	vote
0	N	175000	M	65.0	P	35000.0	1.00820	Y
1	N	175000	M	29.0	PS	7500.0	-1.29617	N
2	N	175000	F	38.0	P	15000.0	1.23072	Y
3	N	175000	F	49.0	P	35000.0	-1.03163	N
4	N	175000	F	23.0	S	35000.0	-1.10496	N
...	...	...	...	...	...	...	...	...
2694	M	15000	M	42.0	S	35000.0	-0.00233	U
2695	M	15000	M	42.0	P	15000.0	-1.26247	N
2696	M	15000	F	28.0	P	15000.0	1.32950	Y
2697	M	15000	F	44.0	P	75000.0	1.42045	Y
2699	M	15000	M	20.0	PS	35000.0	1.38179	Y

2431 rows × 8 columns

In [300]:
```
ctab=pd.crosstab(df["sex"],df["education"])
ctab
```

Out[300]:

education	P	PS	S
sex			
F	552	184	514
M	450	235	496

In [301]:
```
ctab=pd.crosstab(df["sex"],df["education"],rownames=['a'],colnames=['b'])
ctab
```

Out[301]:

b	P	PS	S
a			
F	552	184	514
M	450	235	496

In [302]:
```
ctab=pd.crosstab(df["sex"],df["education"],margins=True)
ctab
```

Out[302]:

education	P	PS	S	All
sex				
F	552	184	514	1250
M	450	235	496	1181
All	1002	419	1010	2431

Vishal Acharya

In [303]:
```
ctab=pd.crosstab(df["sex"],df["education"],margins=True,margins_name="Total")
ctab
```

Out[303]:

education	P	PS	S	Total
sex				
F	552	184	514	1250
M	450	235	496	1181
Total	1002	419	1010	2431

In [304]:
```
ctab=pd.crosstab(df["sex"],df["education"],margins=True,margins_name="Total",normaliz
ctab
```

Out[304]:

education	P	PS	S	Total
sex				
F	0.227067	0.075689	0.211436	0.514192
M	0.185109	0.096668	0.204031	0.485808
Total	0.412176	0.172357	0.415467	1.000000

In [305]:
```
ctab=pd.crosstab(df["sex"],df["education"],margins=True,margins_name="Total",normaliz
ctab
```

Out[305]:

education	P	PS	S	Total
sex				
F	0.550898	0.439141	0.508911	0.514192
M	0.449102	0.560859	0.491089	0.485808

In [306]:
```
ctab=pd.crosstab(df["sex"],df["education"],margins=True,margins_name="Total",normaliz
ctab
```

Out[306]:

education	P	PS	S
sex			
F	0.441600	0.147200	0.411200
M	0.381033	0.198984	0.419983
Total	0.412176	0.172357	0.415467

Vishal Acharya

In [307]:
```
ctab=pd.crosstab(df["sex"],df["education"],values=df["income"],aggfunc={min,max,np.me
ctab
```

```
C:\Users\VISHAL\AppData\Local\Temp\ipykernel_1300\25724937.py:1: FutureWarning: The prov
ided callable <built-in function min> is currently using SeriesGroupBy.min. In a future
version of pandas, the provided callable will be used directly. To keep current behavior
pass the string "min" instead.
  ctab=pd.crosstab(df["sex"],df["education"],values=df["income"],aggfunc={min,max,np.mea
n})
C:\Users\VISHAL\AppData\Local\Temp\ipykernel_1300\25724937.py:1: FutureWarning: The prov
ided callable <function mean at 0x0000022A1DE64310> is currently using SeriesGroupBy.mea
n. In a future version of pandas, the provided callable will be used directly. To keep c
urrent behavior pass the string "mean" instead.
  ctab=pd.crosstab(df["sex"],df["education"],values=df["income"],aggfunc={min,max,np.mea
n})
C:\Users\VISHAL\AppData\Local\Temp\ipykernel_1300\25724937.py:1: FutureWarning: The prov
ided callable <built-in function max> is currently using SeriesGroupBy.max. In a future
version of pandas, the provided callable will be used directly. To keep current behavior
pass the string "max" instead.
  ctab=pd.crosstab(df["sex"],df["education"],values=df["income"],aggfunc={min,max,np.mea
n})
```

Out[307]:

	max			mean			min		
education	**P**	**PS**	**S**	**P**	**PS**	**S**	**P**	**PS**	**S**
sex									
F	125000.0	200000.0	200000.0	16449.275362	70230.978261	35267.509728	2500.0	2500.0	2500.0
M	200000.0	200000.0	200000.0	18866.666667	68085.106383	36456.653226	2500.0	7500.0	2500.0

In [308]:
```
ctab=pd.crosstab([df["sex"],df["region"]],df["education"])
ctab
```

Out[308]:

	education	**P**	**PS**	**S**
sex	**region**			
F	**C**	118	27	124
	M	28	4	11
	N	78	25	56
	S	165	40	125
	SA	163	88	198
M	**C**	120	49	110
	M	15	2	15
	N	55	36	55
	S	145	55	125
	SA	115	93	191

Vishal Acharya

In [309]:
```
ctab=pd.crosstab([df["sex"],df["region"]],[df["education"],df["vote"]])
ctab
```

Out[309]:

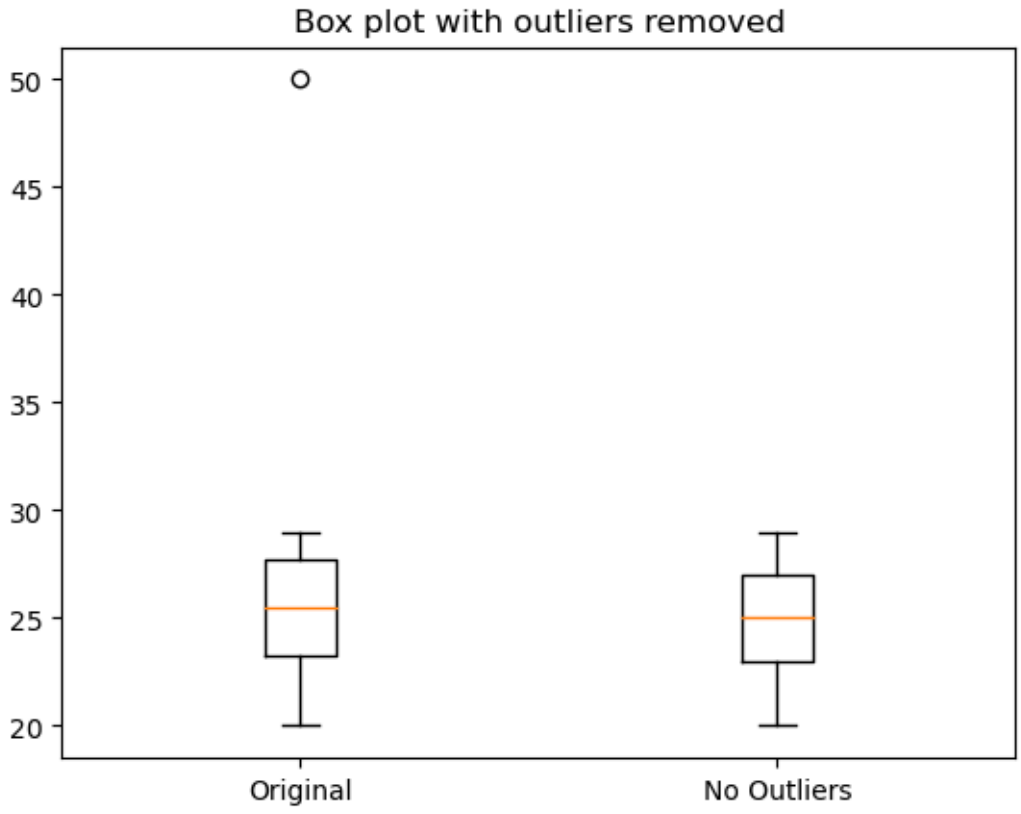
education		P				PS				S			
vote		A	N	U	Y	A	N	U	Y	A	N	U	Y
sex	region												
F	C	5	22	38	53	1	15	3	8	17	37	35	35
	M	0	3	10	15	0	1	1	2	0	2	3	6
	N	6	13	14	45	1	10	2	12	3	17	10	26
	S	8	33	43	81	3	17	7	13	12	37	34	42
	SA	12	41	62	48	8	43	16	21	21	64	61	52
M	C	6	48	34	32	2	33	3	11	13	54	17	26
	M	1	5	3	6	0	0	0	2	1	6	2	6
	N	2	15	11	27	5	19	4	8	12	24	5	14
	S	5	35	36	69	5	32	3	15	6	55	17	47
	SA	4	47	31	33	5	50	7	31	13	89	39	50

Vishal Acharya

how to remove outliers from a Pandas DataFrame and display the remaining data using a box plot:

In [310]:

```python
import pandas as pd
import matplotlib.pyplot as plt

# Create a sample DataFrame
df = pd.DataFrame({'A': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                   'B': [20, 21, 23, 24, 25, 26, 27, 28, 29, 50]})

# Define a function to remove outliers using the interquartile range (IQR) method
def remove_outliers(df, col):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]

# Remove outliers from column B using the remove_outliers function
df_no_outliers = remove_outliers(df, 'B')

# Plot the data using a box plot
fig, ax = plt.subplots()
ax.boxplot([df['B'], df_no_outliers['B']])
ax.set_xticklabels(['Original', 'No Outliers'])
ax.set_title('Box plot with outliers removed')
plt.show()

```



Vishal Acharya

Exercise with corr,parallel_cordinate,scatter_matrix

In [233]:

```python
import pandas as pd
dataset = pd.read_csv('auto-mpg.csv')
```

12. Handle categorical data: Convert categorical data to numeric using the get_dummies() function.

- some other step require

In [235]:
```
v=pd.get_dummies(dataset["origin"])
v
```

Out[235]:

	1	2	3
0	True	False	False
1	True	False	False
2	True	False	False
3	True	False	False
4	True	False	False
...	...	...	...
393	True	False	False
394	False	True	False
395	True	False	False
396	True	False	False
397	True	False	False

398 rows × 3 columns

In [215]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object 
 4   weight        398 non-null    int64  
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64  
 7   origin        398 non-null    int64  
 8   car name      398 non-null    object 
dtypes: float64(3), int64(4), object(2)
memory usage: 28.1+ KB
```

Vishal Acharya

In [216]:
```
1 dataset.tail()
```

Out[216]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

In [217]:
```
1 dataset.head()
```

Out[217]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

In [218]:
```
1 dataset.shape
```

Out[218]: (398, 9)

In [219]:
```
1 dataset.describe()
```

Out[219]:

	mpg	cylinders	displacement	weight	acceleration	model year	origin
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	2970.424623	15.568090	76.010050	1.572864
std	7.815984	1.701004	104.269838	846.841774	2.757689	3.697627	0.802055
min	9.000000	3.000000	68.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.500000	4.000000	104.250000	2223.750000	13.825000	73.000000	1.000000
50%	23.000000	4.000000	148.500000	2803.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	262.000000	3608.000000	17.175000	79.000000	2.000000
max	46.600000	8.000000	455.000000	5140.000000	24.800000	82.000000	3.000000

Vishal Acharya

In [220]:
```
dataset.describe(include='all')
```

Out[220]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
count	398.000000	398.000000	398.000000	398	398.000000	398.000000	398.000000	398.000000
unique	NaN	NaN	NaN	94	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	150	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	22	NaN	NaN	NaN	NaN
mean	23.514573	5.454774	193.425879	NaN	2970.424623	15.568090	76.010050	1.572864
std	7.815984	1.701004	104.269838	NaN	846.841774	2.757689	3.697627	0.802055
min	9.000000	3.000000	68.000000	NaN	1613.000000	8.000000	70.000000	1.000000
25%	17.500000	4.000000	104.250000	NaN	2223.750000	13.825000	73.000000	1.000000
50%	23.000000	4.000000	148.500000	NaN	2803.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	262.000000	NaN	3608.000000	17.175000	79.000000	2.000000
max	46.600000	8.000000	455.000000	NaN	5140.000000	24.800000	82.000000	3.000000

Vishal Acharya

A great aspect of the Pandas module is the corr() method.

The corr() method calculates the relationship between each column in your data set.

In [221]:
```
# Get the numerical columns
numerical_columns = dataset.select_dtypes(include=['number'])

# Create a new DataFrame with only numerical columns
numerical_df = dataset[numerical_columns.columns]
```

In [222]:
```
numerical_df.corr()
```

Out[222]:

	mpg	cylinders	displacement	weight	acceleration	model year	origin
mpg	1.000000	-0.775396	-0.804203	-0.831741	0.420289	0.579267	0.563450
cylinders	-0.775396	1.000000	0.950721	0.896017	-0.505419	-0.348746	-0.562543
displacement	-0.804203	0.950721	1.000000	0.932824	-0.543684	-0.370164	-0.609409
weight	-0.831741	0.896017	0.932824	1.000000	-0.417457	-0.306564	-0.581024
acceleration	0.420289	-0.505419	-0.543684	-0.417457	1.000000	0.288137	0.205873
model year	0.579267	-0.348746	-0.370164	-0.306564	0.288137	1.000000	0.180662
origin	0.563450	-0.562543	-0.609409	-0.581024	0.205873	0.180662	1.000000

- Perfect Correlation: We can see that "mpg" and "mpg" got the number 1.000000, which makes sense, each column always has a perfect relationship with itself.
- Good Correlation: "cylinders" and "displacement" got a 0.950721 correlation, which is a very good correlation, and we can predict that more cylinders means more displacement.
- Bad Correlation: "model year" and "acceleration" got a 0.288137 correlation, which is a very bad correlation, meaning that we can not predict the max pulse by just looking at the duration of the work out, and vice versa.

Scatter Matrix/Pair Plots Returns a numpy.ndarray By default, alpha=1. If you would like to form the graph plot more transparent, then you'll make alpha but 1, such as 0.5 or 0.25.

- If you would like to form the graph plot less transparent, then you'll make alpha greater than 1. This solidifies the graph plot, making it less transparent and more thick and dense,

In [223]:
```
import matplotlib.pyplot as plt
pd.plotting.scatter_matrix(dataset, figsize = [15, 15], marker = 'v', alpha=0.5)
plt.show()
```

Vishal Acharya

Qualitative Data vs Quantitative Data

- Quantitative data relates to information about the quantity of an object – hence it can be measured. For example, if we consider the attribute 'marks', it can be measured using a scale of measurement. Quantitative data is also termed as numeric data. Qualitative data provides information about the quality of an object or information which cannot be measured. For example, if we consider the quality of performance of students in terms of 'Good', 'Average', and 'Poor', it falls under the category of qualitative data. Also, name or roll number of students are information that cannot be measured using some scale

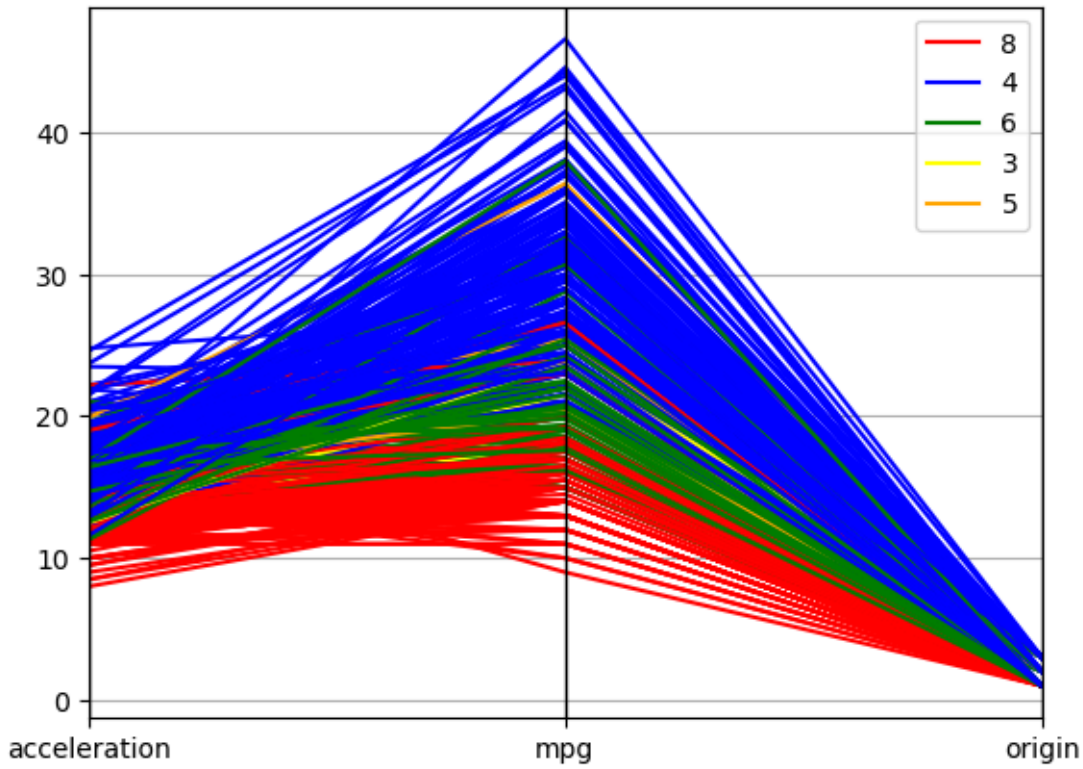
of measurement. So they would fall under qualitative data. Qualitative data is also called categorical data. Quantitative Data can be analyzed by measures like mean, median, mode.

- For qualitative data, we can use parallel coordinates and cross tabulation.

Parallel coordinates

Parallel coordinates charts are commonly used to visualize and analyze high dimensional multivariate data. It represents each data sample as polyline connecting parallel lines where each parallel line represents an attribute of that data sample

In [224]:
```
import matplotlib.pyplot as plt
from pandas.plotting import parallel_coordinates
pll = parallel_coordinates(dataset, 'cylinders', cols=['acceleration','mpg', 'origin'
                           color=('red', 'blue', 'green', 'yellow', 'orange'))
plt.show()
```



In [225]:
```
pd.crosstab(dataset['cylinders'], dataset['model year'], rownames=['cylinders'], colr
```

Out[225]:

model year	70	71	72	73	74	75	76	77	78	79	80	81	82
cylinders													
3	0	0	1	1	0	0	0	1	0	0	1	0	0
4	7	13	14	11	15	12	15	14	17	12	25	21	28
5	0	0	0	0	0	0	0	0	1	1	1	0	0
6	4	8	0	8	7	12	10	5	12	6	2	7	3
8	18	7	13	20	5	6	9	8	6	10	0	1	0

In [226]:
```
dataset = dataset[dataset['horsepower']!='?']
```

Vishal Acharya

In [227]:
```
#all rows with ? are gone
dataset[dataset['horsepower']=='?']
```

Out[227]:

mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name

In [228]:
```
import pandas as pd
#finding outliers in 'mpg'
def find_outliers(ds, col):
  quart1 = ds[col].quantile(0.25)
  quart3 = ds[col].quantile(0.75)
  IQR = quart3 - quart1 #Inter-quartile range
  low_val = quart1 - 1.5*IQR
  high_val = quart3 + 1.5*IQR
  ds = ds.loc[(ds[col] < low_val) | (ds[col] > high_val)]
  return ds
dataset = pd.read_csv('auto-mpg.csv')
find_outliers(dataset, 'mpg')
```

Out[228]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
322	46.6	4	86.0	65	2110	17.9	80	3	mazda glc

In [229]:
```
import pandas as pd
#finding outliers in ''acceleration''
def find_outliers(ds, col):
  quart1 = ds[col].quantile(0.25)
  quart3 = ds[col].quantile(0.75)
  IQR = quart3 - quart1 #Inter-quartile range
  low_val = quart1 - 1.5*IQR
  high_val = quart3 + 1.5*IQR
  ds = ds.loc[(ds[col] < low_val) | (ds[col] > high_val)]
  return ds
dataset = pd.read_csv('auto-mpg.csv')
find_outliers(dataset, 'acceleration')
```

Out[229]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
7	14.0	8	440.0	215	4312	8.5	70	1	plymouth fury iii
9	15.0	8	390.0	190	3850	8.5	70	1	amc ambassador dpl
11	14.0	8	340.0	160	3609	8.0	70	1	plymouth 'cuda 340
59	23.0	4	97.0	54	2254	23.5	72	2	volkswagen type 3
299	27.2	4	141.0	71	3190	24.8	79	2	peugeot 504
326	43.4	4	90.0	48	2335	23.7	80	2	vw dasher (diesel)
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup

Vishal Acharya

In [230]:
```
def remove_outliers(ds, col):
  quart1 = ds[col].quantile(0.25)
  quart3 = ds[col].quantile(0.75)
  IQR = quart3 - quart1 #Interquartile range
  low_val = quart1 - 1.5*IQR
  print(low_val)
  high_val = quart3 + 1.5*IQR
  print(high_val)
  df_out = ds.loc[(ds[col] >= low_val) & (ds[col] <= high_val)]
  return df_out
new_data = remove_outliers(dataset, 'acceleration')
```

```
8.8
22.2
```

In [231]:
```
#outliers removed from new_data
new_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 391 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           391 non-null    float64
 1   cylinders     391 non-null    int64  
 2   displacement  391 non-null    float64
 3   horsepower    391 non-null    object 
 4   weight        391 non-null    int64  
 5   acceleration  391 non-null    float64
 6   model year    391 non-null    int64  
 7   origin        391 non-null    int64  
 8   car name      391 non-null    object 
dtypes: float64(3), int64(4), object(2)
memory usage: 30.5+ KB
```

Vishal Acharya

In [232]:
```
import pandas as pd

data = {
  "name": ["Sally", "Mary", "John"],
  "age": [50, 40, 30],
  "qualified": [True, False, False]
}
idx = ["X", "Y", "Z"]

df = pd.DataFrame(data, index=idx)
print(df)
newdf = df.reset_index()

print(newdf)
```

```
    name  age  qualified
X  Sally   50       True
Y   Mary   40      False
Z   John   30      False
  index   name  age  qualified
0     X  Sally   50       True
1     Y   Mary   40      False
2     Z   John   30      False
```

In [ ]:
```

```

Vishal Acharya

In [3]:

```
import pandas as pd
ipl=pd.read_csv('IPL_Matches_2008_2022.csv')
ipl
```

Out[3]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecisio
0	1312200	Ahmedabad	2022-05-29	2022	Final	Rajasthan Royals	Gujarat Titans	Narendra Modi Stadium, Ahmedabad	Rajasthan Royals	ba
1	1312199	Ahmedabad	2022-05-27	2022	Qualifier 2	Royal Challengers Bangalore	Rajasthan Royals	Narendra Modi Stadium, Ahmedabad	Rajasthan Royals	fie
2	1312198	Kolkata	2022-05-25	2022	Eliminator	Royal Challengers Bangalore	Lucknow Super Giants	Eden Gardens, Kolkata	Lucknow Super Giants	fie
3	1312197	Kolkata	2022-05-24	2022	Qualifier 1	Rajasthan Royals	Gujarat Titans	Eden Gardens, Kolkata	Gujarat Titans	fie
4	1304116	Mumbai	2022-05-22	2022	70	Sunrisers Hyderabad	Punjab Kings	Wankhede Stadium, Mumbai	Sunrisers Hyderabad	ba
...	...	...	...	...	...	...	...	...	...	
945	335986	Kolkata	2008-04-20	2007/08	4	Kolkata Knight Riders	Deccan Chargers	Eden Gardens	Deccan Chargers	ba
946	335985	Mumbai	2008-04-20	2007/08	5	Mumbai Indians	Royal Challengers Bangalore	Wankhede Stadium	Mumbai Indians	ba
947	335984	Delhi	2008-04-19	2007/08	3	Delhi Daredevils	Rajasthan Royals	Feroz Shah Kotla	Rajasthan Royals	ba
948	335983	Chandigarh	2008-04-19	2007/08	2	Kings XI Punjab	Chennai Super Kings	Punjab Cricket Association Stadium, Mohali	Chennai Super Kings	ba
949	335982	Bangalore	2008-04-18	2007/08	1	Royal Challengers Bangalore	Kolkata Knight Riders	M Chinnaswamy Stadium	Royal Challengers Bangalore	fie

950 rows × 20 columns

select single column and mult columns

In [5]:
```
print("single")
print(ipl["City"])
```

```
single
0       Ahmedabad
1       Ahmedabad
2         Kolkata
3         Kolkata
4          Mumbai
          ...
945       Kolkata
946        Mumbai
947         Delhi
948    Chandigarh
949     Bangalore
Name: City, Length: 950, dtype: object
```

In [11]:
```
print("multi")
print(ipl[["City","Date"]])
```

```
multi
           City        Date
0     Ahmedabad  2022-05-29
1     Ahmedabad  2022-05-27
2       Kolkata  2022-05-25
3       Kolkata  2022-05-24
4        Mumbai  2022-05-22
..          ...         ...
945     Kolkata  2008-04-20
946      Mumbai  2008-04-20
947       Delhi  2008-04-19
948  Chandigarh  2008-04-19
949   Bangalore  2008-04-18

[950 rows x 2 columns]
```

In [ ]:
```

```