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.

- Prudhvi Vardhan Notes

Pandas Series

A Pandas Series is like a column in a table. It is a 1-D array holding data of any type.

Importing Pandas

In [4]:
```
import numpy as np
import pandas as pd
```

Series using String

In [6]:
```
# string
country = ['India','Pakistan','USA','Nepal','Srilanka']
pd.Series(country)
```

Out[6]:
```
0       India
1    Pakistan
2         USA
3       Nepal
4    Srilanka
dtype: object
```

In [7]:
```
# integers
marks= [13,24,56,78,100]
pd.Series(marks)
```

Out[7]:
```
0     13
1     24
2     56
3     78
4    100
dtype: int64
```

In [8]:
```
# custom index
marks = [67,57,89,100]
subjects = ['maths','english','science','hindi']

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

Out[8]:
```
maths       67
english     57
science     89
hindi      100
dtype: int64
```

In [10]:
```
# setting a name
marks = pd.Series(marks , index=subjects , name="Jack Marks")
marks
```

Out[10]:
```
maths       67
english     57
science     89
hindi      100
Name: Jack Marks, dtype: int64
```

Series from dictionary

When a Pandas Series is converted to a dictionary using the to_dict() method, the resulting dictionary has the same keys and values as the Series. The index values of the Series become the keys in the dictionary, and the corresponding values become the values in the dictionary.

In [11]:
```
marks = {
    'maths':67,
    'english':57,
    'science':89,
    'hindi':100
}
marks_series = pd.Series(marks,name="jack Marks")
```

In [12]:
```
marks_series
```

Out[12]:
```
maths       67
english     57
science     89
hindi      100
Name: jack Marks, dtype: int64
```

Series Attributes

size: Returns the number of elements in the Series.

In [13]:
```
# size
marks_series.size
```

Out[13]:
```
4
```

dtype: Returns the data type of the values in the Series.

In [14]:
```
# dtype
marks_series.dtype
```

Out[14]:
```
dtype('int64')
```

name: Returns the name of the Series.

In [15]:
```
# name
marks_series.name
```

Out[15]:
```
'jack Marks'
```

unique is an attribute of a Pandas Series that returns an array of the unique values in the Series.

In [16]:
```
# is_unique
marks_series.is_unique
```

Out[16]:
```
True
```

In [17]:
```
pd.Series([1,1,2,3,4,44,2]).is_unique #It gives false because of repetation
```

Out[17]:
```
False
```

index: Returns the index labels of the Series.

In [18]:
```
# index
marks_series.index
```

Out[18]:
```
Index(['maths', 'english', 'science', 'hindi'], dtype='object')
```

values: Returns the data contained in the Series as a NumPy array.

In [19]:
```
# values
marks_series.values
```

Out[19]:
```
array([ 67,  57,  89, 100], dtype=int64)
```

In [20]:
```
type(marks_series.values)
```

Out[20]:
```
numpy.ndarray
```

Series using read_csv

In [21]:
```
# with one col
sub = pd.read_csv("D:\\datascience\\Nitish isr\\Pandas\\subs.csv")
```

Pandas.read_csv

Automatically converts everything into data frames not in series.

In [23]:
```
type(sub)
```

Out[23]:
```
pandas.core.frame.DataFrame
```

In [30]:
```
sub.head(5)
```

Out[30]:

	Subscribers gained
0	48
1	57
2	40
3	43
4	44

To convert data into series,

we have to apply a parameter called as"Squeeze" is equals to True.

In [31]:
```
sub = pd.read_csv("subs.csv",squeeze=True)
```

In [32]:
```
type(sub)
```

Out[32]:
```
pandas.core.series.Series
```

In [33]:
```
sub
```

Out[33]:
```
0       48
1       57
2       40
3       43
4       44
      ...
360    231
361    226
362    155
363    144
364    172
Name: Subscribers gained, Length: 365, dtype: int64
```

In [56]:
```
#With 2 col
kl=pd.read_csv("kohli_ipl.csv",index_col="match_no",squeeze=True)
```

In [57]:
```
kl
```

Out[57]:
```
match_no
1       1
2      23
3      13
4      12
5       1
      ..
211     0
212    20
213    73
214    25
215     7
Name: runs, Length: 215, dtype: int64
```

In [37]:
```
movies=pd.read_csv( "bollywood.csv", index_col="movie",squeeze=True)
```

In [38]:
```
movies
```

Out[38]:
```
movie
Uri: The Surgical Strike                   Vicky Kaushal
Battalion 609                                Vicky Ahuja
The Accidental Prime Minister (film)         Anupam Kher
Why Cheat India                            Emraan Hashmi
Evening Shadows                         Mona Ambegaonkar
                                                    ...
Hum Tumhare Hain Sanam                    Shah Rukh Khan
Aankhen (2002 film)                      Amitabh Bachchan
Saathiya (film)                              Vivek Oberoi
Company (film)                                 Ajay Devgn
Awara Paagal Deewana                         Akshay Kumar
Name: lead, Length: 1500, dtype: object
```

Series Methods

head(n): Returns the first n elements of the Series.

In [40]:
```
# Head
sub.head()
```

Out[40]:
```
0    48
1    57
2    40
3    43
4    44
Name: Subscribers gained, dtype: int64
```

tail(n): Returns the last n elements of the Series.

In [41]:
```
# tail
kl.tail()
```

Out[41]:
```
match_no
211     0
212    20
213    73
214    25
215     7
Name: runs, dtype: int64
```

In [43]:
```
# sample - Gives random data
movies.sample()
```

Out[43]:
```
movie
Enemmy    Sunil Shetty
Name: lead, dtype: object
```

value_counts(): Returns a Series containing the counts of unique values in the Series.

In [44]:
```
# Value Counts
movies.value_counts()
```

Out[44]:
```
Akshay Kumar        48
Amitabh Bachchan    45
Ajay Devgn          38
Salman Khan         31
Sanjay Dutt         26
                    ..
Diganth              1
Parveen Kaur         1
Seema Azmi           1
Akanksha Puri        1
Edwin Fernandes      1
Name: lead, Length: 566, dtype: int64
```

In [45]:
```
#sort_values - temperory changes #### sort_values(): Returns a sorted Series b
kl.sort_values()
```

Out[45]:
```
match_no
87       0
211      0
207      0
206      0
91       0
       ...
164    100
120    100
123    108
126    109
128    113
Name: runs, Length: 215, dtype: int64
```

In [50]:
```
# method chaining
kl.sort_values(ascending=False).head(1).values[0]
```

Out[50]:
```
113
```

In [55]:
```
# For permanent Changes use Inplace
kl.sort_values(inplace=True)
kl
```

Out[55]:
```
match_no
87       0
211      0
207      0
206      0
91       0
       ...
164    100
120    100
123    108
126    109
128    113
Name: runs, Length: 215, dtype: int64
```

In [60]:
```
# sort_index -> inplace -> movies

movies.sort_index()
```

Out[60]:
```
movie
1920 (film)                     Rajniesh Duggall
1920: London                       Sharman Joshi
1920: The Evil Returns               Vicky Ahuja
1971 (2007 film)                  Manoj Bajpayee
2 States (2014 film)                Arjun Kapoor
                                          ...
Zindagi 50-50                          Veena Malik
Zindagi Na Milegi Dobara          Hrithik Roshan
Zindagi Tere Naam           Mithun Chakraborty
Zokkomon                          Darsheel Safary
Zor Lagaa Ke...Haiya!               Meghan Jadhav
Name: lead, Length: 1500, dtype: object
```

In [61]:
```
movies.sort_index(ascending=False)
```

Out[61]:
```
movie
Zor Lagaa Ke...Haiya!               Meghan Jadhav
Zokkomon                          Darsheel Safary
Zindagi Tere Naam           Mithun Chakraborty
Zindagi Na Milegi Dobara          Hrithik Roshan
Zindagi 50-50                          Veena Malik
                                          ...
2 States (2014 film)                Arjun Kapoor
1971 (2007 film)                  Manoj Bajpayee
1920: The Evil Returns               Vicky Ahuja
1920: London                       Sharman Joshi
1920 (film)                     Rajniesh Duggall
Name: lead, Length: 1500, dtype: object
```

Series Maths Methods

Diffence between Count And Size

Count gives the total number of items present in the series. But only NON missing values but, if we have missing values ,it doesnt count them . But, size gives the total item including missing values

In [62]:
```
# count
kl.count()
```

Out[62]: 215

sum(): Returns the sum of the values in the Series.

In [66]:
```
# sum -> Product
sub.sum()
```

Out[66]: 49510

In [67]:
```
sub.product() # Multiply the items
```

Out[67]: 0

Statical Methods

mean(): Returns the mean value of the Series.

In [68]:
```
# mean


sub.mean()
```

Out[68]: 135.64383561643837

median(): Returns the median value of the Series.

In [72]:
```
# median
kl.median()
```

Out[72]: 24.0

mode(): The mode is the value that appears most frequently in the Series.

In [74]:
```
# mode
print(movies.mode())
```

```
0    Akshay Kumar
dtype: object
```

std(): Returns the standard deviation of the values in the Series.

In [71]:
```
# std -> Standard deviation
sub.std()
```

Out[71]: 62.67502303725269

In [75]:
```
# var -> varience
sub.var()
```

Out[75]: 3928.1585127201556

min(): Returns the minimum value of the Series.

In [76]:
```
# min
sub.min()
```

Out[76]: 33

max(): Returns the maximum value of the Series.

In [77]:
```
# max
sub.max()
```

Out[77]: 396

describe(): Generates descriptive statistics of the Series.

In [79]:
```
# describe
movies.describe()
```

Out[79]:
```
count             1500
unique             566
top       Akshay Kumar
freq                48
Name: lead, dtype: object
```

In [80]:
```
kl.describe()
```

Out[80]:
```
count    215.000000
mean      30.855814
std       26.229801
min        0.000000
25%        9.000000
50%       24.000000
75%       48.000000
max      113.000000
Name: runs, dtype: float64
```

In [81]:
```
sub.describe()
```

Out[81]:
```
count    365.000000
mean     135.643836
std       62.675023
min       33.000000
25%       88.000000
50%      123.000000
75%      177.000000
max      396.000000
Name: Subscribers gained, dtype: float64
```

Series Indexing

In [83]:
```
# integer indexing
x = pd.Series([12,13,14,35,46,57,58,79,9])
x[1]
```

Out[83]:
```
13
```

In [85]:
```
# negative indexing
movies[-1]
```

Out[85]:
```
'Akshay Kumar'
```

In [86]:
```
movies[0]
```

Out[86]:
```
'Vicky Kaushal'
```

In [87]:
```
sub[0]
```

Out[87]:
```
48
```

In [90]:
```
# slicing
kl[4:10]
```

Out[90]:
```
match_no
5      1
6      9
7     34
8      0
9     21
10     3
Name: runs, dtype: int64
```

In [95]:
```
#Negative slicing

sub[-5:]
```

Out[95]:
```
360    231
361    226
362    155
363    144
364    172
Name: Subscribers gained, dtype: int64
```

In [96]:
```
movies[-5:]
```

Out[96]:
```
movie
Hum Tumhare Hain Sanam      Shah Rukh Khan
Aankhen (2002 film)       Amitabh Bachchan
Saathiya (film)               Vivek Oberoi
Company (film)                  Ajay Devgn
Awara Paagal Deewana          Akshay Kumar
Name: lead, dtype: object
```

In [97]:
```
movies[::2]
```

Out[97]:
```
movie
Uri: The Surgical Strike                   Vicky Kaushal
The Accidental Prime Minister (film)         Anupam Kher
Evening Shadows                         Mona Ambegaonkar
Fraud Saiyaan                               Arshad Warsi
Manikarnika: The Queen of Jhansi          Kangana Ranaut
                                                ...
Raaz (2002 film)                            Dino Morea
Waisa Bhi Hota Hai Part II                  Arshad Warsi
Kaante                                  Amitabh Bachchan
Aankhen (2002 film)                     Amitabh Bachchan
Company (film)                                Ajay Devgn
Name: lead, Length: 750, dtype: object
```

In [98]:
```
# Fancy indexing
kl[[1,8,22,11,2]]
```

Out[98]:
```
match_no
1      1
8      0
22    38
11    10
2     23
Name: runs, dtype: int64
```

In [99]:
```
# Fancy indexing -> indexing with labels
movies
```

Out[99]:
```
movie
Uri: The Surgical Strike                   Vicky Kaushal
Battalion 609                                Vicky Ahuja
The Accidental Prime Minister (film)         Anupam Kher
Why Cheat India                            Emraan Hashmi
Evening Shadows                         Mona Ambegaonkar
                                                   ...
Hum Tumhare Hain Sanam                    Shah Rukh Khan
Aankhen (2002 film)                      Amitabh Bachchan
Saathiya (film)                              Vivek Oberoi
Company (film)                               Ajay Devgn
Awara Paagal Deewana                       Akshay Kumar
Name: lead, Length: 1500, dtype: object
```

In [100]:
```
movies['Evening Shadows']
```

Out[100]:
```
'Mona Ambegaonkar'
```

Editing the series

In [101]:
```
# using the index number
marks_series
```

Out[101]:
```
maths       67
english     57
science     89
hindi      100
Name: jack Marks, dtype: int64
```

In [102]:
```
marks_series[1]=88
marks_series
```

Out[102]:
```
maths       67
english     88
science     89
hindi      100
Name: jack Marks, dtype: int64
```

In [103]:
```
# we can add data , if it doesnt exist
marks_series['social']=90
marks_series
```

Out[103]:
```
maths       67
english     88
science     89
hindi      100
social      90
Name: jack Marks, dtype: int64
```

In [111]:
```
# using index label
movies
```

Out[111]:
```
movie
Uri: The Surgical Strike                   Vicky Kaushal
Battalion 609                                Vicky Ahuja
The Accidental Prime Minister (film)         Anupam Kher
Why Cheat India                            Emraan Hashmi
Evening Shadows                          Mona Ambegaonkar
                                                    ...
Hum Tumhare Hain Sanam                     Shah Rukh Khan
Aankhen (2002 film)                      Amitabh Bachchan
Saathiya (film)                              Vivek Oberoi
Company (film)                                Ajay Devgn
Awara Paagal Deewana                       Akshay Kumar
Name: lead, Length: 1500, dtype: object
```

In [114]:
```
movies['Hum Tumhare Hain Sanam'] = 'Jack'
```

In [115]:
```
movies
```

Out[115]:
```
movie
Uri: The Surgical Strike                   Vicky Kaushal
Battalion 609                                Vicky Ahuja
The Accidental Prime Minister (film)         Anupam Kher
Why Cheat India                            Emraan Hashmi
Evening Shadows                          Mona Ambegaonkar
                                                    ...
Hum Tumhare Hain Sanam                              Jack
Aankhen (2002 film)                      Amitabh Bachchan
Saathiya (film)                              Vivek Oberoi
Company (film)                                Ajay Devgn
Awara Paagal Deewana                       Akshay Kumar
Name: lead, Length: 1500, dtype: object
```

Series with Python Functionalities

In [117]:
```
# len/type/dir/sorted/max/min
print(len(sub))
print(type(sub))
```

```
365
<class 'pandas.core.series.Series'>
```

In [122]:
```
print(dir(sub))
print(sorted(sub))
```

```
['T', '_AXIS_LEN', '_AXIS_ORDERS', '_AXIS_REVERSED', '_AXIS_TO_AXIS_NUMBER',
'_HANDLED_TYPES', '__abs__', '__add__', '__and__', '__annotations__', '__arra
y__', '__array_priority__', '__array_ufunc__', '__array_wrap__', '__bool__',
'__class__', '__contains__', '__copy__', '__deepcopy__', '__delattr__', '__de
litem__', '__dict__', '__dir__', '__divmod__', '__doc__', '__eq__', '__finali
ze__', '__float__', '__floordiv__', '__format__', '__ge__', '__getattr__', '_
_getattribute__', '__getitem__', '__getstate__', '__gt__', '__hash__', '__iad
d__', '__iand__', '__ifloordiv__', '__imod__', '__imul__', '__init__', '__ini
t_subclass__', '__int__', '__invert__', '__ior__', '__ipow__', '__isub__', '_
_iter__', '__itruediv__', '__ixor__', '__le__', '__len__', '__long__', '__lt_
_', '__matmul__', '__mod__', '__module__', '__mul__', '__ne__', '__neg__', '_
_new__', '__nonzero__', '__or__', '__pos__', '__pow__', '__radd__', '__rand_
_', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv_
_', '__rmatmul__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow_
_', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__setitem__', '__
setstate__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__trued
iv__', '__weakref__', '__xor__', '_accessors', '_accum_func', '_add_numeric_o
perations', '_agg_by_level', '_agg_examples_doc', '_agg_see_also_doc', '_alig
n_frame', '_align_series', '_arith_method', '_as_manager', '_attrs', '_bino
p', '_can_hold_na', '_check_inplace_and_allows_duplicate_labels', '_check_inp
lace_setting', '_check_is_chained_assignment_possible', '_check_label_or_leve
l_ambiguity', '_check_setitem_copy', '_clear_item_cache', '_clip_with_one_bou
nd', '_clip_with_scalar', '_cmp_method', '_consolidate', '_consolidate_inplac
e', '_construct_axes_dict', '_construct_axes_from_arguments', '_construct_res
ult', '_constructor', '_constructor_expanddim', '_convert', '_convert_dtype
s', '_data', '_dir_additions', '_dir_deletions', '_drop_axis', '_drop_labels_
or_levels', '_duplicated', '_find_valid_index', '_flags', '_from_mgr', '_get_
axis', '_get_axis_name', '_get_axis_number', '_get_axis_resolvers', '_get_blo
ck_manager_axis', '_get_bool_data', '_get_cacher', '_get_cleaned_column_resol
vers', '_get_index_resolvers', '_get_label_or_level_values', '_get_numeric_da
ta', '_get_value', '_get_values', '_get_values_tuple', '_get_with', '_gotite
m', '_hidden_attrs', '_index', '_indexed_same', '_info_axis', '_info_axis_nam
e', '_info_axis_number', '_init_dict', '_init_mgr', '_inplace_method', '_inte
rnal_names', '_internal_names_set', '_is_cached', '_is_copy', '_is_label_or_l
evel_reference', '_is_label_reference', '_is_level_reference', '_is_mixed_typ
e', '_is_view', '_item_cache', '_ixs', '_logical_func', '_logical_method', '_
map_values', '_maybe_update_cacher', '_memory_usage', '_metadata', '_mgr', '_
min_count_stat_function', '_name', '_needs_reindex_multi', '_protect_consolid
ate', '_reduce', '_reindex_axes', '_reindex_indexer', '_reindex_multi', '_rei
ndex_with_indexers', '_replace_single', '_repr_data_resource_', '_repr_latex
_', '_reset_cache', '_reset_cacher', '_set_as_cached', '_set_axis', '_set_axi
s_name', '_set_axis_nocheck', '_set_is_copy', '_set_labels', '_set_name', '_s
et_value', '_set_values', '_set_with', '_set_with_engine', '_slice', '_stat_a
xis', '_stat_axis_name', '_stat_axis_number', '_stat_function', '_stat_functi
on_ddof', '_take_with_is_copy', '_typ', '_update_inplace', '_validate_dtype',
'_values', '_where', 'abs', 'add', 'add_prefix', 'add_suffix', 'agg', 'aggreg
ate', 'align', 'all', 'any', 'append', 'apply', 'argmax', 'argmin', 'argsor
t', 'array', 'asfreq', 'asof', 'astype', 'at', 'at_time', 'attrs', 'autocor
r', 'axes', 'backfill', 'between', 'between_time', 'bfill', 'bool', 'clip',
'combine', 'combine_first', 'compare', 'convert_dtypes', 'copy', 'corr', 'cou
nt', 'cov', 'cummax', 'cummin', 'cumprod', 'cumsum', 'describe', 'diff', 'di
v', 'divide', 'divmod', 'dot', 'drop', 'drop_duplicates', 'droplevel', 'dropn
a', 'dtype', 'dtypes', 'duplicated', 'empty', 'eq', 'equals', 'ewm', 'expandi
ng', 'explode', 'factorize', 'ffill', 'fillna', 'filter', 'first', 'first_val
id_index', 'flags', 'floordiv', 'ge', 'get', 'groupby', 'gt', 'hasnans', 'hea
d', 'hist', 'iat', 'idxmax', 'idxmin', 'iloc', 'index', 'infer_objects', 'int
erpolate', 'is_monotonic', 'is_monotonic_decreasing', 'is_monotonic_increasin
```

```
g', 'is_unique', 'isin', 'isna', 'isnull', 'item', 'items', 'iteritems', 'key
s', 'kurt', 'kurtosis', 'last', 'last_valid_index', 'le', 'loc', 'lt', 'mad',
'map', 'mask', 'max', 'mean', 'median', 'memory_usage', 'min', 'mod', 'mode',
'mul', 'multiply', 'name', 'nbytes', 'ndim', 'ne', 'nlargest', 'notna', 'notn
ull', 'nsmallest', 'nunique', 'pad', 'pct_change', 'pipe', 'plot', 'pop', 'po
w', 'prod', 'product', 'quantile', 'radd', 'rank', 'ravel', 'rdiv', 'rdivmo
d', 'reindex', 'reindex_like', 'rename', 'rename_axis', 'reorder_levels', 're
peat', 'replace', 'resample', 'reset_index', 'rfloordiv', 'rmod', 'rmul', 'ro
lling', 'round', 'rpow', 'rsub', 'rtruediv', 'sample', 'searchsorted', 'sem',
'set_axis', 'set_flags', 'shape', 'shift', 'size', 'skew', 'slice_shift', 'so
rt_index', 'sort_values', 'squeeze', 'std', 'sub', 'subtract', 'sum', 'swapax
es', 'swaplevel', 'tail', 'take', 'to_clipboard', 'to_csv', 'to_dict', 'to_ex
cel', 'to_frame', 'to_hdf', 'to_json', 'to_latex', 'to_list', 'to_markdown',
'to_numpy', 'to_period', 'to_pickle', 'to_sql', 'to_string', 'to_timestamp',
'to_xarray', 'transform', 'transpose', 'truediv', 'truncate', 'tz_convert',
'tz_localize', 'unique', 'unstack', 'update', 'value_counts', 'values', 'va
r', 'view', 'where', 'xs']
[33, 33, 35, 37, 39, 40, 40, 40, 40, 42, 42, 43, 44, 44, 44, 45, 46, 46, 48,
49, 49, 49, 49, 50, 50, 50, 51, 54, 56, 56, 56, 56, 57, 61, 62, 64, 65, 65, 6
6, 66, 66, 66, 67, 68, 70, 70, 70, 71, 71, 72, 72, 72, 72, 72, 73, 74, 74, 7
5, 76, 76, 76, 76, 77, 77, 78, 78, 78, 79, 79, 80, 80, 80, 81, 81, 82, 82, 8
3, 83, 83, 84, 84, 84, 85, 86, 86, 86, 87, 87, 87, 87, 88, 88, 88, 88, 88, 8
9, 89, 89, 90, 90, 90, 90, 91, 92, 92, 92, 93, 93, 93, 93, 95, 95, 96, 96, 9
6, 96, 97, 97, 98, 98, 99, 99, 100, 100, 100, 101, 101, 101, 102, 102, 103, 1
03, 104, 104, 104, 105, 105, 105, 105, 105, 105, 105, 105, 105, 108, 108, 10
8, 108, 108, 108, 109, 109, 110, 110, 110, 111, 111, 112, 113, 113, 113, 114,
114, 114, 114, 115, 115, 115, 115, 117, 117, 117, 118, 118, 119, 119, 119, 11
9, 120, 122, 123, 123, 123, 123, 123, 124, 125, 126, 127, 128, 128, 129, 130,
131, 131, 132, 132, 134, 134, 134, 135, 135, 136, 136, 136, 137, 138, 138, 13
8, 139, 140, 144, 145, 146, 146, 146, 146, 147, 149, 150, 150, 150, 150, 151,
152, 152, 152, 153, 153, 153, 154, 154, 154, 155, 155, 156, 156, 156, 156, 15
7, 157, 157, 157, 158, 158, 159, 159, 160, 160, 160, 160, 162, 164, 166, 167,
167, 168, 170, 170, 170, 170, 171, 172, 172, 173, 173, 173, 174, 174, 175, 17
5, 176, 176, 177, 178, 179, 179, 180, 180, 180, 182, 183, 183, 183, 184, 184,
184, 185, 185, 185, 185, 186, 186, 186, 188, 189, 190, 190, 192, 192, 192, 19
6, 196, 196, 197, 197, 202, 202, 202, 203, 204, 206, 207, 209, 210, 210, 211,
212, 213, 214, 216, 219, 220, 221, 221, 222, 222, 224, 225, 225, 226, 227, 22
8, 229, 230, 231, 233, 236, 236, 237, 241, 243, 244, 245, 247, 249, 254, 254,
258, 259, 259, 261, 261, 265, 267, 268, 269, 276, 276, 290, 295, 301, 306, 31
2, 396]
```

In [123]:
```
print(min(sub))
print(max(sub))
```

```
33
396
```

In [125]:
```
# type conversion
list(marks_series)
```

Out[125]:
```
[67, 88, 89, 100, 90]
```

In [126]:
```
dict(marks_series)
```

Out[126]:
```
{'maths': 67, 'english': 88, 'science': 89, 'hindi': 100, 'social': 90}
```

In [129]:
```
# membership operator
'Hum Tumhare Hain Sanam' in movies # In oprator only searches in index values
```

Out[129]:
```
True
```

In [133]:
```
"Jack" in movies.values
```

Out[133]:
```
True
```

In [138]:
```
# looping
for i in movies:
    print(i)
```

```
Vicky Kaushal
Vicky Ahuja
Anupam Kher
Emraan Hashmi
Mona Ambegaonkar
Geetika Vidya Ohlyan
Arshad Warsi
Radhika Apte
Kangana Ranaut
Nawazuddin Siddiqui
Ali Asgar
Ranveer Singh
Prit Kamani
Ajay Devgn
Sushant Singh Rajput
Amitabh Bachchan
Abhimanyu Dasani
Talha Arshad Reshi
Nawazuddin Siddiqui
```

In [139]:
```
for i in movies.index:
    print(i)
```

```
Uri: The Surgical Strike
Battalion 609
The Accidental Prime Minister (film)
Why Cheat India
Evening Shadows
Soni (film)
Fraud Saiyaan
Bombairiya
Manikarnika: The Queen of Jhansi
Thackeray (film)
Amavas
Gully Boy
Hum Chaar
Total Dhamaal
Sonchiriya
Badla (2019 film)
Mard Ko Dard Nahi Hota
Hamid (film)
Photograph (film)
```

In [140]:
```
# Arthematic Operators (Broadcasting)
100-marks_series
```

Out[140]:
```
maths      33
english    12
science    11
hindi       0
social     10
Name: jack Marks, dtype: int64
```

In [141]:
```
100+marks_series
```

Out[141]:
```
maths      167
english    188
science    189
hindi      200
social     190
Name: jack Marks, dtype: int64
```

In [143]:
```
# Relational operators
kl>=50
```

Out[143]:
```
match_no
1      False
2      False
3      False
4      False
5      False
       ...
211    False
212    False
213     True
214    False
215    False
Name: runs, Length: 215, dtype: bool
```

Boolean Indexing on Series

In [146]:
```
# Find no of 50's and 100's scored by kohli
kl[kl>=50].size
```

Out[146]:
```
50
```

In [148]:
```
# find number of ducks
kl[kl == 0].size
```

Out[148]:
```
9
```

In [149]:
```
# Count number of day when I had more than 200 subs a day
sub[sub>=200].size
```

Out[149]:
```
59
```

In [159]:
```
# find actors who have done more than 20 movies
num_mov=movies.value_counts()
num_mov[num_mov>=20]
```

Out[159]:
```
Akshay Kumar        48
Amitabh Bachchan    45
Ajay Devgn          38
Salman Khan         31
Sanjay Dutt         26
Shah Rukh Khan      21
Emraan Hashmi       21
Name: lead, dtype: int64
```

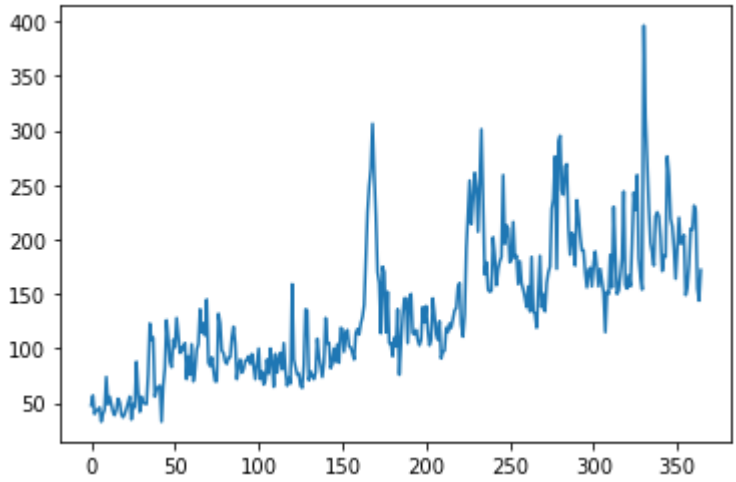
In [160]:
```
num_mov[num_mov>=20].size
```

Out[160]:
```
7
```

Plotting Graphs on Series

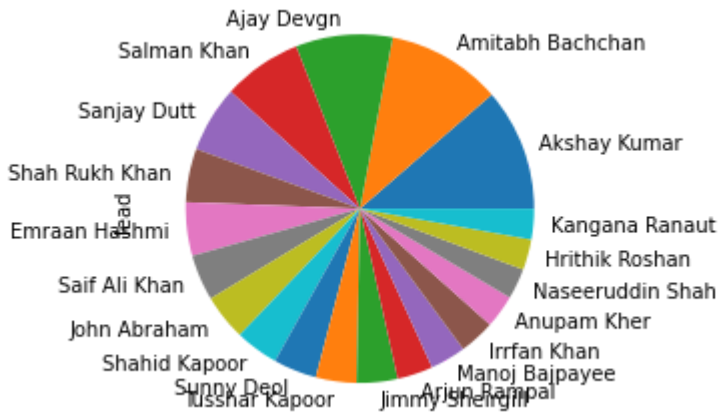
In [162]:
```
sub.plot()
```

Out[162]: <AxesSubplot:>



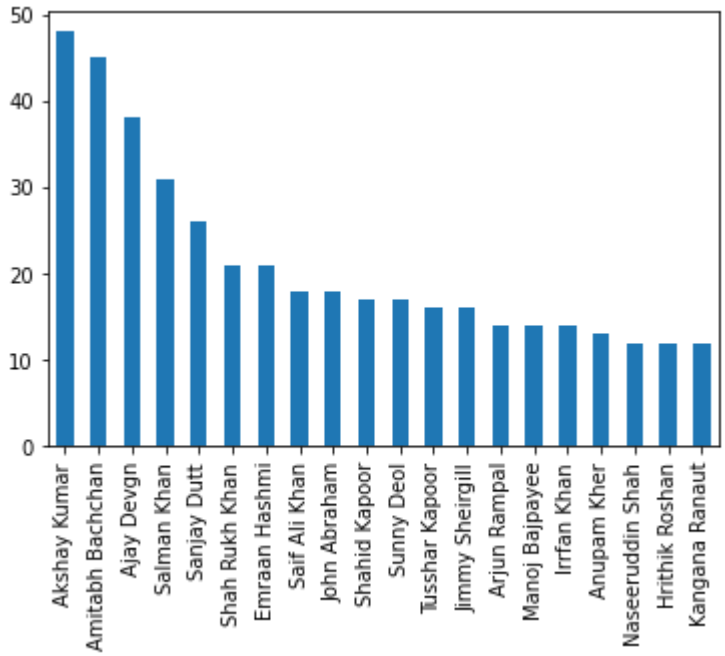
In [164]:
```
movies.value_counts().head(20).plot(kind="pie")
```

Out[164]: <AxesSubplot:ylabel='lead'>



In [165]:
```
movies.value_counts().head(20).plot(kind="bar")
```

Out[165]: <AxesSubplot:>



Some Important Series Methods

In [166]:
```
# astype
# between
# clip
# drop_duplicates
# isnull
# dropna
# fillna
# isin
# apply
# copy
```

In [175]:
```
# astype
import sys
sys.getsizeof(kl)
```

Out[175]: 11752

In [176]:
```
kl
```

Out[176]:
```
match_no
1       1
2      23
3      13
4      12
5       1
       ..
211     0
212    20
213    73
214    25
215     7
Name: runs, Length: 215, dtype: int64
```

In [177]:
```
(kl.astype("int16"))
```

Out[177]:
```
match_no
1       1
2      23
3      13
4      12
5       1
       ..
211     0
212    20
213    73
214    25
215     7
Name: runs, Length: 215, dtype: int16
```

In [178]:
```
sys.getsizeof(kl.astype("int16"))
```

Out[178]:
```
10462
```

In [181]:
```
# between
kl[kl.between(50,60)]
```

Out[181]:
```
match_no
15     50
34     58
44     56
57     57
71     51
73     58
80     57
85     56
103    51
122    52
129    54
131    54
137    55
141    58
144    57
182    50
197    51
198    53
209    58
Name: runs, dtype: int64
```

In [182]:
```
kl[kl.between(50,60)].size
```

Out[182]:
```
19
```

In [183]:
```
# clip
sub.clip(100,200)
```

Out[183]:
```
0      100
1      100
2      100
3      100
4      100
      ...
360    200
361    200
362    155
363    144
364    172
Name: Subscribers gained, Length: 365, dtype: int64
```

In [186]:
```
# drop duplicates #### drop_duplicates(): Returns a Series with duplicates rem

dele = pd.Series([1,2,33,3,3,3,1,23,33,22,33,11])
dele
```

Out[186]:
```
0      1
1      2
2     33
3      3
4      3
5      3
6      1
7     23
8     33
9     22
10    33
11    11
dtype: int64
```

In [188]:
```
dele.drop_duplicates()
```

Out[188]:
```
0      1
1      2
2     33
3      3
7     23
9     22
11    11
dtype: int64
```

In [189]:
```
dele.drop_duplicates(keep='last')
```

Out[189]:
```
1      2
5      3
6      1
7     23
9     22
10    33
11    11
dtype: int64
```

In [190]:
```
movies.drop_duplicates()
```

Out[190]:
```
movie
Uri: The Surgical Strike                   Vicky Kaushal
Battalion 609                                Vicky Ahuja
The Accidental Prime Minister (film)         Anupam Kher
Why Cheat India                            Emraan Hashmi
Evening Shadows                         Mona Ambegaonkar
                                                    ...
Rules: Pyaar Ka Superhit Formula                   Tanuja
Right Here Right Now (film)                         Ankit
Talaash: The Hunt Begins...                Rakhee Gulzar
The Pink Mirror                          Edwin Fernandes
Hum Tumhare Hain Sanam                              Jack
Name: lead, Length: 567, dtype: object
```

In [191]:
```
dele.duplicated().sum()
```

Out[191]:
```
5
```

In [193]:
```
kl.duplicated().sum()
```

Out[193]:
```
137
```

In [194]:
```
dele.count()
```

Out[194]:
```
12
```

isin(values): Returns a boolean Series indicating whether each element in the Series is in the provided values

In [198]:
```
# isnull

kl.isnull().sum()
```

Out[198]:
```
0
```

In [199]:
```
dele.isnull().sum()
```

Out[199]:
```
0
```

In [200]:
```
# dropna

dele.dropna()
```

Out[200]:
```
0      1
1      2
2     33
3      3
4      3
5      3
6      1
7     23
8     33
9     22
10    33
11    11
dtype: int64
```

In [202]:
```
# fillna

dele.fillna(0)
dele.fillna(dele.mean())
```

Out[202]:
```
0      1
1      2
2     33
3      3
4      3
5      3
6      1
7     23
8     33
9     22
10    33
11    11
dtype: int64
```

In [205]:
```
# isin

kl
```

Out[205]:
```
match_no
1       1
2      23
3      13
4      12
5       1
       ..
211     0
212    20
213    73
214    25
215     7
Name: runs, Length: 215, dtype: int64
```

In [207]:
```
kl[(kl==49) | (kl==99)]
```

Out[207]:
```
match_no
82    99
86    49
Name: runs, dtype: int64
```

In [209]:
```
kl[kl.isin([49,99])]
```

Out[209]:
```
match_no
82    99
86    49
Name: runs, dtype: int64
```

In [210]:
```
# apply

movies
```

Out[210]:
```
movie
Uri: The Surgical Strike                   Vicky Kaushal
Battalion 609                                Vicky Ahuja
The Accidental Prime Minister (film)         Anupam Kher
Why Cheat India                            Emraan Hashmi
Evening Shadows                          Mona Ambegaonkar
                                                    ...
Hum Tumhare Hain Sanam                               Jack
Aankhen (2002 film)                      Amitabh Bachchan
Saathiya (film)                              Vivek Oberoi
Company (film)                                 Ajay Devgn
Awara Paagal Deewana                         Akshay Kumar
Name: lead, Length: 1500, dtype: object
```

In [212]:
```
movies.apply(lambda x:x.split()) # split name in to two using Lambda function
```

Out[212]:
```
movie
Uri: The Surgical Strike                   [Vicky, Kaushal]
Battalion 609                                [Vicky, Ahuja]
The Accidental Prime Minister (film)         [Anupam, Kher]
Why Cheat India                            [Emraan, Hashmi]
Evening Shadows                          [Mona, Ambegaonkar]
                                                    ...
Hum Tumhare Hain Sanam                               [Jack]
Aankhen (2002 film)                      [Amitabh, Bachchan]
Saathiya (film)                              [Vivek, Oberoi]
Company (film)                                 [Ajay, Devgn]
Awara Paagal Deewana                         [Akshay, Kumar]
Name: lead, Length: 1500, dtype: object
```

In [213]:
```
movies.apply(lambda x:x.split()[0]) # select first word
```

Out[213]:
```
movie
Uri: The Surgical Strike                   Vicky
Battalion 609                              Vicky
The Accidental Prime Minister (film)     Anupam
Why Cheat India                          Emraan
Evening Shadows                            Mona
                                           ...
Hum Tumhare Hain Sanam                     Jack
Aankhen (2002 film)                      Amitabh
Saathiya (film)                            Vivek
Company (film)                             Ajay
Awara Paagal Deewana                     Akshay
Name: lead, Length: 1500, dtype: object
```

In [214]:
```
movies.apply(lambda x:x.split()[0].upper()) # Upper case
```

Out[214]:
```
movie
Uri: The Surgical Strike                   VICKY
Battalion 609                              VICKY
The Accidental Prime Minister (film)     ANUPAM
Why Cheat India                          EMRAAN
Evening Shadows                            MONA
                                           ...
Hum Tumhare Hain Sanam                     JACK
Aankhen (2002 film)                      AMITABH
Saathiya (film)                            VIVEK
Company (film)                             AJAY
Awara Paagal Deewana                     AKSHAY
Name: lead, Length: 1500, dtype: object
```

In [215]:
```
sub
```

Out[215]:
```
0       48
1       57
2       40
3       43
4       44
      ...
360    231
361    226
362    155
363    144
364    172
Name: Subscribers gained, Length: 365, dtype: int64
```

In [216]:
```
sub.mean()
```

Out[216]:
```
135.64383561643837
```

In [217]:
```
sub.apply(lambda x:'good day' if x > sub.mean() else 'bad day')
```

Out[217]:
```
0       bad day
1       bad day
2       bad day
3       bad day
4       bad day
         ...
360    good day
361    good day
362    good day
363    good day
364    good day
Name: Subscribers gained, Length: 365, dtype: object
```

In [229]:
```
# Copy

kl
```

Out[229]:
```
match_no
1       1
2      23
3      13
4      12
5       1
       ..
211     0
212    20
213    73
214    25
215     7
Name: runs, Length: 215, dtype: int64
```

In [230]:
```
new = kl.head()
```

In [231]:
```
new[1]=100
```

In [232]:
```
new
```

Out[232]:
```
match_no
1    100
2     23
3     13
4     12
5      1
Name: runs, dtype: int64
```

In [233]:
```
kl
```

Out[233]:
```
match_no
1      100
2       23
3       13
4       12
5        1
       ...
211      0
212     20
213     73
214     25
215      7
Name: runs, Length: 215, dtype: int64
```

In [240]:
```
new = kl.head(5).copy()
```

In [241]:
```
new[1]=20
```

In [242]:
```
new
```

Out[242]:
```
match_no
1    20
2    23
3    13
4    12
5     1
Name: runs, dtype: int64
```

In [250]:
```
kl
```

Out[250]:
```
match_no
1      100
2       23
3       13
4       12
5        1
       ...
211      0
212     20
213     73
214     25
215      7
Name: runs, Length: 215, dtype: int64
```

In [ ]:

In [ ]:

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

Creating DataFrame

In [2]:
```
# Using the lists
student_data = [
    [100,80,10],
    [90,70,7],
    [120,100,14],
    [80,50,2]
]

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

Out[2]:

	iq	marks	package
0	100	80	10
1	90	70	7
2	120	100	14
3	80	50	2

In [3]:
```
# using dicts

student_dict = {
    'name':['peter','saint','noeum','parle','samme','dave'],
    'iq':[100,90,120,80,13,90],
    'marks':[80,70,100,50,11,80],
    'package':[10,7,14,2,15,100]
}
students=pd.DataFrame(student_dict)
students
```

Out[3]:

	name	iq	marks	package
0	peter	100	80	10
1	saint	90	70	7
2	noeum	120	100	14
3	parle	80	50	2
4	samme	13	11	15
5	dave	90	80	100

In [4]:
```
students.set_index('name',inplace=True)
students
```

Out[4]:

	iq	marks	package
name			
peter	100	80	10
saint	90	70	7
noeum	120	100	14
parle	80	50	2
samme	13	11	15
dave	90	80	100

In [5]:
```
# Read csv
movies = pd.read_csv("movies.csv")
movies.head()
```

Out[5]:

	title_x	imdb_id	poster_path	wiki_link	title_y	original_title	is_adult	year_
0	Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Uri:_The_Surgica...	Uri: The Surgical Strike	Uri: The Surgical Strike	0	
1	Battalion 609	tt9472208	NaN	https://en.wikipedia.org/wiki/Battalion_609	Battalion 609	Battalion 609	0	
2	The Accidental Prime Minister (film)	tt6986710	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/The_Accidental_P...	The Accidental Prime Minister	The Accidental Prime Minister	0	
3	Why Cheat India	tt8108208	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Why_Cheat_India	Why Cheat India	Why Cheat India	0	
4	Evening Shadows	tt6028796	NaN	https://en.wikipedia.org/wiki/Evening_Shadows	Evening Shadows	Evening Shadows	0	

In [6]:
```
ipl = pd.read_csv("ipl-matches.csv")
ipl.head()
```

Out[6]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecision	SuperOver	WinningTeam	W
0	1312200	Ahmedabad	2022-05-29	2022	Final	Rajasthan Royals	Gujarat Titans	Narendra Modi Stadium, Ahmedabad	Rajasthan Royals	bat	N	Gujarat Titans	Wi
1	1312199	Ahmedabad	2022-05-27	2022	Qualifier 2	Royal Challengers Bangalore	Rajasthan Royals	Narendra Modi Stadium, Ahmedabad	Rajasthan Royals	field	N	Rajasthan Royals	Wi
2	1312198	Kolkata	2022-05-25	2022	Eliminator	Royal Challengers Bangalore	Lucknow Super Giants	Eden Gardens, Kolkata	Lucknow Super Giants	field	N	Royal Challengers Bangalore	
3	1312197	Kolkata	2022-05-24	2022	Qualifier 1	Rajasthan Royals	Gujarat Titans	Eden Gardens,	Gujarat Titans	field	N	Gujarat Titans	Wi

DataFrame Attributes and Methods

In [7]:
```
# shape
movies.shape
```

Out[7]: (1629, 18)

In [8]:
```
ipl.shape
```

Out[8]: (950, 20)

In [9]:
```
# dtype
movies.dtypes
```

Out[9]:
```
title_x              object
imdb_id              object
poster_path          object
wiki_link            object
title_y              object
original_title       object
is_adult              int64
year_of_release       int64
runtime              object
genres               object
imdb_rating         float64
imdb_votes            int64
story                object
summary              object
tagline              object
actors               object
wins_nominations     object
release_date         object
dtype: object
```

In [10]:
```
ipl.dtypes
```

Out[10]:
```
ID                  int64
City               object
Date               object
Season             object
MatchNumber        object
Team1              object
Team2              object
Venue              object
TossWinner         object
TossDecision       object
SuperOver          object
WinningTeam        object
WonBy              object
Margin            float64
method             object
Player_of_Match    object
Team1Players       object
Team2Players       object
Umpire1            object
Umpire2            object
dtype: object
```

In [11]:
```
# index
movies.index
```

Out[11]:
```
RangeIndex(start=0, stop=1629, step=1)
```

In [12]:
```
ipl.index
```

Out[12]:
```
RangeIndex(start=0, stop=950, step=1)
```

In [13]:
```
# Columns
movies.columns
```

Out[13]:
```
Index(['title_x', 'imdb_id', 'poster_path', 'wiki_link', 'title_y',
       'original_title', 'is_adult', 'year_of_release', 'runtime', 'genres',
       'imdb_rating', 'imdb_votes', 'story', 'summary', 'tagline', 'actors',
       'wins_nominations', 'release_date'],
      dtype='object')
```

In [14]:
```
ipl.columns
```

Out[14]:
```
Index(['ID', 'City', 'Date', 'Season', 'MatchNumber', 'Team1', 'Team2',
       'Venue', 'TossWinner', 'TossDecision', 'SuperOver', 'WinningTeam',
       'WonBy', 'Margin', 'method', 'Player_of_Match', 'Team1Players',
       'Team2Players', 'Umpire1', 'Umpire2'],
      dtype='object')
```

In [15]:
```
# Values
students.values
```

Out[15]:
```
array([[100,  80,  10],
       [ 90,  70,   7],
       [120, 100,  14],
       [ 80,  50,   2],
       [ 13,  11,  15],
       [ 90,  80, 100]], dtype=int64)
```

In [16]:
```
ipl.values
```

Out[16]:
```
array([[1312200, 'Ahmedabad', '2022-05-29', ...,
        "['WP Saha', 'Shubman Gill', 'MS Wade', 'HH Pandya', 'DA Miller', 'R Tewatia', 'Rashid Khan', 'R Sai Kishore',
'LH Ferguson', 'Yash Dayal', 'Mohammed Shami']",
        'CB Gaffaney', 'Nitin Menon'],
       [1312199, 'Ahmedabad', '2022-05-27', ...,
        "['YBK Jaiswal', 'JC Buttler', 'SV Samson', 'D Padikkal', 'SO Hetmyer', 'R Parag', 'R Ashwin', 'TA Boult', 'YS C
hahal', 'M Prasidh Krishna', 'OC McCoy']",
        'CB Gaffaney', 'Nitin Menon'],
       [1312198, 'Kolkata', '2022-05-25', ...,
        "['Q de Kock', 'KL Rahul', 'M Vohra', 'DJ Hooda', 'MP Stoinis', 'E Lewis', 'KH Pandya', 'PVD Chameera', 'Mohsin
Khan', 'Avesh Khan', 'Ravi Bishnoi']",
        'J Madanagopal', 'MA Gough'],
       ...,
       [335984, 'Delhi', '2008-04-19', ...,
        "['T Kohli', 'YK Pathan', 'SR Watson', 'M Kaif', 'DS Lehmann', 'RA Jadeja', 'M Rawat', 'D Salunkhe', 'SK Warne',
'SK Trivedi', 'MM Patel']",
        'Aleem Dar', 'GA Pratapkumar'],
       [335983, 'Chandigarh', '2008-04-19', ...,
        "['PA Patel', 'ML Hayden', 'MEK Hussey', 'MS Dhoni', 'SK Raina', 'JDP Oram', 'S Badrinath', 'Joginder Sharma',
'P Amarnath', 'MS Gony', 'M Muralitharan']",
        'MR Benson', 'SL Shastri'],
       [335982, 'Bangalore', '2008-04-18', ...,
        "['SC Ganguly', 'BB McCullum', 'RT Ponting', 'DJ Hussey', 'Mohammad Hafeez', 'LR Shukla', 'WP Saha', 'AB Agarka
r', 'AB Dinda', 'M Kartik', 'I Sharma']",
        'Asad Rauf', 'RE Koertzen']], dtype=object)
```

In [17]:
```
# Sample -> to reduce bias
ipl.sample(2)
```

Out[17]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecision	SuperOver	WinningTeam	WonBy	Ma
226	1178402	Mumbai	2019-04-13	2019	27	Mumbai Indians	Rajasthan Royals	Wankhede Stadium	Rajasthan Royals	field	N	Rajasthan Royals	Wickets	
63	1304057	Mumbai	2022-04-03	2022	11	Punjab Kings	Chennai Super Kings	Brabourne Stadium, Mumbai	Chennai Super Kings	field	N	Punjab Kings	Runs	

In [18]:
```
# info
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1629 entries, 0 to 1628
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   title_x           1629 non-null   object 
 1   imdb_id           1629 non-null   object 
 2   poster_path       1526 non-null   object 
 3   wiki_link         1629 non-null   object 
 4   title_y           1629 non-null   object 
 5   original_title    1629 non-null   object 
 6   is_adult          1629 non-null   int64  
 7   year_of_release   1629 non-null   int64  
 8   runtime           1629 non-null   object 
 9   genres            1629 non-null   object 
 10  imdb_rating       1629 non-null   float64
 11  imdb_votes        1629 non-null   int64  
 12  story             1609 non-null   object 
 13  summary           1629 non-null   object 
 14  tagline           557 non-null    object 
 15  actors            1624 non-null   object 
 16  wins_nominations  707 non-null    object 
 17  release_date      1522 non-null   object 
dtypes: float64(1), int64(3), object(14)
memory usage: 229.2+ KB
```

In [19]:
```
ipl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 950 entries, 0 to 949
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               950 non-null    int64  
 1   City             899 non-null    object 
 2   Date             950 non-null    object 
 3   Season           950 non-null    object 
 4   MatchNumber      950 non-null    object 
 5   Team1            950 non-null    object 
 6   Team2            950 non-null    object 
 7   Venue            950 non-null    object 
 8   TossWinner       950 non-null    object 
 9   TossDecision     950 non-null    object 
 10  SuperOver        946 non-null    object 
 11  WinningTeam      946 non-null    object 
 12  WonBy            950 non-null    object 
 13  Margin           932 non-null    float64
 14  method           19 non-null     object 
 15  Player_of_Match  946 non-null    object 
 16  Team1Players     950 non-null    object 
 17  Team2Players     950 non-null    object 
 18  Umpire1          950 non-null    object 
 19  Umpire2          950 non-null    object 
dtypes: float64(1), int64(1), object(18)
memory usage: 148.6+ KB
```

In [20]:
```
# describe
movies.describe()
```

Out[20]:

	is_adult	year_of_release	imdb_rating	imdb_votes
count	1629.0	1629.000000	1629.000000	1629.000000
mean	0.0	2010.263966	5.557459	5384.263352
std	0.0	5.381542	1.567609	14552.103231
min	0.0	2001.000000	0.000000	0.000000
25%	0.0	2005.000000	4.400000	233.000000
50%	0.0	2011.000000	5.600000	1000.000000
75%	0.0	2015.000000	6.800000	4287.000000
max	0.0	2019.000000	9.400000	310481.000000

In [21]:
```
ipl.describe()
```

Out[21]:

	ID	Margin
count	9.500000e+02	932.000000
mean	8.304852e+05	17.056867
std	3.375678e+05	21.633109
min	3.359820e+05	1.000000
25%	5.012612e+05	6.000000
50%	8.297380e+05	8.000000
75%	1.175372e+06	19.000000
max	1.312200e+06	146.000000

In [22]:
```
# isnull
movies.isnull()
```

Out[22]:

	title_x	imdb_id	poster_path	wiki_link	title_y	original_title	is_adult	year_of_release	runtime	genres	imdb_rating	imdb_votes	story	summa
0	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
1	False	False	True	False	False	False	False	False	False	False	False	False	False	Fals
2	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
3	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
4	False	False	True	False	False	False	False	False	False	False	False	False	False	Fals
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1624	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
1625	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
1626	False	False	True	False	False	False	False	False	False	False	False	False	False	Fals
1627	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
1628	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals

1629 rows × 18 columns

In [23]:
```
movies.isnull().sum()
```

Out[23]:
```
title_x               0
imdb_id               0
poster_path         103
wiki_link             0
title_y               0
original_title        0
is_adult              0
year_of_release       0
runtime               0
genres                0
imdb_rating           0
imdb_votes            0
story                20
summary               0
tagline            1072
actors                5
wins_nominations    922
release_date        107
dtype: int64
```

In [24]:
```
# duplicated
movies.duplicated().sum()
```

Out[24]:
```
0
```

In [25]:
```
# rename
students
```

Out[25]:

name	iq	marks	package
peter	100	80	10
saint	90	70	7
noeum	120	100	14
parle	80	50	2
samme	13	11	15
dave	90	80	100

In [26]:
```
students.rename(columns={'marks':'percent','package':'lpa'},inplace=True)
```

In [ ]:
```
students.drop(columns='name',inplace=True)
```

Maths Method

In [28]:
```
# sum -> Axis Argument
students.sum(axis=1)
```

Out[28]:
```
name
peter    190
saint    167
noeum    234
parle    132
samme     39
dave     270
dtype: int64
```

In [29]:
```
# mean
students.mean()
```

Out[29]:
```
iq         82.166667
percent    65.166667
lpa        24.666667
dtype: float64
```

In [30]:
```
students.min(axis=1)
```

Out[30]:
```
name
peter    10
saint     7
noeum    14
parle     2
samme    11
dave     80
dtype: int64
```

In [31]:
```
students.var()
```

Out[31]:
```
iq         1332.166667
percent     968.166667
lpa        1384.666667
dtype: float64
```

Selecting cols from a DataFrame

In [32]:
```
# single cols
movies['title_x']
```

Out[32]:
```
0                   Uri: The Surgical Strike
1                           Battalion 609
2       The Accidental Prime Minister (film)
3                           Why Cheat India
4                           Evening Shadows
                        ...
1624                  Tera Mera Saath Rahen
1625                   Yeh Zindagi Ka Safar
1626                        Sabse Bada Sukh
1627                                  Daaka
1628                               Humsafar
Name: title_x, Length: 1629, dtype: object
```

In [33]:
```
type(movies['title_x'])
```

Out[33]:
```
pandas.core.series.Series
```

In [137]:
```
# multiple columns
movies[['year_of_release',' actors','title_x']].head(2)
```

In [35]:
```
type(movies[['year_of_release','actors','title_x']].head(2))
```

Out[35]:
```
pandas.core.frame.DataFrame
```

In [36]:
```
ipl[['City','Team1','Team2' ]]
```

Out[36]:

	City	Team1	Team2
0	Ahmedabad	Rajasthan Royals	Gujarat Titans
1	Ahmedabad	Royal Challengers Bangalore	Rajasthan Royals
2	Kolkata	Royal Challengers Bangalore	Lucknow Super Giants
3	Kolkata	Rajasthan Royals	Gujarat Titans
4	Mumbai	Sunrisers Hyderabad	Punjab Kings
...	...	...	...
945	Kolkata	Kolkata Knight Riders	Deccan Chargers
946	Mumbai	Mumbai Indians	Royal Challengers Bangalore
947	Delhi	Delhi Daredevils	Rajasthan Royals
948	Chandigarh	Kings XI Punjab	Chennai Super Kings
949	Bangalore	Royal Challengers Bangalore	Kolkata Knight Riders

950 rows × 3 columns

In [37]:
```
student_dict = {
    'name':['peter','saint','noeum','parle','samme','dave'],
    'iq':[100,90,120,80,13,90],
    'marks':[80,70,100,50,11,80],
    'package':[10,7,14,2,15,100]
}
students=pd.DataFrame(student_dict)
students.set_index('name',inplace=True)
```

In [38]:
```
students
```

Out[38]:

name	iq	marks	package
peter	100	80	10
saint	90	70	7
noeum	120	100	14
parle	80	50	2
samme	13	11	15
dave	90	80	100

Selecting rows from a DataFrame

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

In [39]:
```
# single_row
movies.iloc[1]
```

Out[39]:
```
title_x                                                Battalion 609
imdb_id                                                    tt9472208
poster_path                                                      NaN
wiki_link           https://en.wikipedia.org/wiki/Battalion_609 (https://en.wikipedia.org/wiki/Battalion_609)
title_y                                                Battalion 609
original_title                                         Battalion 609
is_adult                                                           0
year_of_release                                                 2019
runtime                                                          131
genres                                                           War
imdb_rating                                                      4.1
imdb_votes                                                        73
story               The story revolves around a cricket match betw...
summary             The story of Battalion 609 revolves around a c...
tagline                                                          NaN
actors              Vicky Ahuja|Shoaib Ibrahim|Shrikant Kamat|Elen...
wins_nominations                                                 NaN
release_date                                  11 January 2019 (India)
Name: 1, dtype: object
```

In [40]:
```
# Multiple rows
movies.iloc[5:10]
```

Out[40]:

	title_x	imdb_id	poster_path	wiki_link	title_y	original_title	is_adult	y
5	Soni (film)	tt6078866	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Soni_(film)	Soni	Soni	0	
6	Fraud Saiyaan	tt5013008	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Fraud_Saiyaan	Fraud Saiyaan	Fraud Saiyyan	0	
7	Bombairiya	tt4971258	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Bombairiya	Bombairiya	Bombairiya	0	
8	Manikarnika: The Queen of Jhansi	tt6903440	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Manikarnika:_The...	Manikarnika: The Queen of Jhansi	Manikarnika: The Queen of Jhansi	0	
9	Thackeray (film)	tt7777196	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Thackeray_(film)	Thackeray	Thackeray	0	

In [41]:
```
movies.iloc[5:12:2]
```

Out[41]:

	title_x	imdb_id	poster_path	wiki_link	title_y	original_title	is_adult	year_
5	Soni (film)	tt6078866	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Soni_(film)	Soni	Soni	0	
7	Bombairiya	tt4971258	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Bombairiya	Bombairiya	Bombairiya	0	
9	Thackeray (film)	tt7777196	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Thackeray_(film)	Thackeray	Thackeray	0	
11	Gully Boy	tt2395469	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Gully_Boy	Gully Boy	Gully Boy	0	

In [42]:
```
# Fancy indexing
ipl.iloc[[0,4,5]]
```

Out[42]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecision	SuperOver	WinningTeam	WonBy
0	1312200	Ahmedabad	2022-05-29	2022	Final	Rajasthan Royals	Gujarat Titans	Narendra Modi Stadium, Ahmedabad	Rajasthan Royals	bat	N	Gujarat Titans	Wickets
4	1304116	Mumbai	2022-05-22	2022	70	Sunrisers Hyderabad	Punjab Kings	Wankhede Stadium, Mumbai	Sunrisers Hyderabad	bat	N	Punjab Kings	Wickets
5	1304115	Mumbai	2022-05-21	2022	69	Delhi Capitals	Mumbai Indians	Wankhede Stadium, Mumbai	Mumbai Indians	field	N	Mumbai Indians	Wickets

In [43]:
```
# loc ( Location)
students
```

Out[43]:

name	iq	marks	package
peter	100	80	10
saint	90	70	7
noeum	120	100	14
parle	80	50	2
samme	13	11	15
dave	90	80	100

In [44]:
```
students.loc['parle']
```

Out[44]:
```
iq         80
marks      50
package     2
Name: parle, dtype: int64
```

In [45]:
```
students.loc['saint':'samme':2]
```

Out[45]:

name	iq	marks	package
saint	90	70	7
parle	80	50	2

In [46]:
```
# Fancy indexing
students.loc[['saint','dave']]
```

Out[46]:

name	iq	marks	package
saint	90	70	7
dave	90	80	100

In [47]:
```
students.iloc[[0,4,3]]
```

Out[47]:

name	iq	marks	package
peter	100	80	10
samme	13	11	15
parle	80	50	2

Selecting both rows and cols

In [48]:
```
movies.iloc[0:3,0:3]
```

Out[48]:

	title_x	imdb_id	poster_path
0	Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...
1	Battalion 609	tt9472208	NaN
2	The Accidental Prime Minister (film)	tt6986710	https://upload.wikimedia.org/wikipedia/en/thum...

In [49]:
```
movies.loc[0:2,'title_x':'poster_path']
```

Out[49]:

	title_x	imdb_id	poster_path
0	Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...
1	Battalion 609	tt9472208	NaN
2	The Accidental Prime Minister (film)	tt6986710	https://upload.wikimedia.org/wikipedia/en/thum...

Filtering a DataFrame

In [50]:
```
ipl.head(2)
```

Out[50]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecision	SuperOver	WinningTeam	WonB
0	1312200	Ahmedabad	2022-05-29	2022	Final	Rajasthan Royals	Gujarat Titans	Narendra Modi Stadium, Ahmedabad	Rajasthan Royals	bat	N	Gujarat Titans	Wicke
1	1312199	Ahmedabad	2022-05-27	2022	Qualifier 2	Royal Challengers Bangalore	Rajasthan Royals	Narendra Modi Stadium, Ahmedabad	Rajasthan Royals	field	N	Rajasthan Royals	Wicke

In [51]:
```
# find all the final winners

mask=ipl['MatchNumber'] == 'Final'
new_df= ipl[mask]
new_df[['Season','WinningTeam']]
```

Out[51]:

	Season	WinningTeam
0	2022	Gujarat Titans
74	2021	Chennai Super Kings
134	2020/21	Mumbai Indians
194	2019	Mumbai Indians
254	2018	Chennai Super Kings
314	2017	Mumbai Indians
373	2016	Sunrisers Hyderabad
433	2015	Mumbai Indians
492	2014	Kolkata Knight Riders
552	2013	Mumbai Indians
628	2012	Kolkata Knight Riders
702	2011	Chennai Super Kings
775	2009/10	Chennai Super Kings
835	2009	Deccan Chargers
892	2007/08	Rajasthan Royals

In [52]:
```
# In one line
ipl[ipl['MatchNumber']=='Final'][['Season','WinningTeam']]
```

Out[52]:

	Season	WinningTeam
0	2022	Gujarat Titans
74	2021	Chennai Super Kings
134	2020/21	Mumbai Indians
194	2019	Mumbai Indians
254	2018	Chennai Super Kings
314	2017	Mumbai Indians
373	2016	Sunrisers Hyderabad
433	2015	Mumbai Indians
492	2014	Kolkata Knight Riders
552	2013	Mumbai Indians
628	2012	Kolkata Knight Riders
702	2011	Chennai Super Kings
775	2009/10	Chennai Super Kings
835	2009	Deccan Chargers
892	2007/08	Rajasthan Royals

In [53]:
```
# how many super over finishes have occured
ipl.head(2)
```

Out[53]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecision	SuperOver	WinningTeam	WonB
0	1312200	Ahmedabad	2022-05-29	2022	Final	Rajasthan Royals	Gujarat Titans	Narendra Modi Stadium, Ahmedabad	Rajasthan Royals	bat	N	Gujarat Titans	Wicke
1	1312199	Ahmedabad	2022-05-27	2022	Qualifier 2	Royal Challengers Bangalore	Rajasthan Royals	Narendra Modi Stadium, Ahmedabad	Rajasthan Royals	field	N	Rajasthan Royals	Wicke

In [54]:
```
ipl[ipl['SuperOver']=='Y'].shape[0]
```

Out[54]: 14

In [55]:
```
# how many matches has csk won in kolkata
ipl.sample(2)
```

Out[55]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecision	SuperOver	WinningTeam	Wo
364	1082599	Pune	2017-04-11	2017	9	Rising Pune Supergiant	Delhi Daredevils	Maharashtra Cricket Association Stadium	Rising Pune Supergiant	field	N	Delhi Daredevils	
376	981013	Bangalore	2016-05-24	2016	Qualifier 1	Gujarat Lions	Royal Challengers Bangalore	M Chinnaswamy Stadium	Royal Challengers Bangalore	field	N	Royal Challengers Bangalore	Wi

In [56]:
```
ipl[(ipl['City'] == 'Kolkata') & (ipl['WinningTeam'] == 'Chennai Super Kings')].shape[0]
```

Out[56]: 5

In [57]:
```
# toss winner is match winner in percentage
ipl.sample(2)
```

Out[57]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecision	SuperOver	WinningTeam	WonBy	Ma
168	1216529	Abu Dhabi	2020-10-11	2020/21	27	Delhi Capitals	Mumbai Indians	Sheikh Zayed Stadium	Delhi Capitals	bat	N	Mumbai Indians	Wickets	
127	1254064	Mumbai	2021-04-15	2021	7	Delhi Capitals	Rajasthan Royals	Wankhede Stadium, Mumbai	Rajasthan Royals	field	N	Rajasthan Royals	Wickets	

In [58]:
```
(ipl[(ipl['TossWinner']== ipl['WinningTeam'])].shape[0]/ipl.shape[0])*100
```

Out[58]: 51.473684210526315

In [59]:
```
# movies with rating higher than 8 and votes>10000
movies.sample(2)
```

Out[59]:

	title_x	imdb_id	poster_path	wiki_link	title_y	original_title	is_adult	year_of
24	Junglee (2019 film)	tt7463730	https://upload.wikimedia.org/wikipedia/en/e/e2...	https://en.wikipedia.org/wiki/Junglee_(2019_film)	Junglee	Junglee	0	
390	Hey Bro	tt4512230	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Hey_Bro	Hey Bro	Hey Bro	0	

In [60]:
```
movies[(movies['imdb_rating'] > 8) & (movies['imdb_votes'] > 10000)].shape[0]
```

Out[60]: 43

In [61]:
```
# Action movies with rating higher than 7.5
#mask1=movies['genres'].str.split('|').apply(lambda x:'Action' in x)
mask1=movies['genres'].str.contains('Action')
mask2=movies['imdb_rating']>7.5
movies[mask1 & mask2]
```

Out[61]:

	title_x	imdb_id	poster_path	wiki_link	title_y	original_tit
0	Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Uri:_The_Surgica...	Uri: The Surgical Strike	Uri: Th Surgical Strik
41	Family of Thakurganj	tt8897986	https://upload.wikimedia.org/wikipedia/en/9/99...	https://en.wikipedia.org/wiki/Family_of_Thakur...	Family of Thakurganj	Family Thakurga
84	Mukkabaaz	tt7180544	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Mukkabaaz	The Brawler	Mukkabaa
106	Raazi	tt7098658	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Raazi	Raazi	Raa

Adding new cols

In [62]:
```
movies['country']='India'
movies.sample(2)
```

Out[62]:

	title_x	imdb_id	poster_path	wiki_link	title_y	original_title	is_adult	year_of
915	Wanted (2009 film)	tt1084972	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Wanted_(2009_film)	Wanted	Wanted	0	
241	Shaadi Mein Zaroor Aana	tt7469726	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Shaadi_Mein_Zaro...	Shaadi Mein Zaroor Aana	Shaadi Mein Zaroor Aana	0	

In [63]:
```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1629 entries, 0 to 1628
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   title_x           1629 non-null   object 
 1   imdb_id           1629 non-null   object 
 2   poster_path       1526 non-null   object 
 3   wiki_link         1629 non-null   object 
 4   title_y           1629 non-null   object 
 5   original_title    1629 non-null   object 
 6   is_adult          1629 non-null   int64  
 7   year_of_release   1629 non-null   int64  
 8   runtime           1629 non-null   object 
 9   genres            1629 non-null   object 
 10  imdb_rating       1629 non-null   float64
 11  imdb_votes        1629 non-null   int64  
 12  story             1609 non-null   object 
 13  summary           1629 non-null   object 
 14  tagline           557 non-null    object 
 15  actors            1624 non-null   object 
 16  wins_nominations  707 non-null    object 
 17  release_date      1522 non-null   object 
 18  country           1629 non-null   object 
dtypes: float64(1), int64(3), object(15)
memory usage: 241.9+ KB
```

In [138]:
```
# From Existing ones
movies['actors'].str.split('|').apply(lambda x:x[0])
```

In [139]:
```
 # From Existing ones
movies['lead actor']= movies['actors'].str.split('|').apply(lambda x:x[0])
movies
```

Important DataFrame Functions

In [68]:
```
ipl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 950 entries, 0 to 949
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               950 non-null    int64  
 1   City             899 non-null    object 
 2   Date             950 non-null    object 
 3   Season           950 non-null    object 
 4   MatchNumber      950 non-null    object 
 5   Team1            950 non-null    object 
 6   Team2            950 non-null    object 
 7   Venue            950 non-null    object 
 8   TossWinner       950 non-null    object 
 9   TossDecision     950 non-null    object 
 10  SuperOver        946 non-null    object 
 11  WinningTeam      946 non-null    object 
 12  WonBy            950 non-null    object 
 13  Margin           932 non-null    float64
 14  method           19 non-null     object 
 15  Player_of_Match  946 non-null    object 
 16  Team1Players     950 non-null    object 
 17  Team2Players     950 non-null    object 
 18  Umpire1          950 non-null    object 
 19  Umpire2          950 non-null    object 
dtypes: float64(1), int64(1), object(18)
memory usage: 148.6+ KB
```

In [69]:
```
ipl['ID']=ipl['ID'].astype('int32')
```

In [70]:
```
ipl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 950 entries, 0 to 949
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               950 non-null    int32  
 1   City             899 non-null    object 
 2   Date             950 non-null    object 
 3   Season           950 non-null    object 
 4   MatchNumber      950 non-null    object 
 5   Team1            950 non-null    object 
 6   Team2            950 non-null    object 
 7   Venue            950 non-null    object 
 8   TossWinner       950 non-null    object 
 9   TossDecision     950 non-null    object 
 10  SuperOver        946 non-null    object 
 11  WinningTeam      946 non-null    object 
 12  WonBy            950 non-null    object 
 13  Margin           932 non-null    float64
 14  method           19 non-null     object 
 15  Player_of_Match  946 non-null    object 
 16  Team1Players     950 non-null    object 
 17  Team2Players     950 non-null    object 
 18  Umpire1          950 non-null    object 
 19  Umpire2          950 non-null    object 
dtypes: float64(1), int32(1), object(18)
memory usage: 144.9+ KB
```

In [71]:
```
ipl['Season'] = ipl['Season'].astype('category')
```

In [72]:
```
ipl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 950 entries, 0 to 949
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ID               950 non-null    int32
 1   City             899 non-null    object
 2   Date             950 non-null    object
 3   Season           950 non-null    category
 4   MatchNumber      950 non-null    object
 5   Team1            950 non-null    object
 6   Team2            950 non-null    object
 7   Venue            950 non-null    object
 8   TossWinner       950 non-null    object
 9   TossDecision     950 non-null    object
 10  SuperOver        946 non-null    object
 11  WinningTeam      946 non-null    object
 12  WonBy            950 non-null    object
 13  Margin           932 non-null    float64
 14  method           19 non-null     object
 15  Player_of_Match  946 non-null    object
 16  Team1Players     950 non-null    object
 17  Team2Players     950 non-null    object
 18  Umpire1          950 non-null    object
 19  Umpire2          950 non-null    object
dtypes: category(1), float64(1), int32(1), object(17)
memory usage: 139.0+ KB
```

In [73]:
```
ipl['Team1'] = ipl['Team1'].astype('category')
ipl['Team2'] = ipl['Team2'].astype('category')
```

In [74]:
```
ipl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 950 entries, 0 to 949
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ID               950 non-null    int32
 1   City             899 non-null    object
 2   Date             950 non-null    object
 3   Season           950 non-null    category
 4   MatchNumber      950 non-null    object
 5   Team1            950 non-null    category
 6   Team2            950 non-null    category
 7   Venue            950 non-null    object
 8   TossWinner       950 non-null    object
 9   TossDecision     950 non-null    object
 10  SuperOver        946 non-null    object
 11  WinningTeam      946 non-null    object
 12  WonBy            950 non-null    object
 13  Margin           932 non-null    float64
 14  method           19 non-null     object
 15  Player_of_Match  946 non-null    object
 16  Team1Players     950 non-null    object
 17  Team2Players     950 non-null    object
 18  Umpire1          950 non-null    object
 19  Umpire2          950 non-null    object
dtypes: category(3), float64(1), int32(1), object(15)
memory usage: 127.4+ KB
```

More Important Functions

Value Counts

In [143]:
```
# value_counts(series and dataframe)


marks = pd.DataFrame([
    [100,80,10],
    [90,70,7],
    [120,100,14],
    [80,70,14],
    [80,70,14]
],columns=['iq','marks','package'])

marks
```

Out[143]:

	iq	marks	package
0	100	80	10
1	90	70	7
2	120	100	14
3	80	70	14
4	80	70	14

In [76]:
```
marks.value_counts()
```

Out[76]:
```
iq   marks  package
80   70     14         2
90   70     7          1
100  80     10         1
120  100    14         1
dtype: int64
```

In [77]:
```
# find which player has won most potm -> in finals and qualifiers
ipl.sample(2)
```

Out[77]:

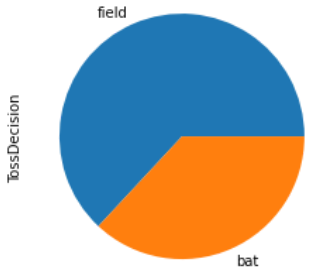
	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecision	SuperOver	WinningTeam	WonBy
304	1136570	Kolkata	2018-04-14	2018	10	Kolkata Knight Riders	Sunrisers Hyderabad	Eden Gardens	Sunrisers Hyderabad	field	N	Sunrisers Hyderabad	Wickets
561	598028	Dharamsala	2013-05-16	2013	67	Kings XI Punjab	Delhi Daredevils	Himachal Pradesh Cricket Association Stadium	Delhi Daredevils	field	N	Kings XI Punjab	Runs

In [78]:
```
ipl[~ipl['MatchNumber'].str.isdigit()]['Player_of_Match'].value_counts() # To reverse the contet use tilt ~
```

Out[78]:
```
KA Pollard           3
F du Plessis         3
SK Raina             3
A Kumble             2
MK Pandey            2
YK Pathan            2
M Vijay              2
JJ Bumrah            2
AB de Villiers       2
SR Watson            2
HH Pandya            1
Harbhajan Singh      1
A Nehra              1
V Sehwag             1
UT Yadav             1
MS Bisla             1
BJ Hodge             1
MEK Hussey           1
MS Dhoni             1
CH Gayle             1
MM Patel             1
DE Bollinger         1
AC Gilchrist         1
RG Sharma            1
DA Warner            1
MC Henriques         1
JC Buttler           1
RM Patidar           1
DA Miller            1
VR Iyer              1
SP Narine            1
RD Gaikwad           1
TA Boult             1
MP Stoinis           1
KS Williamson        1
RR Pant              1
SA Yadav             1
Rashid Khan          1
AD Russell           1
KH Pandya            1
KV Sharma            1
NM Coulter-Nile      1
Washington Sundar    1
BCJ Cutting          1
M Ntini              1
Name: Player_of_Match, dtype: int64
```

In [79]:
```
# Toss decision plot
ipl['TossDecision'].value_counts().plot(kind='pie')
```

Out[79]:
```
<AxesSubplot:ylabel='TossDecision'>
```



In [80]:
```
# No.of matches each team has played
(ipl['Team1'].value_counts() + ipl['Team2'].value_counts()).sort_values(ascending=False)
```

Out[80]:
```
Mumbai Indians                 231
Royal Challengers Bangalore    226
Kolkata Knight Riders          223
Chennai Super Kings            208
Rajasthan Royals               192
Kings XI Punjab                190
Delhi Daredevils               161
Sunrisers Hyderabad            152
Deccan Chargers                 75
Delhi Capitals                  63
Pune Warriors                   46
Gujarat Lions                   30
Punjab Kings                    28
Gujarat Titans                  16
Rising Pune Supergiant          16
Lucknow Super Giants            15
Kochi Tuskers Kerala            14
Rising Pune Supergiants         14
dtype: int64
```

Sort values

In [81]:
```
x = pd.Series([12,14,1,56,89])
x
```

Out[81]:
```
0    12
1    14
2     1
3    56
4    89
dtype: int64
```

In [82]:
```
x.sort_values(ascending=True)
```

Out[82]:
```
2     1
0    12
1    14
3    56
4    89
dtype: int64
```

In [83]:
```
movies.sample(2)
```

Out[83]:

	title_x	imdb_id	poster_path	wiki_link	title_y	original_title	is_adult	year_of_r
107	Hope Aur Hum	tt8324474	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Hope_Aur_Hum	Hope Aur Hum	Hope Aur Hum	0	
666	Tere Naal Love Ho Gaya	tt2130242	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Tere_Naal_Love_H...	Tere Naal Love Ho Gaya	Tere Naal Love Ho Gaya	0	

In [84]:
```
movies.sort_values('title_x', ascending=False)
```

Out[84]:

	title_x	imdb_id	poster_path	wiki_link	title_y	original_title	is_adult	y
1623	Zubeidaa	tt0255713	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Zubeidaa	Zubeidaa	Zubeidaa	0	
939	Zor Lagaa Ke...Haiya!	tt1479857	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Zor_Lagaa_Ke...H...	Zor Lagaa Ke... Haiya!	Zor Lagaa Ke... Haiya!	0	
756	Zokkomon	tt1605790	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Zokkomon	Zokkomon	Zokkomon	0	
670	Zindagi Tere Naam	tt2164702	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Zindagi_Tere_Naam	Zindagi Tere Naam	Zindagi Tere Naam	0	
778	Zindagi Na Milegi Dobara	tt1562872	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Zindagi_Na_Mileg...	Zindagi Na Milegi Dobara	Zindagi Na Milegi Dobara	0	
...	...	...	...	...	...	...	...	
1039	1971 (2007 film)	tt0983990	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/1971_(2007_film)	1971	1971	0	
723	1920: The Evil Returns	tt2222550	https://upload.wikimedia.org/wikipedia/en/e/e7...	https://en.wikipedia.org/wiki/1920:_The_Evil_R...	1920: Evil Returns	1920: Evil Returns	0	
287	1920: London	tt5638500	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/1920_London	1920 London	1920 London	0	
1021	1920 (film)	tt1301698	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/1920_(film)	1920	1920	0	
1498	16 December (film)	tt0313844	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/16_December_(film)	16-Dec	16-Dec	0	

1629 rows × 19 columns

In [85]:
```
students = pd.DataFrame(
    {
        'name':['nitish','ankit','rupesh',np.nan,'mrityunjay',np.nan,'rishabh',np.nan,'aditya',np.nan],
        'college':['bit','iit','vit',np.nan,np.nan,'vlsi','ssit',np.nan,np.nan,'git'],
        'branch':['eee','it','cse',np.nan,'me','ce','civ','cse','bio',np.nan],
        'cgpa':[6.66,8.25,6.41,np.nan,5.6,9.0,7.4,10,7.4,np.nan],
        'package':[4,5,6,np.nan,6,7,8,9,np.nan,np.nan]

    }
)

students
```

Out[85]:

	name	college	branch	cgpa	package
0	nitish	bit	eee	6.66	4.0
1	ankit	iit	it	8.25	5.0
2	rupesh	vit	cse	6.41	6.0
3	NaN	NaN	NaN	NaN	NaN
4	mrityunjay	NaN	me	5.60	6.0
5	NaN	vlsi	ce	9.00	7.0
6	rishabh	ssit	civ	7.40	8.0
7	NaN	NaN	cse	10.00	9.0
8	aditya	NaN	bio	7.40	NaN
9	NaN	git	NaN	NaN	NaN

In [86]:
```
students.sort_values('name', ascending=False, na_position='first') # inplace=True ( for Permanent Changes)
```

Out[86]:

	name	college	branch	cgpa	package
3	NaN	NaN	NaN	NaN	NaN
5	NaN	vlsi	ce	9.00	7.0
7	NaN	NaN	cse	10.00	9.0
9	NaN	git	NaN	NaN	NaN
2	rupesh	vit	cse	6.41	6.0
6	rishabh	ssit	civ	7.40	8.0
0	nitish	bit	eee	6.66	4.0
4	mrityunjay	NaN	me	5.60	6.0
1	ankit	iit	it	8.25	5.0
8	aditya	NaN	bio	7.40	NaN

In [87]:
```
students
```

Out[87]:

	name	college	branch	cgpa	package
0	nitish	bit	eee	6.66	4.0
1	ankit	iit	it	8.25	5.0
2	rupesh	vit	cse	6.41	6.0
3	NaN	NaN	NaN	NaN	NaN
4	mrityunjay	NaN	me	5.60	6.0
5	NaN	vlsi	ce	9.00	7.0
6	rishabh	ssit	civ	7.40	8.0
7	NaN	NaN	cse	10.00	9.0
8	aditya	NaN	bio	7.40	NaN
9	NaN	git	NaN	NaN	NaN

In [88]:
```
movies.sort_values(['year_of_release','title_x'], ascending=[True,False]).head(2)
```

Out[88]:

	title_x	imdb_id	poster_path	wiki_link	title_y	original_title	is_adult	yea
1623	Zubeidaa	tt0255713	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Zubeidaa	Zubeidaa	Zubeidaa	0	
1625	Yeh Zindagi Ka Safar	tt0298607	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Yeh_Zindagi_Ka_S...	Yeh Zindagi Ka Safar	Yeh Zindagi Ka Safar	0	

rank(Series)

In [89]:
```
batsman=pd.read_csv("batsman_runs_ipl.csv")
```

In [90]:
```
batsman.head(2)
```

Out[90]:

	batter	batsman_run
0	A Ashish Reddy	280
1	A Badoni	161

In [91]:
```
batsman['batsman_run'].rank(ascending=False)
```

Out[91]:
```
0      166.5
1      226.0
2      535.0
3      329.0
4      402.5
       ...
600    594.0
601    343.0
602    547.5
603     27.0
604    256.0
Name: batsman_run, Length: 605, dtype: float64
```

In [92]:
```
batsman['batsman_rank'] = batsman['batsman_run'].rank(ascending=False)
batsman.sort_values('batsman_rank')
```

Out[92]:

	batter	batsman_run	batsman_rank
569	V Kohli	6634	1.0
462	S Dhawan	6244	2.0
130	DA Warner	5883	3.0
430	RG Sharma	5881	4.0
493	SK Raina	5536	5.0
...	...	...	...
512	SS Cottrell	0	594.0
466	S Kaushik	0	594.0
203	IC Pandey	0	594.0
467	S Ladda	0	594.0
468	S Lamichhane	0	594.0

605 rows × 3 columns

sort_index (Series and dataframe)

In [93]:
```
marks = {
    'maths':67,
    'english':57,
    'science':89,
    'hindi':100
}

marks_series = pd.Series(marks)
marks_series
```

Out[93]:
```
maths       67
english     57
science     89
hindi      100
dtype: int64
```

In [94]:
```
marks_series.sort_index()
```

Out[94]:
```
english     57
hindi      100
maths       67
science     89
dtype: int64
```

In [95]:
```
movies.sort_index(ascending=False)
```

Out[95]:

	title_x	imdb_id	poster_path	wiki_link	title_y	original_title	is_adult
1628	Humsafar	tt2403201	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Humsafar	Humsafar	Humsafar	0
1627	Daaka	tt10833860	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Daaka	Daaka	Daaka	0
1626	Sabse Bada Sukh	tt0069204	NaN	https://en.wikipedia.org/wiki/Sabse_Bada_Sukh	Sabse Bada Sukh	Sabse Bada Sukh	0
1625	Yeh Zindagi Ka Safar	tt0298607	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Yeh_Zindagi_Ka_S...	Yeh Zindagi Ka Safar	Yeh Zindagi Ka Safar	0
1624	Tera Mera Saath Rahen	tt0301250	https://upload.wikimedia.org/wikipedia/en/2/2b...	https://en.wikipedia.org/wiki/Tera_Mera_Saath_...	Tera Mera Saath Rahen	Tera Mera Saath Rahen	0
...	...	...	...	...	...	...	...
4	Evening Shadows	tt6028796	NaN	https://en.wikipedia.org/wiki/Evening_Shadows	Evening Shadows	Evening Shadows	0
3	Why Cheat India	tt8108208	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Why_Cheat_India	Why Cheat India	Why Cheat India	0
2	The Accidental Prime Minister (film)	tt6986710	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/The_Accidental_P...	The Accidental Prime Minister	The Accidental Prime Minister	0
1	Battalion 609	tt9472208	NaN	https://en.wikipedia.org/wiki/Battalion_609	Battalion 609	Battalion 609	0
0	Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Uri:_The_Surgica...	Uri: The Surgical Strike	Uri: The Surgical Strike	0

1629 rows × 19 columns

In [96]:
```
# set_index(dataframe) -> inplace
batsman.set_index('batter',inplace=True)
```

In [99]:
```
how to delete columns
```

```
  File "C:\Users\user\AppData\Local\Temp/ipykernel_17496/2400687195.py", line 1
    how to delete columns
        ^
SyntaxError: invalid syntax
```

In [140]:
```
# This code drops two columns from the 'batsman' dataframe: 'level_0' and 'index'.
# The 'inplace=True' parameter ensures that the original dataframe is modified instead of creating a new one.

batsman.drop(['index'], axis=1, inplace=True)
```

In [101]:
```
# reset_index(series + dataframe) -> drop parameter
batsman.reset_index(inplace=True)
```

In [102]:
```
batsman
```

Out[102]:

	batter	batsman_run	batsman_rank
0	A Ashish Reddy	280	166.5
1	A Badoni	161	226.0
2	A Chandila	4	535.0
3	A Chopra	53	329.0
4	A Choudhary	25	402.5
...	...	...	...
600	Yash Dayal	0	594.0
601	Yashpal Singh	47	343.0
602	Younis Khan	3	547.5
603	Yuvraj Singh	2754	27.0
604	Z Khan	117	256.0

605 rows × 3 columns

In [103]:
```
# how to replace existing index without loosing
batsman.reset_index().set_index('batsman_rank')
```

Out[103]:

batsman_rank	index	batter	batsman_run
166.5	0	A Ashish Reddy	280
226.0	1	A Badoni	161
535.0	2	A Chandila	4
329.0	3	A Chopra	53
402.5	4	A Choudhary	25
...	...	...	...
594.0	600	Yash Dayal	0
343.0	601	Yashpal Singh	47
547.5	602	Younis Khan	3
27.0	603	Yuvraj Singh	2754
256.0	604	Z Khan	117

605 rows × 3 columns

In [104]:
```
batsman
```

Out[104]:

	batter	batsman_run	batsman_rank
0	A Ashish Reddy	280	166.5
1	A Badoni	161	226.0
2	A Chandila	4	535.0
3	A Chopra	53	329.0
4	A Choudhary	25	402.5
...	...	...	...
600	Yash Dayal	0	594.0
601	Yashpal Singh	47	343.0
602	Younis Khan	3	547.5
603	Yuvraj Singh	2754	27.0
604	Z Khan	117	256.0

605 rows × 3 columns

In [105]:
```
# series to dataframe using reset_index
marks_series.reset_index()
```

Out[105]:

	index	0
0	maths	67
1	english	57
2	science	89
3	hindi	100

In [106]:
```
type(marks_series.reset_index())
```

Out[106]:
```
pandas.core.frame.DataFrame
```

rename(dataframe) -> index

In [107]:
```
movies.set_index('title_x',inplace=True)
```

In [108]:
```
movies
```

Out[108]:

title_x	imdb_id	poster_path	wiki_link	title_y	original_title	is_adult	year_
Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Uri:_The_Surgica...	Uri: The Surgical Strike	Uri: The Surgical Strike	0	
Battalion 609	tt9472208	NaN	https://en.wikipedia.org/wiki/Battalion_609	Battalion 609	Battalion 609	0	
The Accidental Prime Minister (film)	tt6986710	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/The_Accidental_P...	The Accidental Prime Minister	The Accidental Prime Minister	0	
Why Cheat India	tt8108208	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Why_Cheat_India	Why Cheat India	Why Cheat India	0	
Evening Shadows	tt6028796	NaN	https://en.wikipedia.org/wiki/Evening_Shadows	Evening Shadows	Evening Shadows	0	
...	...	...	...	...	...	...	...
Tera Mera Saath Rahen	tt0301250	https://upload.wikimedia.org/wikipedia/en/2/2b...	https://en.wikipedia.org/wiki/Tera_Mera_Saath_...	Tera Mera Saath Rahen	Tera Mera Saath Rahen	0	
Yeh Zindagi Ka Safar	tt0298607	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Yeh_Zindagi_Ka_S...	Yeh Zindagi Ka Safar	Yeh Zindagi Ka Safar	0	
Sabse Bada Sukh	tt0069204	NaN	https://en.wikipedia.org/wiki/Sabse_Bada_Sukh	Sabse Bada Sukh	Sabse Bada Sukh	0	
Daaka	tt10833860	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Daaka	Daaka	Daaka	0	
Humsafar	tt2403201	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Humsafar	Humsafar	Humsafar	0	

1629 rows × 18 columns

In [109]:
```
#Rename the columns
movies.rename(columns={'imdb_id' : 'imdb', 'poster_path':'link'},inplace=True)
```

In [110]:
```
movies
```

Out[110]:

title_x	imdb	link	wiki_link	title_y	original_title	is_adult	year_
Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Uri:_The_Surgica...	Uri: The Surgical Strike	Uri: The Surgical Strike	0	
Battalion 609	tt9472208	NaN	https://en.wikipedia.org/wiki/Battalion_609	Battalion 609	Battalion 609	0	
The Accidental Prime Minister (film)	tt6986710	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/The_Accidental_P...	The Accidental Prime Minister	The Accidental Prime Minister	0	
Why Cheat India	tt8108208	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Why_Cheat_India	Why Cheat India	Why Cheat India	0	
Evening Shadows	tt6028796	NaN	https://en.wikipedia.org/wiki/Evening_Shadows	Evening Shadows	Evening Shadows	0	
...	...	...	...	...	...	...	...
Tera Mera Saath Rahen	tt0301250	https://upload.wikimedia.org/wikipedia/en/2/2b...	https://en.wikipedia.org/wiki/Tera_Mera_Saath_...	Tera Mera Saath Rahen	Tera Mera Saath Rahen	0	
Yeh Zindagi Ka Safar	tt0298607	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Yeh_Zindagi_Ka_S...	Yeh Zindagi Ka Safar	Yeh Zindagi Ka Safar	0	
Sabse Bada Sukh	tt0069204	NaN	https://en.wikipedia.org/wiki/Sabse_Bada_Sukh	Sabse Bada Sukh	Sabse Bada Sukh	0	
Daaka	tt10833860	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Daaka	Daaka	Daaka	0	
Humsafar	tt2403201	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Humsafar	Humsafar	Humsafar	0	

1629 rows × 18 columns

In [111]:
```
# Rename the index
movies.rename(index={'Uri: The Surgical Strike':'uri','Humsafar':'Hum'})
```

Out[111]:

title_x	imdb	link	wiki_link	title_y	original_title	is_adult	year_
uri	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Uri:_The_Surgica...	Uri: The Surgical Strike	Uri: The Surgical Strike	0	
Battalion 609	tt9472208	NaN	https://en.wikipedia.org/wiki/Battalion_609	Battalion 609	Battalion 609	0	
The Accidental Prime Minister (film)	tt6986710	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/The_Accidental_P...	The Accidental Prime Minister	The Accidental Prime Minister	0	
Why Cheat India	tt8108208	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Why_Cheat_India	Why Cheat India	Why Cheat India	0	
Evening Shadows	tt6028796	NaN	https://en.wikipedia.org/wiki/Evening_Shadows	Evening Shadows	Evening Shadows	0	
...	...	...	...	...	...	...	...
Tera Mera Saath Rahen	tt0301250	https://upload.wikimedia.org/wikipedia/en/2/2b...	https://en.wikipedia.org/wiki/Tera_Mera_Saath_...	Tera Mera Saath Rahen	Tera Mera Saath Rahen	0	
Yeh Zindagi Ka Safar	tt0298607	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Yeh_Zindagi_Ka_S...	Yeh Zindagi Ka Safar	Yeh Zindagi Ka Safar	0	
Sabse Bada Sukh	tt0069204	NaN	https://en.wikipedia.org/wiki/Sabse_Bada_Sukh	Sabse Bada Sukh	Sabse Bada Sukh	0	
Daaka	tt10833860	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Daaka	Daaka	Daaka	0	
Hum	tt2403201	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Humsafar	Humsafar	Humsafar	0	

1629 rows × 18 columns

unique

In [112]:
```
# unique(series)
temp = pd.Series([1,1,2,2,3,3,4,4,5,5,np.nan,np.nan])
print(temp)
temp.unique()
```

```
0     1.0
1     1.0
2     2.0
3     2.0
4     3.0
5     3.0
6     4.0
7     4.0
8     5.0
9     5.0
10    NaN
11    NaN
dtype: float64
```

Out[112]:
```
array([ 1.,  2.,  3.,  4.,  5., nan])
```

In [113]:
```
ipl['Season'].unique()
```

Out[113]:
```
['2022', '2021', '2020/21', '2019', '2018', ..., '2012', '2011', '2009/10', '2009', '2007/08']
Length: 15
Categories (15, object): ['2007/08', '2009', '2009/10', '2011', ..., '2019', '2020/21', '2021', '2022']
```

- nunique : returns the number of unique elements in a pandas Series or DataFrame. It doesn't count the missing values (NaNs) by default. If you want to count the missing values, you can set the argument "dropna" to False.
- unique: returns the unique elements in a pandas Series or DataFrame. It counts the missing values (NaNs) by default. If you don't want to count the missing values, you can use the "dropna" argument and set it to True.

In [114]:
```
len(ipl['Season'].unique())
```

Out[114]:
```
15
```

In [115]:
```
# nunique(series + dataframe) -> does not count nan -> dropna parameter
ipl['Season'].nunique()
```

Out[115]:
```
15
```

isnull(series + dataframe)

In [116]:
```
students
```

Out[116]:

	name	college	branch	cgpa	package
0	nitish	bit	eee	6.66	4.0
1	ankit	iit	it	8.25	5.0
2	rupesh	vit	cse	6.41	6.0
3	NaN	NaN	NaN	NaN	NaN
4	mrityunjay	NaN	me	5.60	6.0
5	NaN	vlsi	ce	9.00	7.0
6	rishabh	ssit	civ	7.40	8.0
7	NaN	NaN	cse	10.00	9.0
8	aditya	NaN	bio	7.40	NaN
9	NaN	git	NaN	NaN	NaN

In [117]:
```
students['name'].isnull()
```

Out[117]:
```
0    False
1    False
2    False
3     True
4    False
5     True
6    False
7     True
8    False
9     True
Name: name, dtype: bool
```

In [118]:
```
# notnull(series + dataframe)
students['name'].notnull()
```

Out[118]:
```
0     True
1     True
2     True
3    False
4     True
5    False
6     True
7    False
8     True
9    False
Name: name, dtype: bool
```

In [119]:
```
students['name'][students['name'].notnull()]
```

Out[119]:
```
0        nitish
1         ankit
2        rupesh
4    mrityunjay
6       rishabh
8        aditya
Name: name, dtype: object
```

In [120]:
```
# hasnans(series)
students['college'].hasnans
```

Out[120]:
```
True
```

In [121]:
```
students.isnull()
```

Out[121]:

	name	college	branch	cgpa	package
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	True	True	True	True	True
4	False	True	False	False	False
5	True	False	False	False	False
6	False	False	False	False	False
7	True	True	False	False	False
8	False	True	False	False	True
9	True	False	True	True	True

In [122]:
```
students.notnull()
```

Out[122]:

	name	college	branch	cgpa	package
0	True	True	True	True	True
1	True	True	True	True	True
2	True	True	True	True	True
3	False	False	False	False	False
4	True	False	True	True	True
5	False	True	True	True	True
6	True	True	True	True	True
7	False	False	True	True	True
8	True	False	True	True	False
9	False	True	False	False	False

dropna -> for Missing Values

In [123]:
```
# dropna(series + dataframe) -> how parameter -> works like or
students['name'].dropna()
```

Out[123]:
```
0        nitish
1         ankit
2        rupesh
4    mrityunjay
6       rishabh
8        aditya
Name: name, dtype: object
```

how : {'any', 'all'}, default 'any'

```
    Determine if row or column is removed from DataFrame, when we have
    at least one NA or all NA.


    * 'any' : If any NA values are present, drop that row or column.
    * 'all' : If all values are NA, drop that row or column.
```

In [124]:
```
students.dropna(how='any')
```

Out[124]:

	name	college	branch	cgpa	package
0	nitish	bit	eee	6.66	4.0
1	ankit	iit	it	8.25	5.0
2	rupesh	vit	cse	6.41	6.0
6	rishabh	ssit	civ	7.40	8.0

In [125]:
```
students.dropna(how='all')
```

Out[125]:

	name	college	branch	cgpa	package
0	nitish	bit	eee	6.66	4.0
1	ankit	iit	it	8.25	5.0
2	rupesh	vit	cse	6.41	6.0
4	mrityunjay	NaN	me	5.60	6.0
5	NaN	vlsi	ce	9.00	7.0
6	rishabh	ssit	civ	7.40	8.0
7	NaN	NaN	cse	10.00	9.0
8	aditya	NaN	bio	7.40	NaN
9	NaN	git	NaN	NaN	NaN

subset : array-like, optional

```
    Labels along other axis to consider, e.g. if you are dropping rows
    these would be a list of columns to include.
```

In [126]:
```
students.dropna(subset=['name'])
```

Out[126]:

	name	college	branch	cgpa	package
0	nitish	bit	eee	6.66	4.0
1	ankit	iit	it	8.25	5.0
2	rupesh	vit	cse	6.41	6.0
4	mrityunjay	NaN	me	5.60	6.0
6	rishabh	ssit	civ	7.40	8.0
8	aditya	NaN	bio	7.40	NaN

In [127]:
```
students.dropna(subset=['name','college'])
```

Out[127]:

	name	college	branch	cgpa	package
0	nitish	bit	eee	6.66	4.0
1	ankit	iit	it	8.25	5.0
2	rupesh	vit	cse	6.41	6.0
6	rishabh	ssit	civ	7.40	8.0

fill na (series + dataframe)

In [128]:
```
students
```

Out[128]:

	name	college	branch	cgpa	package
0	nitish	bit	eee	6.66	4.0
1	ankit	iit	it	8.25	5.0
2	rupesh	vit	cse	6.41	6.0
3	NaN	NaN	NaN	NaN	NaN
4	mrityunjay	NaN	me	5.60	6.0
5	NaN	vlsi	ce	9.00	7.0
6	rishabh	ssit	civ	7.40	8.0
7	NaN	NaN	cse	10.00	9.0
8	aditya	NaN	bio	7.40	NaN
9	NaN	git	NaN	NaN	NaN

In [129]:
```
students['name'].fillna('unknown')
```

Out[129]:
```
0        nitish
1         ankit
2        rupesh
3       unknown
4    mrityunjay
5       unknown
6       rishabh
7       unknown
8        aditya
9       unknown
Name: name, dtype: object
```

In [130]:
```
students.fillna('0')
```

Out[130]:

	name	college	branch	cgpa	package
0	nitish	bit	eee	6.66	4.0
1	ankit	iit	it	8.25	5.0
2	rupesh	vit	cse	6.41	6.0
3	0	0	0	0	0
4	mrityunjay	0	me	5.6	6.0
5	0	vlsi	ce	9.0	7.0
6	rishabh	ssit	civ	7.4	8.0
7	0	0	cse	10.0	9.0
8	aditya	0	bio	7.4	0
9	0	git	0	0	0

In [131]:
```
students['package'].fillna(students['package'].mean())
```

Out[131]:
```
0    4.000000
1    5.000000
2    6.000000
3    6.428571
4    6.000000
5    7.000000
6    8.000000
7    9.000000
8    6.428571
9    6.428571
Name: package, dtype: float64
```

In [132]:
```
students['name'].fillna(method='ffill') #forward fill method
```

Out[132]:
```
0        nitish
1         ankit
2        rupesh
3        rupesh
4    mrityunjay
5    mrityunjay
6       rishabh
7       rishabh
8        aditya
9        aditya
Name: name, dtype: object
```

In [133]:
```
students['name'].fillna(method='bfill') # backward fill method
```

Out[133]:
```
0        nitish
1         ankit
2        rupesh
3    mrityunjay
4    mrityunjay
5       rishabh
6       rishabh
7        aditya
8        aditya
9           NaN
Name: name, dtype: object
```

drop_duplicates

In [134]:
```
# drop_duplicates(series + dataframe) -> works like and -> duplicated()
```

In [135]:
```
marks
```

Out[135]:
```
{'maths': 67, 'english': 57, 'science': 89, 'hindi': 100}
```

In [144]:
```
marks
```

Out[144]:

	iq	marks	package
0	100	80	10
1	90	70	7
2	120	100	14
3	80	70	14
4	80	70	14

In [145]:
```
marks.drop_duplicates()
```

Out[145]:

	iq	marks	package
0	100	80	10
1	90	70	7
2	120	100	14
3	80	70	14

In [146]:
```
marks.drop_duplicates(keep='last')
```

Out[146]:

	iq	marks	package
0	100	80	10
1	90	70	7
2	120	100	14
4	80	70	14

In [148]:
```
# find the last match played by virat kohli in Delhi
ipl.sample(2)
```

Out[148]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecision	SuperOver	WinningTeam	WonBy	M
6	1304114	Mumbai	2022-05-20	2022	68	Chennai Super Kings	Rajasthan Royals	Brabourne Stadium, Mumbai	Chennai Super Kings	bat	N	Rajasthan Royals	Wickets	
276	1136598	Indore	2018-05-06	2018	38	Rajasthan Royals	Kings XI Punjab	Holkar Cricket Stadium	Kings XI Punjab	field	N	Kings XI Punjab	Wickets	

In [149]:
```
ipl['all Players'] = ipl['Team1Players'] + ipl['Team2Players']
ipl.sample(2)
```

Out[149]:

nue	TossWinner	TossDecision	...	WinningTeam	WonBy	Margin	method	Player_of_Match	Team1Players	Team2Players	Umpire1	Umpire2
MA ıram ium, ɔauk	Rajasthan Royals	bat	...	Chennai Super Kings	Wickets	8.0	NaN	MEK Hussey	['MEK Hussey', 'M Vijay', 'SK Raina', 'JA Mork...	['SR Watson', 'R Dravid', 'AL Menaria', 'J Bot...	SS Hazare	RB Tiffin
ıede ium, nbai	Delhi Capitals	field	...	Royal Challengers Bangalore	Runs	16.0	NaN	KD Karthik	['F du Plessis', 'Anuj Rawat', 'V Kohli', 'GJ ...	['PP Shaw', 'DA Warner', 'MR Marsh', 'RR Pant'...	Chirra Ravikanthreddy	J Madanagopal

In [154]:
```
def did_kohli_play(players_list):
    return 'V Kohli' in players_list
```

In [160]:
```
ipl['did_kohli_play'] = ipl['all Players'].apply(did_kohli_play)
```

In [165]:
```
ipl.sample(2)
```

Out[165]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecision	...	Margin	method	Player_of_Matc
940	335991	Chandigarh	2008-04-25	2007/08	10	Kings XI Punjab	Mumbai Indians	Punjab Cricket Association Stadium, Mohali	Mumbai Indians	field	...	66.0	NaN	KC Sangakka
826	419114	Delhi	2010-03-17	2009/10	9	Delhi Daredevils	Mumbai Indians	Feroz Shah Kotla	Delhi Daredevils	field	...	98.0	NaN	SR Tendulk

2 rows × 23 columns

In [166]:
```
ipl[( ipl['City'] == 'Delhi')  & (ipl['did_kohli_play'] == True)]
```

Out[166]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecision	...	Margin	method	Player_of_Match
208	1178421	Delhi	2019-04-28	2019	46	Delhi Capitals	Royal Challengers Bangalore	Arun Jaitley Stadium	Delhi Capitals	bat	...	16.0	NaN	S Dhawan
269	1136605	Delhi	2018-05-12	2018	45	Delhi Daredevils	Royal Challengers Bangalore	Arun Jaitley Stadium	Royal Challengers Bangalore	field	...	5.0	NaN	AB de Villiers
318	1082646	Delhi	2017-05-14	2017	56	Delhi Daredevils	Royal Challengers Bangalore	Feroz Shah Kotla	Royal Challengers Bangalore	bat	...	10.0	NaN	HV Patel
467	829757	Delhi	2015-04-26	2015	26	Delhi Daredevils	Royal Challengers Bangalore	Feroz Shah Kotla	Royal Challengers Bangalore	field	...	10.0	NaN	VR Aaron
571	598054	Delhi	2013-05-10	2013	57	Delhi Daredevils	Royal Challengers Bangalore	Feroz Shah Kotla	Delhi Daredevils	field	...	4.0	NaN	JD Unadkat
638	548372	Delhi	2012-05-17	2012	67	Delhi Daredevils	Royal Challengers Bangalore	Feroz Shah Kotla	Delhi Daredevils	field	...	21.0	NaN	CH Gayle
746	501227	Delhi	2011-04-26	2011	30	Delhi Daredevils	Royal Challengers Bangalore	Feroz Shah Kotla	Royal Challengers Bangalore	field	...	3.0	NaN	V Kohli
801	419140	Delhi	2010-04-04	2009/10	35	Delhi Daredevils	Royal Challengers Bangalore	Feroz Shah Kotla	Delhi Daredevils	bat	...	37.0	NaN	PD Collingwood
933	335998	Delhi	2008-04-30	2007/08	17	Delhi Daredevils	Royal Challengers Bangalore	Feroz Shah Kotla	Royal Challengers Bangalore	field	...	10.0	NaN	GD McGrath

9 rows × 23 columns

In [168]:
```
ipl[( ipl['City'] == 'Delhi')  & (ipl['did_kohli_play'] == True)].drop_duplicates(subset=['City','did_kohli_play'],keep='
```

Out[168]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossWinner	TossDecision	...	Margin	method	Player_of_Match	Tea
208	1178421	Delhi	2019-04-28	2019	46	Delhi Capitals	Royal Challengers Bangalore	Arun Jaitley Stadium	Delhi Capitals	bat	...	16.0	NaN	S Dhawan	['F D

1 rows × 23 columns

drop(series + dataframe)

In [169]:
```
# Series
temp = pd.Series([10,2,3,16,45,78,10])
temp
```

Out[169]:
```
0    10
1     2
2     3
3    16
4    45
5    78
6    10
dtype: int64
```

In [170]:
```
temp.drop(index=[0,6])
```

Out[170]:
```
1     2
2     3
3    16
4    45
5    78
dtype: int64
```

In [171]:
```
students
```

Out[171]:

	name	college	branch	cgpa	package
0	nitish	bit	eee	6.66	4.0
1	ankit	iit	it	8.25	5.0
2	rupesh	vit	cse	6.41	6.0
3	NaN	NaN	NaN	NaN	NaN
4	mrityunjay	NaN	me	5.60	6.0
5	NaN	vlsi	ce	9.00	7.0
6	rishabh	ssit	civ	7.40	8.0
7	NaN	NaN	cse	10.00	9.0
8	aditya	NaN	bio	7.40	NaN
9	NaN	git	NaN	NaN	NaN

In [172]:
```
students.drop(columns=['branch','cgpa']) # To delete Columns
```

Out[172]:

	name	college	package
0	nitish	bit	4.0
1	ankit	iit	5.0
2	rupesh	vit	6.0
3	NaN	NaN	NaN
4	mrityunjay	NaN	6.0
5	NaN	vlsi	7.0
6	rishabh	ssit	8.0
7	NaN	NaN	9.0
8	aditya	NaN	NaN
9	NaN	git	NaN

In [173]:
```
students.drop(index=[0,8]) # to delete rows
```

Out[173]:

	name	college	branch	cgpa	package
1	ankit	iit	it	8.25	5.0
2	rupesh	vit	cse	6.41	6.0
3	NaN	NaN	NaN	NaN	NaN
4	mrityunjay	NaN	me	5.60	6.0
5	NaN	vlsi	ce	9.00	7.0
6	rishabh	ssit	civ	7.40	8.0
7	NaN	NaN	cse	10.00	9.0
9	NaN	git	NaN	NaN	NaN

In [174]:
```
students.set_index('name')
```

Out[174]:

	college	branch	cgpa	package
name				
nitish	bit	eee	6.66	4.0
ankit	iit	it	8.25	5.0
rupesh	vit	cse	6.41	6.0
NaN	NaN	NaN	NaN	NaN
mrityunjay	NaN	me	5.60	6.0
NaN	vlsi	ce	9.00	7.0
rishabh	ssit	civ	7.40	8.0
NaN	NaN	cse	10.00	9.0
aditya	NaN	bio	7.40	NaN
NaN	git	NaN	NaN	NaN

In [175]:
```
students.set_index('name').drop(index=['nitish','aditya']) # delete by names
```

Out[175]:

	college	branch	cgpa	package
name				
ankit	iit	it	8.25	5.0
rupesh	vit	cse	6.41	6.0
NaN	NaN	NaN	NaN	NaN
mrityunjay	NaN	me	5.60	6.0
NaN	vlsi	ce	9.00	7.0
rishabh	ssit	civ	7.40	8.0
NaN	NaN	cse	10.00	9.0
NaN	git	NaN	NaN	NaN

apply(series + dataframe)

In [176]:
```
# series
temp = pd.Series([10,20,30,40,50])

temp
```

Out[176]:
```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

In [177]:
```
def sigmoid(value):
    return 1/1+np.exp(-value)
```

In [178]:
```
temp.apply(sigmoid)
```

Out[178]:
```
0    1.000045
1    1.000000
2    1.000000
3    1.000000
4    1.000000
dtype: float64
```

In [179]:
```
# On data Frame
points = pd.DataFrame(
    {
        '1st point':[(3,4),(-6,5),(0,0),(-10,1),(4,5)],
        '2nd point':[(-3,4),(0,0),(2,2),(10,10),(1,1)]
    }
)

points
```

Out[179]:

	1st point	2nd point
0	(3, 4)	(-3, 4)
1	(-6, 5)	(0, 0)
2	(0, 0)	(2, 2)
3	(-10, 1)	(10, 10)
4	(4, 5)	(1, 1)

In [180]:
```
def euclidean(row):
    point_A = row['1st point']
    point_B = row['2nd point']
    return ((point_A[0] - point_B[0])** 2 + (point_A[1]- point_B[1])**2 )** 0.5
```

In [182]:
```
points.apply(euclidean,axis=1) # mention axis=1 because its row wise
```

Out[182]:
```
0     6.000000
1     7.810250
2     2.828427
3    21.931712
4     5.000000
dtype: float64
```

In [184]:
```
points['distance'] = points.apply(euclidean,axis=1)
points
```

Out[184]:

	1st point	2nd point	distance
0	(3, 4)	(-3, 4)	6.000000
1	(-6, 5)	(0, 0)	7.810250
2	(0, 0)	(2, 2)	2.828427
3	(-10, 1)	(10, 10)	21.931712
4	(4, 5)	(1, 1)	5.000000

Groupby (Applies on Categorical data)

In [187]:
```
movies = pd.read_csv("imdb-top-1000.csv")
```

In [189]:
```
movies.head(1)
```

Out[189]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Votes	Gross	Metascore
0	The Shawshank Redemption	1994	142	Drama	9.3	Frank Darabont	Tim Robbins	2343110	28341469.0	80.0

In [190]:
```
movies.groupby('Genre')
```

Out[190]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001EFE1E1F670>
```

In [195]:
```
generes =movies.groupby('Genre')
```

In [196]:
```
# Applying builtin aggregation fuctions on groupby objects
generes.sum()
```

Out[196]:

Genre	Runtime	IMDB_Rating	No_of_Votes	Gross	Metascore
Action	22196	1367.3	72282412	3.263226e+10	10499.0
Adventure	9656	571.5	22576163	9.496922e+09	5020.0
Animation	8166	650.3	21978630	1.463147e+10	6082.0
Biography	11970	698.6	24006844	8.276358e+09	6023.0
Comedy	17380	1224.7	27620327	1.566387e+10	9840.0
Crime	13524	857.8	33533615	8.452632e+09	6706.0
Drama	36049	2299.7	61367304	3.540997e+10	19208.0
Family	215	15.6	551221	4.391106e+08	158.0
Fantasy	170	16.0	146222	7.827267e+08	0.0
Film-Noir	312	23.9	367215	1.259105e+08	287.0
Horror	1123	87.0	3742556	1.034649e+09	880.0
Mystery	1429	95.7	4203004	1.256417e+09	633.0
Thriller	108	7.8	27733	1.755074e+07	81.0
Western	593	33.4	1289665	5.822151e+07	313.0

In [197]:
```
generes.mean()
```

Out[197]:

Genre	Runtime	IMDB_Rating	No_of_Votes	Gross	Metascore
Action	129.046512	7.949419	420246.581395	1.897224e+08	73.419580
Adventure	134.111111	7.937500	313557.819444	1.319017e+08	78.437500
Animation	99.585366	7.930488	268032.073171	1.784326e+08	81.093333
Biography	136.022727	7.938636	272805.045455	9.404952e+07	76.240506
Comedy	112.129032	7.901290	178195.658065	1.010572e+08	78.720000
Crime	126.392523	8.016822	313398.271028	7.899656e+07	77.080460
Drama	124.737024	7.957439	212343.612457	1.225259e+08	79.701245
Family	107.500000	7.800000	275610.500000	2.195553e+08	79.000000
Fantasy	85.000000	8.000000	73111.000000	3.913633e+08	NaN
Film-Noir	104.000000	7.966667	122405.000000	4.197018e+07	95.666667
Horror	102.090909	7.909091	340232.363636	9.405902e+07	80.000000
Mystery	119.083333	7.975000	350250.333333	1.047014e+08	79.125000
Thriller	108.000000	7.800000	27733.000000	1.755074e+07	81.000000
Western	148.250000	8.350000	322416.250000	1.455538e+07	78.250000

In [198]:
```
generes.min()
```

Out[198]:

Genre	Series_Title	Released_Year	Runtime	IMDB_Rating	Director	Star1	No_of_Votes	Gross	Metascore
Action	300	1924	45	7.6	Abhishek Chaubey	Aamir Khan	25312	3296.0	33.0
Adventure	2001: A Space Odyssey	1925	88	7.6	Akira Kurosawa	Aamir Khan	29999	61001.0	41.0
Animation	Akira	1940	71	7.6	Adam Elliot	Adrian Molina	25229	128985.0	61.0
Biography	12 Years a Slave	1928	93	7.6	Adam McKay	Adrien Brody	27254	21877.0	48.0
Comedy	(500) Days of Summer	1921	68	7.6	Alejandro G. Iñárritu	Aamir Khan	26337	1305.0	45.0
Crime	12 Angry Men	1931	80	7.6	Akira Kurosawa	Ajay Devgn	27712	6013.0	47.0
Drama	1917	1925	64	7.6	Aamir Khan	Abhay Deol	25088	3600.0	28.0
Family	E.T. the Extra-Terrestrial	1971	100	7.8	Mel Stuart	Gene Wilder	178731	4000000.0	67.0
Fantasy	Das Cabinet des Dr. Caligari	1920	76	7.9	F.W. Murnau	Max Schreck	57428	337574718.0	NaN
Film-Noir	Shadow of a Doubt	1941	100	7.8	Alfred Hitchcock	Humphrey Bogart	59556	449191.0	94.0
Horror	Alien	1933	71	7.6	Alejandro Amenábar	Anthony Perkins	27007	89029.0	46.0
Mystery	Dark City	1938	96	7.6	Alex Proyas	Bernard-Pierre Donnadieu	33982	1035953.0	52.0
Thriller	Wait Until Dark	1967	108	7.8	Terence Young	Audrey Hepburn	27733	17550741.0	81.0
Western	Il buono, il brutto, il cattivo	1965	132	7.8	Clint Eastwood	Clint Eastwood	65659	5321508.0	69.0

In [206]:
```
# find the top 3 genres by total earning
movies.groupby('Genre').sum()['Gross'].sort_values(ascending=False).head(3)
```

Out[206]:
```
Genre
Drama     3.540997e+10
Action    3.263226e+10
Comedy    1.566387e+10
Name: Gross, dtype: float64
```

In [212]:
```
#Second Approach
movies.groupby('Genre')['Gross'].sum().sort_values()
```

Out[212]:
```
Genre
Thriller     1.755074e+07
Western      5.822151e+07
Film-Noir    1.259105e+08
Family       4.391106e+08
Fantasy      7.827267e+08
Horror       1.034649e+09
Mystery      1.256417e+09
Biography    8.276358e+09
Crime        8.452632e+09
Adventure    9.496922e+09
Animation    1.463147e+10
Comedy       1.566387e+10
Action       3.263226e+10
Drama        3.540997e+10
Name: Gross, dtype: float64
```

In [214]:
```
movies.groupby('Genre')['Gross'].sum().sort_values(ascending= False).head(3)
```

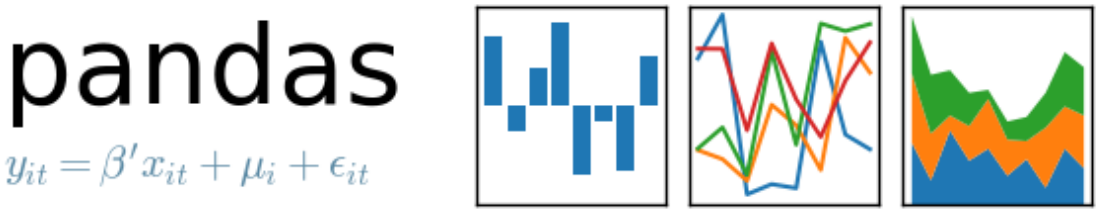
Out[214]:
```
Genre
Drama     3.540997e+10
Action    3.263226e+10
Comedy    1.566387e+10
Name: Gross, dtype: float64
```

In [217]:
```
# find the genre with highest avg IMDB rating
movies.groupby('Genre')['IMDB_Rating'].mean().sort_values(ascending=False).head(1)
```

Out[217]:
```
Genre
Western    8.35
Name: IMDB_Rating, dtype: float64
```

Pandas groupby

groupby in pandas is a function that lets you group data in a DataFrame based on specific criteria, and then apply aggregate functions to each group. It's a powerful tool for data analysis that allows you to quickly and easily calculate summary statistics for your data.



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

Groupby (Applies on Categorical data)

In [2]:
```
movies = pd.read_csv(" imdb-top-1000.csv")
movies.head(3)
```

Out[2]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Votes
0	The Shawshank Redemption	1994	142	Drama	9.3	Frank Darabont	Tim Robbins	234311(
1	The Godfather	1972	175	Crime	9.2	Francis Ford Coppola	Marlon Brando	1620367
2	The Dark Knight	2008	152	Action	9.0	Christopher Nolan	Christian Bale	2303232

In [3]:
```
movies.groupby('Genre')
```

Out[3]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000024A502D5550>
```

In [4]:
```
generes =movies.groupby('Genre')
```

In [5]:
```
# Applying builtin aggregation fuctions on groupby objects
generes.sum().head(2)
```

Out[5]:

Genre	Runtime	IMDB_Rating	No_of_Votes	Gross	Metascore
Action	22196	1367.3	72282412	3.263226e+10	10499.0
Adventure	9656	571.5	22576163	9.496922e+09	5020.0

In [6]:
```
#Second Approach
movies.groupby('Genre')['Gross'].sum().sort_values()
```

Out[6]:
```
Genre
Thriller     1.755074e+07
Western      5.822151e+07
Film-Noir    1.259105e+08
Family       4.391106e+08
Fantasy      7.827267e+08
Horror       1.034649e+09
Mystery      1.256417e+09
Biography    8.276358e+09
Crime        8.452632e+09
Adventure    9.496922e+09
Animation    1.463147e+10
Comedy       1.566387e+10
Action       3.263226e+10
Drama        3.540997e+10
Name: Gross, dtype: float64
```

In [7]:
```
# find the genre with highest avg IMDB rating
movies.groupby('Genre')['IMDB_Rating'].mean().sort_values(ascending=False).hea
```

Out[7]:
```
Genre
Western    8.35
Name: IMDB_Rating, dtype: float64
```

In [8]:
```
# find director with most popularity
movies.groupby('Director')['No_of_Votes'].sum().sort_values(ascending=False).h
```

Out[8]:
```
Director
Christopher Nolan    11578345
Name: No_of_Votes, dtype: int64
```

In [9]:
```
# find the highest rated movie of each genre
movies.head(1)
```

Out[9]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Votes
0	The Shawshank Redemption	1994	142	Drama	9.3	Frank Darabont	Tim Robbins	2343110

In [10]:
```
# find number of movies done by each actor
#movies['Star1'].value_counts() # first method
movies.groupby('Star1')['Series_Title'].count().sort_values(ascending=False)
```

Out[10]:
```
Star1
Tom Hanks             12
Robert De Niro        11
Clint Eastwood        10
Al Pacino             10
Leonardo DiCaprio      9
                      ..
Glen Hansard           1
Giuseppe Battiston     1
Giulietta Masina       1
Gerardo Taracena       1
Ömer Faruk Sorak       1
Name: Series_Title, Length: 660, dtype: int64
```

GroupBy Attributes and Methods

In [12]:
```
# find total number of groups -> len
len(movies.groupby('Genre')) # 14 different groups
```

Out[12]: 14

In [13]:
```
movies['Genre'].nunique() # second method
```

Out[13]: 14

In [14]:
```
# find items in each group -> size
movies.groupby('Genre').size() # index based
```

Out[14]:
```
Genre
Action       172
Adventure     72
Animation     82
Biography     88
Comedy       155
Crime        107
Drama        289
Family         2
Fantasy        2
Film-Noir      3
Horror        11
Mystery       12
Thriller       1
Western        4
dtype: int64
```

In [15]:
```
movies['Genre'].value_counts() # second method
```

Out[15]:
```
Drama        289
Action       172
Comedy       155
Crime        107
Biography     88
Animation     82
Adventure     72
Mystery       12
Horror        11
Western        4
Film-Noir      3
Fantasy        2
Family         2
Thriller       1
Name: Genre, dtype: int64
```

In [18]:
```
# first()/last() -> nth item
#movies.groupby('Genre').first()
#movies.groupby('Genre').last()
movies.groupby('Genre').nth(6) ---> #gives 7th movie
```

Out[18]:

Genre	Series_Title	Released_Year	Runtime	IMDB_Rating	Director	Star1	No_of_Vote
Action	Star Wars: Episode V - The Empire Strikes Back	1980	124	8.7	Irvin Kershner	Mark Hamill	115931
Adventure	North by Northwest	1959	136	8.3	Alfred Hitchcock	Cary Grant	29919
Animation	WALL·E	2008	98	8.4	Andrew Stanton	Ben Burtt	99979
Biography	Braveheart	1995	178	8.3	Mel Gibson	Mel Gibson	95918
Comedy	The Great Dictator	1940	125	8.4	Charles Chaplin	Charles Chaplin	20315
Crime	Se7en	1995	127	8.6	David Fincher	Morgan Freeman	144509
Drama	It's a Wonderful Life	1946	130	8.6	Frank Capra	James Stewart	40580
Horror	Get Out	2017	104	7.7	Jordan Peele	Daniel Kaluuya	49285
Mystery	Sleuth	1972	138	8.0	Joseph L. Mankiewicz	Laurence Olivier	4474

In [19]:
```
# get_group -> vs filtering
movies.groupby('Genre').get_group('Horror')
```

Out[19]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Vo
49	Psycho	1960	109	Horror	8.5	Alfred Hitchcock	Anthony Perkins	6042
75	Alien	1979	117	Horror	8.4	Ridley Scott	Sigourney Weaver	7878
271	The Thing	1982	109	Horror	8.1	John Carpenter	Kurt Russell	3712
419	The Exorcist	1973	122	Horror	8.0	William Friedkin	Ellen Burstyn	3623
544	Night of the Living Dead	1968	96	Horror	7.9	George A. Romero	Duane Jones	1165
707	The Innocents	1961	100	Horror	7.8	Jack Clayton	Deborah Kerr	270
724	Get Out	2017	104	Horror	7.7	Jordan Peele	Daniel Kaluuya	4928
844	Halloween	1978	91	Horror	7.7	John Carpenter	Donald Pleasence	2331
876	The Invisible Man	1933	71	Horror	7.7	James Whale	Claude Rains	306
932	Saw	2004	103	Horror	7.6	James Wan	Cary Elwes	3790
948	The Others	2001	101	Horror	7.6	Alejandro Amenábar	Nicole Kidman	3376

In [20]:
```
movies.groupby('Genre').get_group('Fantasy')
```

Out[20]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Votes
321	Das Cabinet des Dr. Caligari	1920	76	Fantasy	8.1	Robert Wiene	Werner Krauss	57428
568	Nosferatu	1922	94	Fantasy	7.9	F.W. Murnau	Max Schreck	88794

In [21]:
```
movies[movies['Genre']=='Fantasy'] #Second Method
```

Out[21]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Votes
321	Das Cabinet des Dr. Caligari	1920	76	Fantasy	8.1	Robert Wiene	Werner Krauss	57428
568	Nosferatu	1922	94	Fantasy	7.9	F.W. Murnau	Max Schreck	88794

In [24]:
```
# groups
movies.groupby('Genre').groups ---> #Dictionary= gives index postion
```

Out[24]:
```
{'Action': [2, 5, 8, 10, 13, 14, 16, 29, 30, 31, 39, 42, 44, 55, 57, 59, 60,
63, 68, 72, 106, 109, 129, 130, 134, 140, 142, 144, 152, 155, 160, 161, 166,
168, 171, 172, 177, 181, 194, 201, 202, 216, 217, 223, 224, 236, 241, 262, 27
5, 294, 308, 320, 325, 326, 331, 337, 339, 340, 343, 345, 348, 351, 353, 356,
357, 362, 368, 369, 375, 376, 390, 410, 431, 436, 473, 477, 479, 482, 488, 49
3, 496, 502, 507, 511, 532, 535, 540, 543, 564, 569, 570, 573, 577, 582, 583,
602, 605, 608, 615, 623, ...], 'Adventure': [21, 47, 93, 110, 114, 116, 118,
137, 178, 179, 191, 193, 209, 226, 231, 247, 267, 273, 281, 300, 301, 304, 30
6, 323, 329, 361, 366, 377, 402, 406, 415, 426, 458, 470, 497, 498, 506, 513,
514, 537, 549, 552, 553, 566, 576, 604, 609, 618, 638, 647, 675, 681, 686, 69
2, 711, 713, 739, 755, 781, 797, 798, 851, 873, 884, 912, 919, 947, 957, 964,
966, 984, 991], 'Animation': [23, 43, 46, 56, 58, 61, 66, 70, 101, 135, 146,
151, 158, 170, 197, 205, 211, 213, 219, 229, 230, 242, 245, 246, 270, 330, 33
2, 358, 367, 378, 386, 389, 394, 395, 399, 401, 405, 409, 469, 499, 510, 516,
518, 522, 578, 586, 592, 595, 596, 599, 633, 640, 643, 651, 665, 672, 694, 72
8, 740, 741, 744, 756, 758, 761, 771, 783, 796, 799, 822, 828, 843, 875, 891,
892, 902, 906, 920, 956, 971, 976, 986, 992], 'Biography': [7, 15, 18, 35, 3
8, 54, 102, 107, 131, 139, 147, 157, 159, 173, 176, 212, 215, 218, 228, 235,
243, 263, 276, 282, 290, 298, 317, 328, 338, 342, 346, 359, 360, 365, 372, 37
3, 385, 411, 416, 418, 424, 429, 484, 525, 536, 542, 545, 575, 579, 587, 600,
606, 614, 622, 632, 635, 644, 649, 650, 657, 671, 673, 684, 729, 748, 753, 75
7, 759, 766, 770, 779, 809, 810, 815, 820, 831, 849, 858, 877, 882, 897, 910,
915, 923, 940, 949, 952, 987], 'Comedy': [19, 26, 51, 52, 64, 78, 83, 95, 96,
112, 117, 120, 127, 128, 132, 153, 169, 183, 192, 204, 207, 208, 214, 221, 23
3, 238, 240, 250, 251, 252, 256, 261, 266, 277, 284, 311, 313, 316, 318, 322,
327, 374, 379, 381, 392, 396, 403, 413, 414, 417, 427, 435, 445, 446, 449, 45
5, 459, 460, 463, 464, 466, 471, 472, 475, 481, 490, 494, 500, 503, 509, 526,
528, 530, 531, 533, 538, 539, 541, 547, 557, 558, 562, 563, 565, 574, 591, 59
3, 594, 598, 613, 626, 630, 660, 662, 667, 679, 680, 683, 687, 701, ...], 'Cr
ime': [1, 3, 4, 6, 22, 25, 27, 28, 33, 37, 41, 71, 77, 79, 86, 87, 103, 108,
111, 113, 123, 125, 133, 136, 162, 163, 164, 165, 180, 186, 187, 189, 198, 22
2, 232, 239, 255, 257, 287, 288, 299, 305, 335, 363, 364, 380, 384, 397, 437,
438, 441, 442, 444, 450, 451, 465, 474, 480, 485, 487, 505, 512, 519, 520, 52
3, 527, 546, 556, 560, 584, 597, 603, 607, 611, 621, 639, 653, 664, 669, 676,
695, 708, 723, 762, 763, 767, 775, 791, 795, 802, 811, 823, 827, 833, 885, 89
5, 921, 922, 926, 938, ...], 'Drama': [0, 9, 11, 17, 20, 24, 32, 34, 36, 40,
45, 50, 53, 62, 65, 67, 73, 74, 76, 80, 82, 84, 85, 88, 89, 90, 91, 92, 94, 9
7, 98, 99, 100, 104, 105, 121, 122, 124, 126, 138, 141, 143, 148, 149, 150, 1
54, 156, 167, 174, 175, 182, 184, 185, 188, 190, 195, 196, 199, 200, 203, 20
6, 210, 225, 227, 234, 237, 244, 248, 249, 253, 254, 258, 259, 260, 264, 265,
268, 269, 272, 274, 278, 279, 280, 283, 285, 286, 289, 291, 292, 293, 295, 29
6, 297, 302, 303, 307, 310, 312, 314, 315, ...], 'Family': [688, 698], 'Fanta
sy': [321, 568], 'Film-Noir': [309, 456, 712], 'Horror': [49, 75, 271, 419, 5
44, 707, 724, 844, 876, 932, 948], 'Mystery': [69, 81, 119, 145, 220, 393, 42
0, 714, 829, 899, 959, 961], 'Thriller': [700], 'Western': [12, 48, 115, 69
1]}
```

In [25]:
```
# describe
movies.groupby('Genre').describe()
```

Out[25]:

	Runtime								IMDB_Rating	.
	count	mean	std	min	25%	50%	75%	max	count	mean	.
Genre											
Action	172.0	129.046512	28.500706	45.0	110.75	127.5	143.25	321.0	172.0	7.949419	.
Adventure	72.0	134.111111	33.317320	88.0	109.00	127.0	149.00	228.0	72.0	7.937500	.
Animation	82.0	99.585366	14.530471	71.0	90.00	99.5	106.75	137.0	82.0	7.930488	.
Biography	88.0	136.022727	25.514466	93.0	120.00	129.0	146.25	209.0	88.0	7.938636	.
Comedy	155.0	112.129032	22.946213	68.0	96.00	106.0	124.50	188.0	155.0	7.901290	.
Crime	107.0	126.392523	27.689231	80.0	106.50	122.0	141.50	229.0	107.0	8.016822	.
Drama	289.0	124.737024	27.740490	64.0	105.00	121.0	137.00	242.0	289.0	7.957439	.
Family	2.0	107.500000	10.606602	100.0	103.75	107.5	111.25	115.0	2.0	7.800000	.
Fantasy	2.0	85.000000	12.727922	76.0	80.50	85.0	89.50	94.0	2.0	8.000000	.
Film-Noir	3.0	104.000000	4.000000	100.0	102.00	104.0	106.00	108.0	3.0	7.966667	.
Horror	11.0	102.090909	13.604812	71.0	98.00	103.0	109.00	122.0	11.0	7.909091	.
Mystery	12.0	119.083333	14.475423	96.0	110.75	117.5	130.25	138.0	12.0	7.975000	.
Thriller	1.0	108.000000	NaN	108.0	108.00	108.0	108.00	108.0	1.0	7.800000	.
Western	4.0	148.250000	17.153717	132.0	134.25	148.0	162.00	165.0	4.0	8.350000	.

14 rows × 40 columns

In [30]:
```
# sample
movies.groupby('Genre').sample(2,replace=True)
```

Out[30]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_
308	White Heat	1949	114	Action	8.1	Raoul Walsh	James Cagney	
171	Die Hard	1988	132	Action	8.2	John McTiernan	Bruce Willis	
300	Ben-Hur	1959	212	Adventure	8.1	William Wyler	Charlton Heston	
47	Back to the Future	1985	116	Adventure	8.5	Robert Zemeckis	Michael J. Fox	
640	Les triplettes de Belleville	2003	80	Animation	7.8	Sylvain Chomet	Michèle Caucheteux	
799	South Park: Bigger, Longer & Uncut	1999	81	Animation	7.7	Trey Parker	Trey Parker	
632	The World's Fastest Indian	2005	127	Biography	7.8	Roger Donaldson	Anthony Hopkins	
372	Mar adentro	2014	126	Biography	8.0	Alejandro Amenábar	Javier Bardem	
846	Love and Death	1975	85	Comedy	7.7	Woody Allen	Woody Allen	
660	The Sandlot	1993	101	Comedy	7.8	David Mickey Evans	Tom Guiry	
397	Bound by Honor	1993	180	Crime	8.0	Taylor Hackford	Damian Chapa	
108	Scarface	1983	170	Crime	8.3	Brian De Palma	Al Pacino	
53	Capharnaüm	2018	126	Drama	8.4	Nadine Labaki	Zain Al Rafeea	
894	Creed	2015	133	Drama	7.6	Ryan Coogler	Michael B. Jordan	
688	E.T. the Extra-Terrestrial	1982	115	Family	7.8	Steven Spielberg	Henry Thomas	
688	E.T. the Extra-Terrestrial	1982	115	Family	7.8	Steven Spielberg	Henry Thomas	
568	Nosferatu	1922	94	Fantasy	7.9	F.W. Murnau	Max Schreck	
321	Das Cabinet des Dr. Caligari	1920	76	Fantasy	8.1	Robert Wiene	Werner Krauss	
309	The Third Man	1949	104	Film-Noir	8.1	Carol Reed	Orson Welles	
712	Shadow of a Doubt	1943	108	Film-Noir	7.8	Alfred Hitchcock	Teresa Wright	
49	Psycho	1960	109	Horror	8.5	Alfred Hitchcock	Anthony Perkins	

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_
948	The Others	2001	101	Horror	7.6	Alejandro Amenábar	Nicole Kidman	
81	Rear Window	1954	112	Mystery	8.4	Alfred Hitchcock	James Stewart	
959	Dark City	1998	100	Mystery	7.6	Alex Proyas	Rufus Sewell	
700	Wait Until Dark	1967	108	Thriller	7.8	Terence Young	Audrey Hepburn	
700	Wait Until Dark	1967	108	Thriller	7.8	Terence Young	Audrey Hepburn	
691	The Outlaw Josey Wales	1976	135	Western	7.8	Clint Eastwood	Clint Eastwood	
48	Once Upon a Time in the West	1968	165	Western	8.5	Sergio Leone	Henry Fonda	

In [31]:
```
# nunique
movies.groupby('Genre').nunique() # unique --> unique items , nunique--> gives
```

Out[31]:

Genre	Series_Title	Released_Year	Runtime	IMDB_Rating	Director	Star1	No_of_Votes	Gr
Action	172	61	78	15	123	121	172	
Adventure	72	49	58	10	59	59	72	
Animation	82	35	41	11	51	77	82	
Biography	88	44	56	13	76	72	88	
Comedy	155	72	70	11	113	133	155	
Crime	106	56	65	14	86	85	107	
Drama	289	83	95	14	211	250	288	
Family	2	2	2	1	2	2	2	
Fantasy	2	2	2	2	2	2	2	
Film-Noir	3	3	3	3	3	3	3	
Horror	11	11	10	8	10	11	11	
Mystery	12	11	10	8	10	11	12	
Thriller	1	1	1	1	1	1	1	
Western	4	4	4	4	2	2	4	

aggregate method

In [33]:
```
# passing dict
movies.groupby('Genre').agg(
    {
        'Runtime':'mean',
        'IMDB_Rating':'mean',
        'No_of_Votes':'sum',
        'Gross':'sum',
        'Metascore':'min'

    }

)
```

Out[33]:

Genre	Runtime	IMDB_Rating	No_of_Votes	Gross	Metascore
Action	129.046512	7.949419	72282412	3.263226e+10	33.0
Adventure	134.111111	7.937500	22576163	9.496922e+09	41.0
Animation	99.585366	7.930488	21978630	1.463147e+10	61.0
Biography	136.022727	7.938636	24006844	8.276358e+09	48.0
Comedy	112.129032	7.901290	27620327	1.566387e+10	45.0
Crime	126.392523	8.016822	33533615	8.452632e+09	47.0
Drama	124.737024	7.957439	61367304	3.540997e+10	28.0
Family	107.500000	7.800000	551221	4.391106e+08	67.0
Fantasy	85.000000	8.000000	146222	7.827267e+08	NaN
Film-Noir	104.000000	7.966667	367215	1.259105e+08	94.0
Horror	102.090909	7.909091	3742556	1.034649e+09	46.0
Mystery	119.083333	7.975000	4203004	1.256417e+09	52.0
Thriller	108.000000	7.800000	27733	1.755074e+07	81.0
Western	148.250000	8.350000	1289665	5.822151e+07	69.0

In [37]:
```
#Passsing List

movies.groupby('Genre').agg(['min','max','mean','sum','median'])
```

Out[37]:

	Runtime					IMDB_Rating					...
	min	max	mean	sum	median	min	max	mean	sum	median	...
Genre											
Action	45	321	129.046512	22196	127.5	7.6	9.0	7.949419	1367.3	7.9	...
Adventure	88	228	134.111111	9656	127.0	7.6	8.6	7.937500	571.5	7.9	...
Animation	71	137	99.585366	8166	99.5	7.6	8.6	7.930488	650.3	7.9	...
Biography	93	209	136.022727	11970	129.0	7.6	8.9	7.938636	698.6	7.9	...
Comedy	68	188	112.129032	17380	106.0	7.6	8.6	7.901290	1224.7	7.9	...
Crime	80	229	126.392523	13524	122.0	7.6	9.2	8.016822	857.8	8.0	...
Drama	64	242	124.737024	36049	121.0	7.6	9.3	7.957439	2299.7	8.0	...
Family	100	115	107.500000	215	107.5	7.8	7.8	7.800000	15.6	7.8	...
Fantasy	76	94	85.000000	170	85.0	7.9	8.1	8.000000	16.0	8.0	...
Film-Noir	100	108	104.000000	312	104.0	7.8	8.1	7.966667	23.9	8.0	...
Horror	71	122	102.090909	1123	103.0	7.6	8.5	7.909091	87.0	7.8	...
Mystery	96	138	119.083333	1429	117.5	7.6	8.4	7.975000	95.7	8.0	...
Thriller	108	108	108.000000	108	108.0	7.8	7.8	7.800000	7.8	7.8	...
Western	132	165	148.250000	593	148.0	7.8	8.8	8.350000	33.4	8.4	...

14 rows × 25 columns

In [39]:
```
# Adding both the syntax
movies.groupby('Genre').agg(
    {
        'Runtime':['min','mean'],
        'IMDB_Rating':'mean',
        'No_of_Votes':['sum','max'],
        'Gross':'sum',
        'Metascore':'min'
    }
)
```

Out[39]:

	Runtime		IMDB_Rating	No_of_Votes		Gross	Metascore
	min	mean	mean	sum	max	sum	min
Genre							
Action	45	129.046512	7.949419	72282412	2303232	3.263226e+10	33.0
Adventure	88	134.111111	7.937500	22576163	1512360	9.496922e+09	41.0
Animation	71	99.585366	7.930488	21978630	999790	1.463147e+10	61.0
Biography	93	136.022727	7.938636	24006844	1213505	8.276358e+09	48.0
Comedy	68	112.129032	7.901290	27620327	939631	1.566387e+10	45.0
Crime	80	126.392523	8.016822	33533615	1826188	8.452632e+09	47.0
Drama	64	124.737024	7.957439	61367304	2343110	3.540997e+10	28.0
Family	100	107.500000	7.800000	551221	372490	4.391106e+08	67.0
Fantasy	76	85.000000	8.000000	146222	88794	7.827267e+08	NaN
Film-Noir	100	104.000000	7.966667	367215	158731	1.259105e+08	94.0
Horror	71	102.090909	7.909091	3742556	787806	1.034649e+09	46.0
Mystery	96	119.083333	7.975000	4203004	1129894	1.256417e+09	52.0
Thriller	108	108.000000	7.800000	27733	27733	1.755074e+07	81.0
Western	132	148.250000	8.350000	1289665	688390	5.822151e+07	69.0

In [40]:
```
# looping on groups
for group , data in movies.groupby('Genre'):
    print(data)
```

```
                                        Series_Title Released_Year  Runt
ime  \
2                                   The Dark Knight          2008
152
5       The Lord of the Rings: The Return of the King          2003
201
8                                         Inception          2010
148
10   The Lord of the Rings: The Fellowship of the Ring          2001
178
13              The Lord of the Rings: The Two Towers          2002
179
..                                               ...           ...
...
968                                  Falling Down          1993
113
979                                  Lethal Weapon          1987
109
982                                      Mad Max 2          1981
```

In [41]:
```
# find the highest rated movie of each genre

df = pd.DataFrame(columns=movies.columns)

for group , data in movies.groupby('Genre'):
    df= df.append(data[data['IMDB_Rating'] == data['IMDB_Rating'].max()])
df
```

Out[41]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1
2	The Dark Knight	2008	152	Action	9.0	Christopher Nolan	Christian Bale
21	Interstellar	2014	169	Adventure	8.6	Christopher Nolan	Matthew McConaughey
23	Sen to Chihiro no kamikakushi	2001	125	Animation	8.6	Hayao Miyazaki	Daveigh Chase
7	Schindler's List	1993	195	Biography	8.9	Steven Spielberg	Liam Neeson
19	Gisaengchung	2019	132	Comedy	8.6	Bong Joon Ho	Kang-ho Song
26	La vita è bella	1997	116	Comedy	8.6	Roberto Benigni	Roberto Benigni
1	The Godfather	1972	175	Crime	9.2	Francis Ford Coppola	Marlon Brando
0	The Shawshank Redemption	1994	142	Drama	9.3	Frank Darabont	Tim Robbins
688	E.T. the Extra-Terrestrial	1982	115	Family	7.8	Steven Spielberg	Henry Thomas
698	Willy Wonka & the Chocolate Factory	1971	100	Family	7.8	Mel Stuart	Gene Wilder
321	Das Cabinet des Dr. Caligari	1920	76	Fantasy	8.1	Robert Wiene	Werner Krauss
309	The Third Man	1949	104	Film-Noir	8.1	Carol Reed	Orson Welles
49	Psycho	1960	109	Horror	8.5	Alfred Hitchcock	Anthony Perkins
69	Memento	2000	113	Mystery	8.4	Christopher Nolan	Guy Pearce
81	Rear Window	1954	112	Mystery	8.4	Alfred Hitchcock	James Stewart
700	Wait Until Dark	1967	108	Thriller	7.8	Terence Young	Audrey Hepburn
12	Il buono, il brutto, il cattivo	1966	161	Western	8.8	Sergio Leone	Clint Eastwood

split (apply) combine

In [42]:
```
# apply -> builtin function
movies.groupby('Genre').apply(min)
```

Out[42]:

Genre	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1
Action	300	1924	45	Action	7.6	Abhishek Chaubey	Aamir Khan
Adventure	2001: A Space Odyssey	1925	88	Adventure	7.6	Akira Kurosawa	Aamir Khan
Animation	Akira	1940	71	Animation	7.6	Adam Elliot	Adrian Molina
Biography	12 Years a Slave	1928	93	Biography	7.6	Adam McKay	Adrien Brody
Comedy	(500) Days of Summer	1921	68	Comedy	7.6	Alejandro G. Iñárritu	Aamir Khan
Crime	12 Angry Men	1931	80	Crime	7.6	Akira Kurosawa	Ajay Devgn
Drama	1917	1925	64	Drama	7.6	Aamir Khan	Abhay Deol
Family	E.T. the Extra-Terrestrial	1971	100	Family	7.8	Mel Stuart	Gene Wilder
Fantasy	Das Cabinet des Dr. Caligari	1920	76	Fantasy	7.9	F.W. Murnau	Max Schreck
Film-Noir	Shadow of a Doubt	1941	100	Film-Noir	7.8	Alfred Hitchcock	Humphrey Bogart
Horror	Alien	1933	71	Horror	7.6	Alejandro Amenábar	Anthony Perkins
Mystery	Dark City	1938	96	Mystery	7.6	Alex Proyas	Bernard-Pierre Donnadieu
Thriller	Wait Until Dark	1967	108	Thriller	7.8	Terence Young	Audrey Hepburn
Western	Il buono, il brutto, il cattivo	1965	132	Western	7.8	Clint Eastwood	Clint Eastwood

In [43]:
```
# find number of movies starting with A for each group
def foo(group):
    print(group) # type = Dataframe
    return group
```

In [44]:
```
movies.groupby('Genre').apply(foo)
```

```
                                          Series_Title Released_Year  Runt
ime  \
2                                    The Dark Knight          2008
152
5       The Lord of the Rings: The Return of the King          2003
201
8                                         Inception          2010
148
10   The Lord of the Rings: The Fellowship of the Ring          2001
178
13              The Lord of the Rings: The Two Towers          2002
179
..                                               ...           ...
...
968                                  Falling Down          1993
113
979                                  Lethal Weapon          1987
109
```

In [49]:
```
#Custom Logic
def foo(group):
     return group['Series_Title'].str.startswith('A').sum()
```

In [50]:
```
movies.groupby('Genre').apply(foo)
```

Out[50]:
```
Genre
Action       10
Adventure     2
Animation     2
Biography     9
Comedy       14
Crime         4
Drama        21
Family        0
Fantasy       0
Film-Noir     0
Horror        1
Mystery       0
Thriller      0
Western       0
dtype: int64
```

In [51]:
```
# find ranking of each movie in the group according to IMDB score
def rank_movie(group):
    group['genre_rank']=group['IMDB_Rating'].rank(ascending=False)
    return group
```

In [52]:
```
movies.groupby('Genre').apply(rank_movie)
```

Out[52]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_
0	The Shawshank Redemption	1994	142	Drama	9.3	Frank Darabont	Tim Robbins	23
1	The Godfather	1972	175	Crime	9.2	Francis Ford Coppola	Marlon Brando	162
2	The Dark Knight	2008	152	Action	9.0	Christopher Nolan	Christian Bale	230
3	The Godfather: Part II	1974	202	Crime	9.0	Francis Ford Coppola	Al Pacino	112
4	12 Angry Men	1957	96	Crime	9.0	Sidney Lumet	Henry Fonda	68
...	...	...	...	...	...	...	...	
995	Breakfast at Tiffany's	1961	115	Comedy	7.6	Blake Edwards	Audrey Hepburn	16
996	Giant	1956	201	Drama	7.6	George Stevens	Elizabeth Taylor	3
997	From Here to Eternity	1953	118	Drama	7.6	Fred Zinnemann	Burt Lancaster	4
998	Lifeboat	1944	97	Drama	7.6	Alfred Hitchcock	Tallulah Bankhead	2
999	The 39 Steps	1935	86	Crime	7.6	Alfred Hitchcock	Robert Donat	5

1000 rows × 11 columns

In [55]:
```
 # find normalized IMDB rating group wise
#x normalized = (x - x minimum) / (x maximum - x minimum)
def normal(group):
    group['normal_rating'] = (group['IMDB_Rating']- group['IMDB_Rating'].min()
    return group

movies.groupby('Genre').apply(normal)
```

Out[55]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_
0	The Shawshank Redemption	1994	142	Drama	9.3	Frank Darabont	Tim Robbins	23
1	The Godfather	1972	175	Crime	9.2	Francis Ford Coppola	Marlon Brando	16
2	The Dark Knight	2008	152	Action	9.0	Christopher Nolan	Christian Bale	23
3	The Godfather: Part II	1974	202	Crime	9.0	Francis Ford Coppola	Al Pacino	11
4	12 Angry Men	1957	96	Crime	9.0	Sidney Lumet	Henry Fonda	6
...	...	...	...	...	...	...	...	
995	Breakfast at Tiffany's	1961	115	Comedy	7.6	Blake Edwards	Audrey Hepburn	1
996	Giant	1956	201	Drama	7.6	George Stevens	Elizabeth Taylor	
997	From Here to Eternity	1953	118	Drama	7.6	Fred Zinnemann	Burt Lancaster	
998	Lifeboat	1944	97	Drama	7.6	Alfred Hitchcock	Tallulah Bankhead	
999	The 39 Steps	1935	86	Crime	7.6	Alfred Hitchcock	Robert Donat	

1000 rows × 11 columns

groupby on multiple cols

In [57]:
```
combo = movies.groupby(['Director','Star1'])
#size
combo.size()
```

Out[57]:
```
Director             Star1
Aamir Khan           Amole Gupte       1
Aaron Sorkin         Eddie Redmayne    1
Abdellatif Kechiche  Léa Seydoux       1
Abhishek Chaubey     Shahid Kapoor     1
Abhishek Kapoor      Amit Sadh         1
                                      ..
Zaza Urushadze       Lembit Ulfsak     1
Zoya Akhtar          Hrithik Roshan    1
                     Vijay Varma       1
Çagan Irmak          Çetin Tekindor    1
Ömer Faruk Sorak     Cem Yilmaz        1
Length: 898, dtype: int64
```

In [59]:
```
#if we want particular combo---> get_group
combo.get_group(('Aamir Khan','Amole Gupte'))
```

Out[59]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_Votes	
65	Taare Zameen Par	2007	165	Drama	8.4	Aamir Khan	Amole Gupte	168895	1

In [62]:
```
# find the most earning actor-> director combo
combo['Gross'].sum().sort_values(ascending=False).head(2)
```

Out[62]:
```
Director        Star1
Akira Kurosawa  Toshirô Mifune    2.999877e+09
Anthony Russo   Joe Russo         2.205039e+09
Name: Gross, dtype: float64
```

In [75]:
```
# find the best(in-terms of metascore(avg)) actor->genre combo
movies.groupby(['Star1','Genre'])['Metascore'].mean().reset_index().sort_value
```

Out[75]:

	Star1	Genre	Metascore
230	Ellar Coltrane	Drama	100.0

In [77]:
```
# agg on multiple groupby
combo.agg(['min','max','mean'])
```

Out[77]:

		Runtime			IMDB_Rating			No_of_Votes			
		min	max	mean	min	max	mean	min	max	mean	n
Director	Star1										
Aamir Khan	Amole Gupte	165	165	165.0	8.4	8.4	8.4	168895	168895	168895.0	122386
Aaron Sorkin	Eddie Redmayne	129	129	129.0	7.8	7.8	7.8	89896	89896	89896.0	85309041
Abdellatif Kechiche	Léa Seydoux	180	180	180.0	7.7	7.7	7.7	138741	138741	138741.0	219967
Abhishek Chaubey	Shahid Kapoor	148	148	148.0	7.8	7.8	7.8	27175	27175	27175.0	21842830
Abhishek Kapoor	Amit Sadh	130	130	130.0	7.7	7.7	7.7	32628	32628	32628.0	112252
...	...	...	...	...	...	...	...	...	...	...	
Zaza Urushadze	Lembit Ulfsak	87	87	87.0	8.2	8.2	8.2	40382	40382	40382.0	14450
Zoya Akhtar	Hrithik Roshan	155	155	155.0	8.1	8.1	8.1	67927	67927	67927.0	310848
	Vijay Varma	154	154	154.0	8.0	8.0	8.0	31886	31886	31886.0	556653
Çagan Irmak	Çetin Tekindor	112	112	112.0	8.3	8.3	8.3	78925	78925	78925.0	46185536
Ömer Faruk Sorak	Cem Yilmaz	127	127	127.0	8.0	8.0	8.0	56960	56960	56960.0	19620607

898 rows × 15 columns

Excercise

In [78]:
```
ipl = pd.read_csv("deliveries.csv")
ipl.head(2)
```

Out[78]:

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_s
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	1	DA Warner	S Dhawan	TS Mills	
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	2	DA Warner	S Dhawan	TS Mills	

2 rows × 21 columns

In [80]:
```
ipl.shape # ball by ball dataset
```

Out[80]:
```
(179078, 21)
```

In [87]:
```
# find the top 10 batsman in terms of runs
ipl.groupby('batsman')['batsman_runs'].sum().sort_values(ascending=False).rese
```

Out[87]:

	batsman	batsman_runs
0	V Kohli	5434
1	SK Raina	5415
2	RG Sharma	4914
3	DA Warner	4741
4	S Dhawan	4632
5	CH Gayle	4560
6	MS Dhoni	4477
7	RV Uthappa	4446
8	AB de Villiers	4428
9	G Gambhir	4223

In [94]:
```
# find the batsman with max no of sixes
six = ipl[ipl['batsman_runs']==6]

six.groupby('batsman')['batsman'].count().sort_values(ascending=False).head(2)
```

Out[94]:
```
batsman
CH Gayle          327
AB de Villiers    214
Name: batsman, dtype: int64
```

In [105]:
```
# find batsman with most number of 4's and 6's in last 5 overs

temp = ipl[ipl['over']>15]
temp = temp[(temp['batsman_runs'] == 4) | (temp['batsman_runs'] == 6)]

temp.groupby('batsman')['batsman'].count().sort_values(ascending=False).head(1
```

Out[105]: 'MS Dhoni'

In [110]:
```
# find V Kohli's record against all teams

temp = ipl[ipl['batsman'] == 'V Kohli']

temp.groupby('bowling_team')['batsman_runs'].sum().sort_values(ascending=False
```

Out[110]:

	bowling_team	batsman_runs
0	Delhi Daredevils	763
1	Chennai Super Kings	749

In [118]:
```
# Create a function that can return the highest score of any batsman

temp = ipl[ipl['batsman'] == 'V Kohli']

temp.groupby('match_id')['batsman_runs'].sum().sort_values(ascending=False).he
```

Out[118]: 113

In [122]:
```
def highest(batsman):
    temp = ipl[ipl['batsman'] == batsman]
    return  temp.groupby('match_id')['batsman_runs'].sum().sort_values(ascendi
```

In [123]:
```
highest('DA Warner')
```

Out[123]: 126

In [ ]:
```

```

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

In [2]:
```
courses = pd.read_csv("courses.csv")
students = pd.read_csv("students.csv")
may = pd.read_csv("reg-month1.csv")
june = pd.read_csv("reg-month2.csv")
matches = pd.read_csv("matches.csv")
deliveries = pd.read_csv("deliveries.csv")
```

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

Out[3]:

	course_id	course_name	price
0	1	python	2499
1	2	sql	3499

In [4]:
```
students.head(2)
```

Out[4]:

	student_id	name	partner
0	1	Kailash Harjo	23
1	2	Esha Butala	1

In [5]:
```
may.head(2)
```

Out[5]:

	student_id	course_id
0	23	1
1	15	5

In [6]:
```
june.head(2)
```

Out[6]:

	student_id	course_id
0	3	5
1	16	7

In [7]:
```
matches.head(2)
```

Out[7]:

	id	season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets	player_of_ma
0	1	2017	Hyderabad	2017-04-05	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Sunrisers Hyderabad	35	0	Yuvraj Si
1	2	2017	Pune	2017-04-06	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	0	Rising Pune Supergiant	0	7	SPD Sr

Concat

it is a powerful function that allows you to concatenate two or more DataFrames along a particular axis (row-wise or column-wise). You can control how the data is concatenated by specifying several parameters, such as axis, join, ignore_index, and keys.

In [8]:
```
regs = pd.concat([may,june],ignore_index=True) # Vertically merged
regs
```

Out[8]:

	student_id	course_id
0	23	1
1	15	5
2	18	6
3	23	4
4	16	9
5	18	1
6	1	1
7	7	8
8	22	3
9	15	1
10	19	4
11	1	6
12	7	10
13	11	7
14	13	3
15	24	4
16	21	1
17	16	5
18	23	3
19	17	7
20	23	6
21	25	1
22	19	2
23	25	10
24	3	3
25	3	5
26	16	7
27	12	10
28	12	1
29	14	9
30	7	7
31	7	2
32	16	3
33	17	10
34	11	8
35	14	6
36	12	5
37	12	7
38	18	8
39	1	10
40	1	9
41	2	5
42	7	6
43	22	5
44	22	6
45	23	9
46	23	5
47	14	4
48	14	1
49	11	10
50	42	9
51	50	8
52	38	1

In [9]:
```
# Multi_index DataFrame

multi = pd.concat([may,june],keys=['may','june'])
multi
```

Out[9]:

		student_id	course_id
may	0	23	1
	1	15	5
	2	18	6
	3	23	4
	4	16	9
	5	18	1
	6	1	1
	7	7	8
	8	22	3
	9	15	1
	10	19	4
	11	1	6
	12	7	10
	13	11	7
	14	13	3
	15	24	4
	16	21	1
	17	16	5
	18	23	3
	19	17	7
	20	23	6
	21	25	1
	22	19	2
	23	25	10
	24	3	3
june	0	3	5
	1	16	7
	2	12	10
	3	12	1
	4	14	9
	5	7	7
	6	7	2
	7	16	3
	8	17	10
	9	11	8
	10	14	6
	11	12	5
	12	12	7
	13	18	8
	14	1	10
	15	1	9
	16	2	5
	17	7	6
	18	22	5
	19	22	6
	20	23	9
	21	23	5
	22	14	4
	23	14	1
	24	11	10
	25	42	9
	26	50	8
	27	38	1

In [10]:
```
multi.loc['may']
```

Out[10]:

	student_id	course_id
0	23	1
1	15	5
2	18	6
3	23	4
4	16	9
5	18	1
6	1	1
7	7	8
8	22	3
9	15	1
10	19	4
11	1	6
12	7	10
13	11	7
14	13	3
15	24	4
16	21	1
17	16	5
18	23	3
19	17	7
20	23	6
21	25	1
22	19	2
23	25	10
24	3	3

In [11]:
```
multi.loc[('june',0)]
```

Out[11]:
```
student_id    3
course_id     5
Name: (june, 0), dtype: int64
```

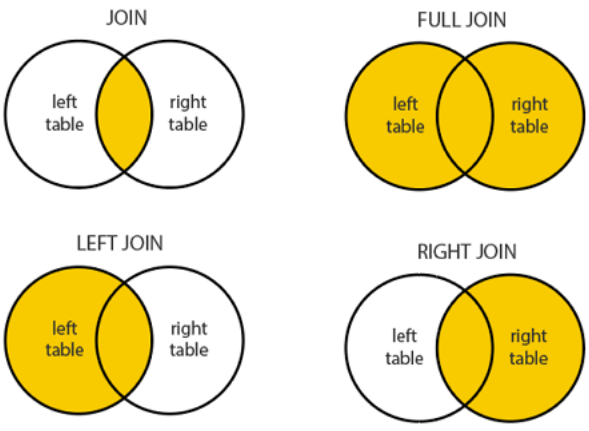
In [12]:
```
# Horizontally placed

pd.concat([may,june],axis=1)
```

Out[12]:

	student_id	course_id	student_id	course_id
0	23.0	1.0	3	5
1	15.0	5.0	16	7
2	18.0	6.0	12	10
3	23.0	4.0	12	1
4	16.0	9.0	14	9
5	18.0	1.0	7	7
6	1.0	1.0	7	2
7	7.0	8.0	16	3
8	22.0	3.0	17	10
9	15.0	1.0	11	8
10	19.0	4.0	14	6
11	1.0	6.0	12	5
12	7.0	10.0	12	7
13	11.0	7.0	18	8
14	13.0	3.0	1	10
15	24.0	4.0	1	9
16	21.0	1.0	2	5
17	16.0	5.0	7	6
18	23.0	3.0	22	5
19	17.0	7.0	22	6
20	23.0	6.0	23	9
21	25.0	1.0	23	5
22	19.0	2.0	14	4
23	25.0	10.0	14	1
24	3.0	3.0	11	10
25	NaN	NaN	42	9
26	NaN	NaN	50	8
27	NaN	NaN	38	1

Merge

On Joins



Inner Join

For joining any data ,

In each set of data, there should to be a "common" column. Students[student_id] and regs[student_id] are listed here. We combine based on the student_id, however the inner join only displays the data that is "Common" across the two dataframes.

In [13]:
```
students.merge(regs, how= 'inner' , on = 'student_id').tail()
```

Out[13]:

	student_id	name	partner	course_id
45	23	Chhavi Lachman	18	9
46	23	Chhavi Lachman	18	5
47	24	Radhika Suri	17	4
48	25	Shashank D'Alia	2	1
49	25	Shashank D'Alia	2	10

Left Join

Here we have same column --- > course_id

on basis on this we can merge using left join.

Regardless of whether or not the right side data leaves, it prints all of the left side data. so , we can see left data (Numpy , c++) but we cannot see any right side data which is student_id here, courses reflect = Left and regs reflect = right

In [14]:
```
courses.merge(regs,how='left',on='course_id').tail(5)
```

Out[14]:

	course_id	course_name	price	student_id
50	10	pyspark	2499	17.0
51	10	pyspark	2499	1.0
52	10	pyspark	2499	11.0
53	11	Numpy	699	NaN
54	12	C++	1299	NaN

Right join

In [15]:
```
temp_df = pd.DataFrame({
    'student_id':[26,27,28],
    'name':['Nitish','Ankit','Rahul'],
    'partner':[28,26,17]
})

students = pd.concat([students,temp_df],ignore_index=True)
```

In [16]:
```
students.tail()
```

Out[16]:

	student_id	name	partner
23	24	Radhika Suri	17
24	25	Shashank D'Alia	2
25	26	Nitish	28
26	27	Ankit	26
27	28	Rahul	17

Regs data(50,51,52) in the current case does not contain students data, however even this, data is printed since the join was done right.

why.?
because when using a right join, all right side data is printed regardless of whether the left side data exits or not.

here right reflects = regs , Left reflects = students

In [17]:
```
students.merge(regs, how='right',on='student_id').tail(5)
```

Out[17]:

	student_id	name	partner	course_id
48	14	Pranab Natarajan	22.0	1
49	11	David Mukhopadhyay	20.0	10
50	42	NaN	NaN	9
51	50	NaN	NaN	8
52	38	NaN	NaN	1

Since there is no course_id in the student data in the current case, "Nan" data is displayed.

Why was a left join performed using the student_id? Regardless of whether or not the right side data leaves, it prints all of the left side data.

here Left reflects = students , right reflects = regs

In [18]:
```
students.merge(regs, how='left',on='student_id').tail(5)
```

Out[18]:

	student_id	name	partner	course_id
55	25	Shashank D'Alia	2	1.0
56	25	Shashank D'Alia	2	10.0
57	26	Nitish	28	NaN
58	27	Ankit	26	NaN
59	28	Rahul	17	NaN

Outer join

Initially the left join data is clearly apparent with (Nitish, Ankit, Rahul) data written,

but the right side data (course id) is blank. like which,

Right join shows Nan even though we don't have any data for (42, 50, 38), but we can see the course's id column because it's a right join.

Finally, we may view both data sets, both common and individual, regardless of whether they have ever been. As seen in the outer join

In [19]:
```
students.merge(regs ,how ='outer', on= 'student_id' ).tail(10)
```

Out[19]:

	student_id	name	partner	course_id
53	23	Chhavi Lachman	18.0	5.0
54	24	Radhika Suri	17.0	4.0
55	25	Shashank D'Alia	2.0	1.0
56	25	Shashank D'Alia	2.0	10.0
57	26	Nitish	28.0	NaN
58	27	Ankit	26.0	NaN
59	28	Rahul	17.0	NaN
60	42	NaN	NaN	9.0
61	50	NaN	NaN	8.0
62	38	NaN	NaN	1.0

In [20]:
```
# 1. find total revenue generated
regs.merge(courses, how = 'inner' , on = 'course_id')['price'].sum()
```

Out[20]:
```
154247
```

In [27]:
```
# 2. find month by month revenue
temp = pd.concat([may,june], keys=['may','june']).reset_index()
temp.merge(courses,on = 'course_id').groupby('level_0')['price'].sum()
```

Out[27]:
```
level_0
june    65072
may     89175
Name: price, dtype: int64
```

In [32]:
```
# 3. Print the registration table
# cols -> name -> course -> price

regs.merge(students, on = 'student_id').merge(courses , on='course_id')
```

Out[32]:

	student_id	course_id	name	partner	course_name	price
0	23	1	Chhavi Lachman	18	python	2499
1	15	1	Preet Sha	16	python	2499
2	18	1	Fardeen Mahabir	13	python	2499
3	1	1	Kailash Harjo	23	python	2499
4	21	1	Seema Kota	15	python	2499
5	25	1	Shashank D'Alia	2	python	2499
6	12	1	Radha Dutt	19	python	2499
7	14	1	Pranab Natarajan	22	python	2499
8	23	4	Chhavi Lachman	18	machine learning	9999
9	19	4	Qabeel Raman	12	machine learning	9999
10	24	4	Radhika Suri	17	machine learning	9999
11	14	4	Pranab Natarajan	22	machine learning	9999
12	23	3	Chhavi Lachman	18	data analysis	4999
13	16	3	Elias Dodiya	25	data analysis	4999
14	22	3	Yash Sethi	21	data analysis	4999
15	13	3	Munni Varghese	24	data analysis	4999
16	3	3	Parveen Bhalla	3	data analysis	4999
17	23	6	Chhavi Lachman	18	power bi	1899
18	18	6	Fardeen Mahabir	13	power bi	1899
19	1	6	Kailash Harjo	23	power bi	1899
20	7	6	Tarun Thaker	9	power bi	1899
21	22	6	Yash Sethi	21	power bi	1899
22	14	6	Pranab Natarajan	22	power bi	1899
23	23	9	Chhavi Lachman	18	plotly	699
24	16	9	Elias Dodiya	25	plotly	699
25	1	9	Kailash Harjo	23	plotly	699
26	14	9	Pranab Natarajan	22	plotly	699
27	23	5	Chhavi Lachman	18	tableau	2499
28	15	5	Preet Sha	16	tableau	2499
29	16	5	Elias Dodiya	25	tableau	2499
30	22	5	Yash Sethi	21	tableau	2499
31	3	5	Parveen Bhalla	3	tableau	2499
32	12	5	Radha Dutt	19	tableau	2499
33	2	5	Esha Butala	1	tableau	2499
34	18	8	Fardeen Mahabir	13	pandas	1099
35	7	8	Tarun Thaker	9	pandas	1099
36	11	8	David Mukhopadhyay	20	pandas	1099
37	16	7	Elias Dodiya	25	ms sxcel	1599
38	7	7	Tarun Thaker	9	ms sxcel	1599
39	11	7	David Mukhopadhyay	20	ms sxcel	1599
40	17	7	Yasmin Palan	7	ms sxcel	1599
41	12	7	Radha Dutt	19	ms sxcel	1599
42	1	10	Kailash Harjo	23	pyspark	2499
43	7	10	Tarun Thaker	9	pyspark	2499
44	11	10	David Mukhopadhyay	20	pyspark	2499
45	17	10	Yasmin Palan	7	pyspark	2499
46	25	10	Shashank D'Alia	2	pyspark	2499
47	12	10	Radha Dutt	19	pyspark	2499
48	7	2	Tarun Thaker	9	sql	3499
49	19	2	Qabeel Raman	12	sql	3499

In [33]:
```
regs.merge(students, on = 'student_id').merge(courses , on='course_id')[['name','course_name','price']]
```

Out[33]:

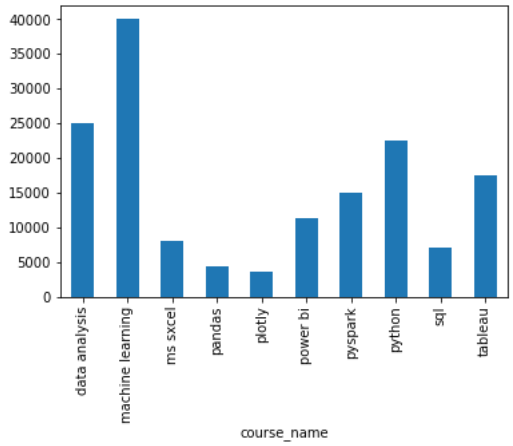
	name	course_name	price
0	Chhavi Lachman	python	2499
1	Preet Sha	python	2499
2	Fardeen Mahabir	python	2499
3	Kailash Harjo	python	2499
4	Seema Kota	python	2499
5	Shashank D'Alia	python	2499
6	Radha Dutt	python	2499
7	Pranab Natarajan	python	2499
8	Chhavi Lachman	machine learning	9999
9	Qabeel Raman	machine learning	9999
10	Radhika Suri	machine learning	9999
11	Pranab Natarajan	machine learning	9999
12	Chhavi Lachman	data analysis	4999
13	Elias Dodiya	data analysis	4999
14	Yash Sethi	data analysis	4999
15	Munni Varghese	data analysis	4999
16	Parveen Bhalla	data analysis	4999
17	Chhavi Lachman	power bi	1899
18	Fardeen Mahabir	power bi	1899
19	Kailash Harjo	power bi	1899
20	Tarun Thaker	power bi	1899
21	Yash Sethi	power bi	1899
22	Pranab Natarajan	power bi	1899
23	Chhavi Lachman	plotly	699
24	Elias Dodiya	plotly	699
25	Kailash Harjo	plotly	699
26	Pranab Natarajan	plotly	699
27	Chhavi Lachman	tableau	2499
28	Preet Sha	tableau	2499
29	Elias Dodiya	tableau	2499
30	Yash Sethi	tableau	2499
31	Parveen Bhalla	tableau	2499
32	Radha Dutt	tableau	2499
33	Esha Butala	tableau	2499
34	Fardeen Mahabir	pandas	1099
35	Tarun Thaker	pandas	1099
36	David Mukhopadhyay	pandas	1099
37	Elias Dodiya	ms sxcel	1599
38	Tarun Thaker	ms sxcel	1599
39	David Mukhopadhyay	ms sxcel	1599
40	Yasmin Palan	ms sxcel	1599
41	Radha Dutt	ms sxcel	1599
42	Kailash Harjo	pyspark	2499
43	Tarun Thaker	pyspark	2499
44	David Mukhopadhyay	pyspark	2499
45	Yasmin Palan	pyspark	2499
46	Shashank D'Alia	pyspark	2499
47	Radha Dutt	pyspark	2499
48	Tarun Thaker	sql	3499
49	Qabeel Raman	sql	3499

In [38]:
```
# 4. Plot bar chart for revenue/course
regs.merge(courses,on ='course_id').groupby('course_name')['price'].sum()
```

Out[38]:
```
course_name
data analysis       24995
machine learning    39996
ms sxcel             7995
pandas               4396
plotly               3495
power bi            11394
pyspark             14994
python              22491
sql                  6998
tableau             17493
Name: price, dtype: int64
```

In [41]:
```
regs.merge(courses,on ='course_id').groupby('course_name')['price'].sum().plot(kind='bar')
```

Out[41]:
```
<AxesSubplot:xlabel='course_name'>
```



intersect1d

Find the intersection of two arrays. Return the sorted, unique values that are in both of the input arrays.

In [45]:
```
# 5. find students who enrolled in both the months
common_students_id = np.intersect1d(may['student_id'],june['student_id'])
common_students_id
```

Out[45]:
```
array([ 1,  3,  7, 11, 16, 17, 18, 22, 23], dtype=int64)
```

In [47]:
```
students[students['student_id'].isin(common_students_id)]
```

Out[47]:

	student_id	name	partner
0	1	Kailash Harjo	23
2	3	Parveen Bhalla	3
6	7	Tarun Thaker	9
10	11	David Mukhopadhyay	20
15	16	Elias Dodiya	25
16	17	Yasmin Palan	7
17	18	Fardeen Mahabir	13
21	22	Yash Sethi	21
22	23	Chhavi Lachman	18

numpy.setdiff1d()

function find the set difference of two arrays and return the unique values in arr1 that are not in arr2.

In [52]:
```
# 6. find course that got no enrollment
# courses['course_id']
# regs['course_id']

course_id_list = np.setdiff1d(courses['course_id'], regs['course_id'])
courses[courses['course_id'].isin(course_id_list)]
```

Out[52]:

	course_id	course_name	price
10	11	Numpy	699
11	12	C++	1299

In [53]:
```
# 7. find students who did not enroll into any courses

student_id_list = np.setdiff1d(students['student_id'], regs['student_id'])
students[students['student_id'].isin(student_id_list)]
```

Out[53]:

	student_id	name	partner
3	4	Marlo Dugal	14
4	5	Kusum Bahri	6
5	6	Lakshmi Contractor	10
7	8	Radheshyam Dey	5
8	9	Nitika Chatterjee	4
9	10	Aayushman Sant	8
19	20	Hanuman Hegde	11
25	26	Nitish	28
26	27	Ankit	26
27	28	Rahul	17

In [55]:
```
students[students['student_id'].isin(student_id_list)].shape[0]
```

Out[55]: 10

In [56]:
```
# Percentage of students Enrolled

(10/28)*100
```

Out[56]: 35.714285714285715

Self Join

A self join is a regular join, but the table is joined with itself.

here, left_on = partner from outside students on left , right_on =student_id from iside students on right .

In [60]:
```
# 8. Print student name -> partner name for all enrolled students
# self join
students.merge(students,how ='inner',left_on = 'partner', right_on= 'student_id')[['name_x','name_y']]
```

Out[60]:

	name_x	name_y
0	Kailash Harjo	Chhavi Lachman
1	Esha Butala	Kailash Harjo
2	Parveen Bhalla	Parveen Bhalla
3	Marlo Dugal	Pranab Natarajan
4	Kusum Bahri	Lakshmi Contractor
5	Lakshmi Contractor	Aayushman Sant
6	Tarun Thaker	Nitika Chatterjee
7	Radheshyam Dey	Kusum Bahri
8	Nitika Chatterjee	Marlo Dugal
9	Aayushman Sant	Radheshyam Dey
10	David Mukhopadhyay	Hanuman Hegde
11	Radha Dutt	Qabeel Raman
12	Munni Varghese	Radhika Suri
13	Pranab Natarajan	Yash Sethi
14	Preet Sha	Elias Dodiya
15	Elias Dodiya	Shashank D'Alia
16	Yasmin Palan	Tarun Thaker
17	Fardeen Mahabir	Munni Varghese
18	Qabeel Raman	Radha Dutt
19	Hanuman Hegde	David Mukhopadhyay
20	Seema Kota	Preet Sha
21	Yash Sethi	Seema Kota
22	Chhavi Lachman	Fardeen Mahabir
23	Radhika Suri	Yasmin Palan
24	Rahul	Yasmin Palan
25	Shashank D'Alia	Esha Butala
26	Nitish	Rahul
27	Ankit	Nitish

In [70]:
```
# 9. find top 3 students who did most number enrollments
regs.merge(students, on='student_id').groupby(['student_id','name'])['name'].count().sort_values(ascending=False).head(3)
```

Out[70]:
```
student_id  name
23          Chhavi Lachman    6
7           Tarun Thaker      5
1           Kailash Harjo     4
Name: name, dtype: int64
```

In [81]:
```
# 10. find top 5 students who spent most amount of money on courses
regs.merge(students , on ='student_id').merge(courses, on= 'course_id').groupby(['student_id','name'])['price'].sum().sort_values
```

Out[81]:
```
student_id  name
23          Chhavi Lachman      22594
14          Pranab Natarajan    15096
19          Qabeel Raman        13498
7           Tarun Thaker        10595
24          Radhika Suri         9999
Name: price, dtype: int64
```

In [82]:
```
# Alternate syntax for merge
# students.merge(regs)

pd.merge(students,regs , how='inner', on= 'student_id')
```

Out[82]:

	student_id	name	partner	course_id
0	1	Kailash Harjo	23	1
1	1	Kailash Harjo	23	6
2	1	Kailash Harjo	23	10
3	1	Kailash Harjo	23	9
4	2	Esha Butala	1	5
5	3	Parveen Bhalla	3	3
6	3	Parveen Bhalla	3	5
7	7	Tarun Thaker	9	8
8	7	Tarun Thaker	9	10
9	7	Tarun Thaker	9	7
10	7	Tarun Thaker	9	2
11	7	Tarun Thaker	9	6
12	11	David Mukhopadhyay	20	7
13	11	David Mukhopadhyay	20	8
14	11	David Mukhopadhyay	20	10
15	12	Radha Dutt	19	10
16	12	Radha Dutt	19	1
17	12	Radha Dutt	19	5
18	12	Radha Dutt	19	7
19	13	Munni Varghese	24	3
20	14	Pranab Natarajan	22	9
21	14	Pranab Natarajan	22	6
22	14	Pranab Natarajan	22	4
23	14	Pranab Natarajan	22	1
24	15	Preet Sha	16	5
25	15	Preet Sha	16	1
26	16	Elias Dodiya	25	9
27	16	Elias Dodiya	25	5
28	16	Elias Dodiya	25	7
29	16	Elias Dodiya	25	3
30	17	Yasmin Palan	7	7
31	17	Yasmin Palan	7	10
32	18	Fardeen Mahabir	13	6
33	18	Fardeen Mahabir	13	1
34	18	Fardeen Mahabir	13	8
35	19	Qabeel Raman	12	4
36	19	Qabeel Raman	12	2
37	21	Seema Kota	15	1
38	22	Yash Sethi	21	3
39	22	Yash Sethi	21	5
40	22	Yash Sethi	21	6
41	23	Chhavi Lachman	18	1
42	23	Chhavi Lachman	18	4
43	23	Chhavi Lachman	18	3
44	23	Chhavi Lachman	18	6
45	23	Chhavi Lachman	18	9
46	23	Chhavi Lachman	18	5
47	24	Radhika Suri	17	4
48	25	Shashank D'Alia	2	1
49	25	Shashank D'Alia	2	10

In [87]:
```
# IPL Problems

# find top 3 stadiums with highest sixes/match ratio

matches
```

Out[87]:

	id	season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets	player_
0	1	2017	Hyderabad	2017-04-05	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Sunrisers Hyderabad	35	0	Yu
1	2	2017	Pune	2017-04-06	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	0	Rising Pune Supergiant	0	7	S
2	3	2017	Rajkot	2017-04-07	Gujarat Lions	Kolkata Knight Riders	Kolkata Knight Riders	field	normal	0	Kolkata Knight Riders	0	10	
3	4	2017	Indore	2017-04-08	Rising Pune Supergiant	Kings XI Punjab	Kings XI Punjab	field	normal	0	Kings XI Punjab	0	6	G
4	5	2017	Bangalore	2017-04-08	Royal Challengers Bangalore	Delhi Daredevils	Royal Challengers Bangalore	bat	normal	0	Royal Challengers Bangalore	15	0	K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
631	632	2016	Raipur	2016-05-22	Delhi Daredevils	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Royal Challengers Bangalore	0	6	
632	633	2016	Bangalore	2016-05-24	Gujarat Lions	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Royal Challengers Bangalore	0	4	AB
633	634	2016	Delhi	2016-05-25	Sunrisers Hyderabad	Kolkata Knight Riders	Kolkata Knight Riders	field	normal	0	Sunrisers Hyderabad	22	0	MC I
634	635	2016	Delhi	2016-05-27	Gujarat Lions	Sunrisers Hyderabad	Sunrisers Hyderabad	field	normal	0	Sunrisers Hyderabad	0	4	D
635	636	2016	Bangalore	2016-05-29	Sunrisers Hyderabad	Royal Challengers Bangalore	Sunrisers Hyderabad	bat	normal	0	Sunrisers Hyderabad	8	0	BC

636 rows × 18 columns

In [89]:
```
deliveries
```

Out[89]:

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over	...	bye_runs	legbye_runs	noball_runs	penalt
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	1	DA Warner	S Dhawan	TS Mills	0	...	0	0	0	
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	2	DA Warner	S Dhawan	TS Mills	0	...	0	0	0	
2	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	3	DA Warner	S Dhawan	TS Mills	0	...	0	0	0	
3	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	4	DA Warner	S Dhawan	TS Mills	0	...	0	0	0	
4	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	5	DA Warner	S Dhawan	TS Mills	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
179073	11415	2	Chennai Super Kings	Mumbai Indians	20	2	RA Jadeja	SR Watson	SL Malinga	0	...	0	0	0	
179074	11415	2	Chennai Super Kings	Mumbai Indians	20	3	SR Watson	RA Jadeja	SL Malinga	0	...	0	0	0	
179075	11415	2	Chennai Super Kings	Mumbai Indians	20	4	SR Watson	RA Jadeja	SL Malinga	0	...	0	0	0	
179076	11415	2	Chennai Super Kings	Mumbai Indians	20	5	SN Thakur	RA Jadeja	SL Malinga	0	...	0	0	0	
179077	11415	2	Chennai Super Kings	Mumbai Indians	20	6	SN Thakur	RA Jadeja	SL Malinga	0	...	0	0	0	

179078 rows × 21 columns

In [94]:
```
temp = pd.merge(deliveries,matches ,how ='inner',left_on='match_id',right_on='id')

temp.head(2)
```

Out[94]:

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over	...	result	dl_applied	winner	win_by_runs	win_
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	1	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35	
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	2	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35	

2 rows × 39 columns

In [101]:
```
six_df=temp[temp['batsman_runs']==6]
six_df.head(2)
```

Out[101]:

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over	...	result	dl_applied	winner	win_by_runs
10	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	2	4	DA Warner	S Dhawan	A Choudhary	0	...	normal	0	Sunrisers Hyderabad	35
47	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	8	4	MC Henriques	S Dhawan	TM Head	0	...	normal	0	Sunrisers Hyderabad	35

2 rows × 39 columns

In [105]:
```
#stadium --> sixes
number_six = six_df.groupby('venue')['venue'].count()
number_six.head()
```

Out[105]:
```
venue
Barabati Stadium              68
Brabourne Stadium            114
Buffalo Park                  27
De Beers Diamond Oval         34
Dr DY Patil Sports Academy   173
Name: venue, dtype: int64
```

In [108]:
```
# Number of matches
number_matches = matches['venue'].value_counts()
number_matches.head()
```

Out[108]:
```
M Chinnaswamy Stadium                      66
Eden Gardens                               61
Feroz Shah Kotla                           60
Wankhede Stadium                           57
Rajiv Gandhi International Stadium, Uppal  49
Name: venue, dtype: int64
```

In [112]:
```
(number_six/number_matches).sort_values(ascending=False).head()
```

Out[112]:
```
Holkar Cricket Stadium                               17.600000
M Chinnaswamy Stadium                                13.227273
Sharjah Cricket Stadium                              12.666667
Himachal Pradesh Cricket Association Stadium         12.000000
Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium  11.727273
Name: venue, dtype: float64
```

In [113]:
```
# find orange cap holder of all the seasons
```

Out[113]:

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over	...	result	dl_applied	winner	win_by_runs
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	1	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	2	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35
2	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	3	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35
3	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	4	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35
4	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	5	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
150455	636	2	Royal Challengers Bangalore	Sunrisers Hyderabad	20	2	Sachin Baby	CJ Jordan	B Kumar	0	...	normal	0	Sunrisers Hyderabad	8
150456	636	2	Royal Challengers Bangalore	Sunrisers Hyderabad	20	3	Sachin Baby	CJ Jordan	B Kumar	0	...	normal	0	Sunrisers Hyderabad	8
150457	636	2	Royal Challengers Bangalore	Sunrisers Hyderabad	20	4	Iqbal Abdulla	Sachin Baby	B Kumar	0	...	normal	0	Sunrisers Hyderabad	8
150458	636	2	Royal Challengers Bangalore	Sunrisers Hyderabad	20	5	Sachin Baby	Iqbal Abdulla	B Kumar	0	...	normal	0	Sunrisers Hyderabad	8
150459	636	2	Royal Challengers Bangalore	Sunrisers Hyderabad	20	6	Iqbal Abdulla	Sachin Baby	B Kumar	0	...	normal	0	Sunrisers Hyderabad	8

150460 rows × 39 columns

In [114]:
```
df = pd.merge(deliveries,matches ,how ='inner',left_on='match_id',right_on='id')

df.head(2)
```

Out[114]:

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over	...	result	dl_applied	winner	win_by_runs	win_
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	1	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35	
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	2	DA Warner	S Dhawan	TS Mills	0	...	normal	0	Sunrisers Hyderabad	35	

2 rows × 39 columns

In [117]:
```
df.groupby(['season','batsman'])['batsman_runs'].sum()
```

Out[117]:
```
season  batsman
2008    A Chopra              42
        A Kumble              13
        A Mishra              37
        A Mukund               0
        A Nehra                3
                              ...
2017    Washington Sundar      9
        YK Pathan            143
        YS Chahal             13
        Yuvraj Singh         252
        Z Khan                 4
Name: batsman_runs, Length: 1531, dtype: int64
```

In [120]:
```
df.groupby(['season','batsman'])['batsman_runs'].sum().reset_index().sort_values('batsman_runs',ascending=False)
```

Out[120]:

	season	batsman	batsman_runs
1383	2016	V Kohli	973
1278	2016	DA Warner	848
910	2013	MEK Hussey	733
684	2012	CH Gayle	733
852	2013	CH Gayle	720
...	...	...	...
1467	2017	MM Patel	0
658	2012	AC Blizzard	0
475	2011	AB Dinda	0
1394	2017	AD Nath	0
58	2008	L Balaji	0

1531 rows × 3 columns

In [123]:
```
'])['batsman_runs'].sum().reset_index().sort_values('batsman_runs',ascending=False).drop_duplicates(subset='season',keep='first')
```

Out[123]:

	season	batsman	batsman_runs
1383	2016	V Kohli	973
910	2013	MEK Hussey	733
684	2012	CH Gayle	733
1088	2014	RV Uthappa	660
1422	2017	DA Warner	641
446	2010	SR Tendulkar	618
115	2008	SE Marsh	616
502	2011	CH Gayle	608
229	2009	ML Hayden	572
1148	2015	DA Warner	562

In [124]:
```
oupby(['season','batsman'])['batsman_runs'].sum().reset_index().sort_values('batsman_runs',ascending=False).sort_values('season')
```

Out[124]:

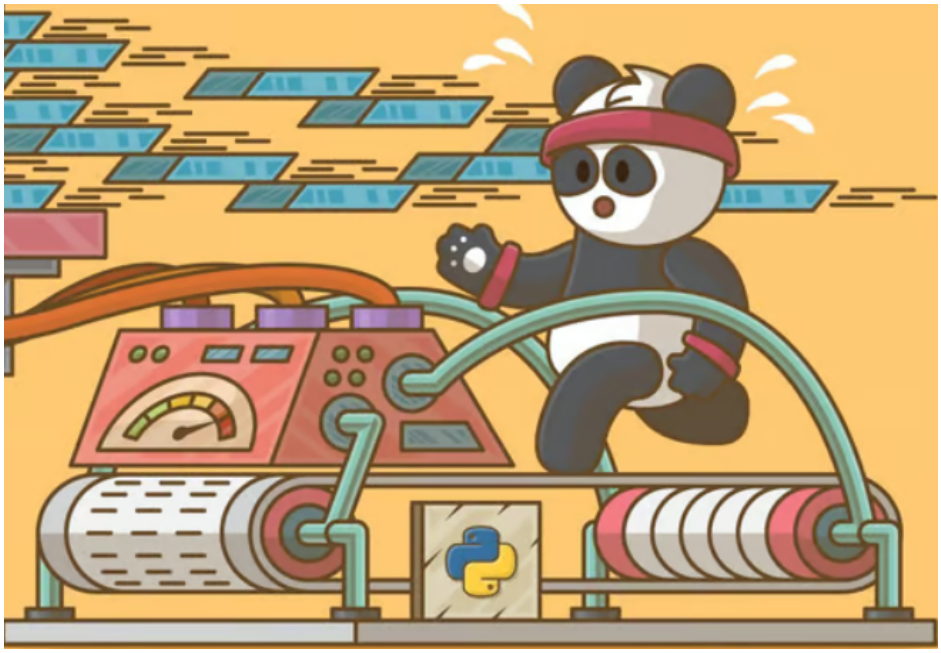
	season	batsman	batsman_runs
58	2008	L Balaji	0
45	2008	I Sharma	11
12	2008	AM Nayar	206
31	2008	DNT Zoysa	11
67	2008	M Ntini	11
...	...	...	...
1424	2017	DL Chahar	14
1515	2017	Swapnil Singh	12
1516	2017	TA Boult	5
1470	2017	MP Stoinis	17
1400	2017	AR Bawne	12

1531 rows × 3 columns

In [ ]:

What is MultiIndex in Pandas?

In pandas, a multi-index, also known as a **hierarchical index**, is a way to represent two or more dimensions of data in a single index. This is useful when you have data that can be grouped or categorized by more than one variable.



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

Series is 1D and DataFrames are 2D objects

- But why?
- And what exactly is index?

In [2]:
```
# can we have multiple index? Let's try
index_val = [('cse',2019),('cse',2020),('cse',2021),('cse',2022),('ece',2019),('ece',2020),('ece',2021),('ece',2022)]
data = pd.Series([1,2,3,4,5,6,7,8],index=index_val)
data
```

Out[2]:
```
(cse, 2019)    1
(cse, 2020)    2
(cse, 2021)    3
(cse, 2022)    4
(ece, 2019)    5
(ece, 2020)    6
(ece, 2021)    7
(ece, 2022)    8
dtype: int64
```

In [3]:
```
# The problem?
#data['cse']
```

The solution : multiindex series-2D(also known as Hierarchical Indexing)

multiple index levels within a single index

In [4]:
```
# how to create multiindex object
# 1. pd.MultiIndex.from_tuples()
index_val = [('cse',2019),('cse',2020),('cse',2021),('cse',2022),('ece',2019),('ece',2020),('ece',2021),('ece',2022)]
multiindex = pd.MultiIndex.from_tuples(index_val)
multiindex.levels[0]
```

Out[4]:
```
Index(['cse', 'ece'], dtype='object')
```

In [5]:
```
# 2. pd.MultiIndex.from_product()
pd.MultiIndex.from_product([['cse','ece'],[2019,2020,2021,2022]])
```

Out[5]:
```
MultiIndex([('cse', 2019),
            ('cse', 2020),
            ('cse', 2021),
            ('cse', 2022),
            ('ece', 2019),
            ('ece', 2020),
            ('ece', 2021),
            ('ece', 2022)],
           )
```

In [6]:
```
# creating a series with multiindex object
sample = pd.Series([1,2,3,4,5,6,7,8],index=multiindex)
sample
```

Out[6]:
```
cse  2019    1
     2020    2
     2021    3
     2022    4
ece  2019    5
     2020    6
     2021    7
     2022    8
dtype: int64
```

In [7]:
```
# how to fetch items from such a series
sample[('cse',2022)]
```

Out[7]:
```
4
```

In [8]:
```
sample['cse']
```

Out[8]:
```
2019    1
2020    2
2021    3
2022    4
dtype: int64
```

unstack

reshape the given Pandas DataFrame by transposing specified row level to column level

In [9]:
```
temp = sample.unstack()
temp
```

Out[9]:

	2019	2020	2021	2022
cse	1	2	3	4
ece	5	6	7	8

stack

reshapes the given DataFrame by converting the column label to a row index.

In [10]:
```
temp.stack()
```

Out[10]:
```
cse  2019    1
     2020    2
     2021    3
     2022    4
ece  2019    5
     2020    6
     2021    7
     2022    8
dtype: int64
```

so why we should study Multi Index

because we can convert any dataframe dimension, including 3D, 4D, 10D, and 20D, to 1Dimension (Series) and 2Dimension (Dataframes).

In [11]:
```
# multi index dataframes
branch_df1 = pd.DataFrame(
    [
        [1,2],
        [3,4],
        [5,6],
        [7,8],
        [9,10],
        [11,12],
        [13,14],
        [15,16],
    ],
    index = multiindex,
    columns = ['avg_package','students']
)

branch_df1
```

Out[11]:

		avg_package	students
cse	**2019**	1	2
	2020	3	4
	2021	5	6
	2022	7	8
ece	**2019**	9	10
	2020	11	12
	2021	13	14
	2022	15	16

In [12]:
```
branch_df1.loc['cse']
```

Out[12]:

	avg_package	students
2019	1	2
2020	3	4
2021	5	6
2022	7	8

In [13]:
```
branch_df1['avg_package']
```

Out[13]:
```
cse  2019     1
     2020     3
     2021     5
     2022     7
ece  2019     9
     2020    11
     2021    13
     2022    15
Name: avg_package, dtype: int64
```

In [14]:
```
branch_df1['students']
```

Out[14]:
```
cse  2019     2
     2020     4
     2021     6
     2022     8
ece  2019    10
     2020    12
     2021    14
     2022    16
Name: students, dtype: int64
```

In [15]:
```
branch_df1.loc['ece']
```

Out[15]:

	avg_package	students
2019	9	10
2020	11	12
2021	13	14
2022	15	16

multiindex df from columns perspective

In [16]:
```
branch_df2 = pd.DataFrame(
    [
        [1,2,0,0],
        [3,4,0,0],
        [5,6,0,0],
        [7,8,0,0],
    ],
    index = [2019,2020,2021,2022],
    columns = pd.MultiIndex.from_product([['delhi','mumbai'],['avg_package','students']])
)

branch_df2
```

Out[16]:

	delhi		mumbai	
	avg_package	students	avg_package	students
2019	1	2	0	0
2020	3	4	0	0
2021	5	6	0	0
2022	7	8	0	0

In [17]:
```
branch_df2['delhi']
```

Out[17]:

	avg_package	students
2019	1	2
2020	3	4
2021	5	6
2022	7	8

In [18]:
```
branch_df2.loc[2019]
```

Out[18]:
```
delhi   avg_package    1
        students       2
mumbai  avg_package    0
        students       0
Name: 2019, dtype: int64
```

In [19]:
```
branch_df2.iloc[1]
```

Out[19]:
```
delhi   avg_package    3
        students       4
mumbai  avg_package    0
        students       0
Name: 2020, dtype: int64
```

Multiindex df in terms of both cols and index

In [20]:
```
branch_df3 = pd.DataFrame(
    [
        [1,2,0,0],
        [3,4,0,0],
        [5,6,0,0],
        [7,8,0,0],
        [9,10,0,0],
        [11,12,0,0],
        [13,14,0,0],
        [15,16,0,0],
    ],
    index = multiindex,
    columns = pd.MultiIndex.from_product([['delhi','mumbai'],['avg_package','students']])
)

branch_df3

#here index= multiindex is a name , already we have stored data of ece and cse in above
```

Out[20]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

Stacking and Unstacking

In [21]:
```
branch_df1
```

Out[21]:

		avg_package	students
cse	2019	1	2
	2020	3	4
	2021	5	6
	2022	7	8
ece	2019	9	10
	2020	11	12
	2021	13	14
	2022	15	16

In [22]:
```
 # After applying Unstack
branch_df1.unstack()
```

Out[22]:

	avg_package				students			
	2019	2020	2021	2022	2019	2020	2021	2022
cse	1	3	5	7	2	4	6	8
ece	9	11	13	15	10	12	14	16

In [23]:
```
branch_df1.unstack().unstack()
```

Out[23]:
```
avg_package  2019  cse     1
                   ece     9
             2020  cse     3
                   ece    11
             2021  cse     5
                   ece    13
             2022  cse     7
                   ece    15
students     2019  cse     2
                   ece    10
             2020  cse     4
                   ece    12
             2021  cse     6
                   ece    14
             2022  cse     8
                   ece    16
dtype: int64
```

The stack() method

It can be used to move the columns to the index. This means that the columns will become the rows, and the rows will become the columns.
The stack method can be used to move the columns to the index

In [24]:
```
 # After applying Unstack + stack
branch_df1.unstack().stack()
```

Out[24]:

		avg_package	students
cse	**2019**	1	2
	2020	3	4
	2021	5	6
	2022	7	8
ece	**2019**	9	10
	2020	11	12
	2021	13	14
	2022	15	16

In [25]:
```
# applying multiple stack

branch_df1.unstack().stack().stack()
```

Out[25]:
```
cse  2019  avg_package     1
           students        2
     2020  avg_package     3
           students        4
     2021  avg_package     5
           students        6
     2022  avg_package     7
           students        8
ece  2019  avg_package     9
           students       10
     2020  avg_package    11
           students       12
     2021  avg_package    13
           students       14
     2022  avg_package    15
           students       16
dtype: int64
```

In [26]:
```
# Example : 2
branch_df2
```

Out[26]:

	delhi		mumbai	
	avg_package	students	avg_package	students
2019	1	2	0	0
2020	3	4	0	0
2021	5	6	0	0
2022	7	8	0	0

The Unstack()

It is method can be used to move the index to the columns. This means that the index will become the rows, and the rows will become the columns.

The unstack method can be used to move the index to the columns

In [27]:
```
branch_df2.unstack()
```

Out[27]:
```
delhi   avg_package  2019    1
                     2020    3
                     2021    5
                     2022    7
        students     2019    2
                     2020    4
                     2021    6
                     2022    8
mumbai  avg_package  2019    0
                     2020    0
                     2021    0
                     2022    0
        students     2019    0
                     2020    0
                     2021    0
                     2022    0
dtype: int64
```

In [28]:
```
branch_df2.stack()
```

Out[28]:

		delhi	mumbai
2019	**avg_package**	1	0
	students	2	0
2020	**avg_package**	3	0
	students	4	0
2021	**avg_package**	5	0
	students	6	0
2022	**avg_package**	7	0
	students	8	0

In [29]:
```
branch_df2.stack().stack()
```

Out[29]:
```
2019  avg_package  delhi     1
                   mumbai    0
      students     delhi     2
                   mumbai    0
2020  avg_package  delhi     3
                   mumbai    0
      students     delhi     4
                   mumbai    0
2021  avg_package  delhi     5
                   mumbai    0
      students     delhi     6
                   mumbai    0
2022  avg_package  delhi     7
                   mumbai    0
      students     delhi     8
                   mumbai    0
dtype: int64
```

In [30]:
```
# Working on 4D data
branch_df3
```

Out[30]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

In [31]:
```
branch_df3.stack()
```

Out[31]:

			delhi	mumbai
cse	2019	avg_package	1	0
		students	2	0
	2020	avg_package	3	0
		students	4	0
	2021	avg_package	5	0
		students	6	0
	2022	avg_package	7	0
		students	8	0
ece	2019	avg_package	9	0
		students	10	0
	2020	avg_package	11	0
		students	12	0
	2021	avg_package	13	0
		students	14	0
	2022	avg_package	15	0
		students	16	0

In [32]:
```
branch_df3.stack().stack()
```

Out[32]:
```
cse  2019  avg_package  delhi      1
                        mumbai     0
           students     delhi      2
                        mumbai     0
      2020  avg_package  delhi      3
                        mumbai     0
           students     delhi      4
                        mumbai     0
      2021  avg_package  delhi      5
                        mumbai     0
           students     delhi      6
                        mumbai     0
      2022  avg_package  delhi      7
                        mumbai     0
           students     delhi      8
                        mumbai     0
ece  2019  avg_package  delhi      9
                        mumbai     0
           students     delhi     10
                        mumbai     0
      2020  avg_package  delhi     11
                        mumbai     0
           students     delhi     12
                        mumbai     0
      2021  avg_package  delhi     13
                        mumbai     0
           students     delhi     14
                        mumbai     0
      2022  avg_package  delhi     15
                        mumbai     0
           students     delhi     16
                        mumbai     0
dtype: int64
```

In [33]:
```
# Unstacking on 4D data
branch_df3.unstack()
```

Out[33]:

	delhi								mumbai							
	avg_package				students				avg_package				students			
	2019	2020	2021	2022	2019	2020	2021	2022	2019	2020	2021	2022	2019	2020	2021	2022
cse	1	3	5	7	2	4	6	8	0	0	0	0	0	0	0	0
ece	9	11	13	15	10	12	14	16	0	0	0	0	0	0	0	0

In [34]:
```
branch_df3.unstack().unstack()
```

Out[34]:
```
delhi   avg_package  2019  cse     1
                             ece     9
                      2020  cse     3
                             ece    11
                      2021  cse     5
                             ece    13
                      2022  cse     7
                             ece    15
        students      2019  cse     2
                             ece    10
                      2020  cse     4
                             ece    12
                      2021  cse     6
                             ece    14
                      2022  cse     8
                             ece    16
mumbai  avg_package  2019  cse     0
                             ece     0
                      2020  cse     0
                             ece     0
                      2021  cse     0
                             ece     0
                      2022  cse     0
                             ece     0
        students      2019  cse     0
                             ece     0
                      2020  cse     0
                             ece     0
                      2021  cse     0
                             ece     0
                      2022  cse     0
                             ece     0
dtype: int64
```

Working with multiindex dataframes

In [35]:
```
# Multi index dataframes works same as normal dataframes
branch_df3
```

Out[35]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

In [36]:
```
# head and tail
branch_df3.head()
```

Out[36]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0

In [37]:
```
# shape
branch_df3.shape
```

Out[37]:
```
(8, 4)
```

In [38]:
```
# info
branch_df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 8 entries, ('cse', 2019) to ('ece', 2022)
Data columns (total 4 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   (delhi, avg_package)  8 non-null      int64
 1   (delhi, students)     8 non-null      int64
 2   (mumbai, avg_package) 8 non-null      int64
 3   (mumbai, students)    8 non-null      int64
dtypes: int64(4)
memory usage: 932.0+ bytes
```

In [39]:
```
# duplicated -> isnull
branch_df3.duplicated()
```

Out[39]:
```
cse  2019    False
     2020    False
     2021    False
     2022    False
ece  2019    False
     2020    False
     2021    False
     2022    False
dtype: bool
```

In [40]:
```
branch_df3.isnull()
```

Out[40]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	False	False	False	False
	2020	False	False	False	False
	2021	False	False	False	False
	2022	False	False	False	False
ece	2019	False	False	False	False
	2020	False	False	False	False
	2021	False	False	False	False
	2022	False	False	False	False

In [41]:
```
# Extracting rows single

branch_df3.loc[('cse',2022)]
```

Out[41]:
```
delhi   avg_package    7
        students       8
mumbai  avg_package    0
        students       0
Name: (cse, 2022), dtype: int64
```

In [42]:
```
# Extracting multiple rows

branch_df3.loc[('cse',2019):('ece',2020):2]
```

Out[42]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2021	5	6	0	0
ece	2019	9	10	0	0

In [43]:
```
# Using iloc
branch_df3.iloc[0:5:2]
```

Out[43]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2021	5	6	0	0
ece	2019	9	10	0	0

In [44]:
```
# Extracting single columns
branch_df3['delhi']['students']
```

Out[44]:
```
cse  2019     2
     2020     4
     2021     6
     2022     8
ece  2019    10
     2020    12
     2021    14
     2022    16
Name: students, dtype: int64
```

In [45]:
```
# we want to extract delhi - students , mumbai - avg_package
branch_df3
```

Out[45]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

In [46]:
```
#here [:] all rows ,
#columns : delhi=avg_package[0],students[1],mumbai=avg_package[2],students[3]

branch_df3.iloc[:,1:3]
```

Out[46]:

		delhi	mumbai
		students	avg_package
cse	2019	2	0
	2020	4	0
	2021	6	0
	2022	8	0
ece	2019	10	0
	2020	12	0
	2021	14	0
	2022	16	0

In [47]:
```
# Extracting both rows and columns
branch_df3.iloc[[0,4],[1,2]]
```

Out[47]:

		delhi	mumbai
		students	avg_package
cse	2019	2	0
ece	2019	10	0

In [48]:
```
# sort index
# both -> descending -> diff order
# based on one level

branch_df3
```

Out[48]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

In [49]:
```
branch_df3.sort_index(ascending=False)
```

Out[49]:

		delhi		mumbai	
		avg_package	students	avg_package	students
ece	2022	15	16	0	0
	2021	13	14	0	0
	2020	11	12	0	0
	2019	9	10	0	0
cse	2022	7	8	0	0
	2021	5	6	0	0
	2020	3	4	0	0
	2019	1	2	0	0

In [50]:
```
# if we want year in descending order
branch_df3.sort_index(ascending=[False ,True])
```

Out[50]:

		delhi		mumbai	
		avg_package	students	avg_package	students
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0

In [51]:
```
# multiindex dataframe(col) -> transpose
branch_df3.transpose()
```

Out[51]:

		cse				ece			
		2019	2020	2021	2022	2019	2020	2021	2022
delhi	avg_package	1	3	5	7	9	11	13	15
	students	2	4	6	8	10	12	14	16
mumbai	avg_package	0	0	0	0	0	0	0	0
	students	0	0	0	0	0	0	0	0

In [52]:
```
# swaplevel
branch_df3
```

Out[52]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

In [53]:
```
# On rows
branch_df3.swaplevel()
```

Out[53]:

		delhi		mumbai	
		avg_package	students	avg_package	students
2019	cse	1	2	0	0
2020	cse	3	4	0	0
2021	cse	5	6	0	0
2022	cse	7	8	0	0
2019	ece	9	10	0	0
2020	ece	11	12	0	0
2021	ece	13	14	0	0
2022	ece	15	16	0	0

In [54]:
```
# on columns

branch_df3.swaplevel(axis=1)
```

Out[54]:

		avg_package	students	avg_package	students
		delhi	delhi	mumbai	mumbai
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

Long(Tall) Vs Wide data

"Long" format

country	year	metric
x	1960	10
x	1970	13
x	2010	15
y	1960	20
y	1970	23
y	2010	25
z	1960	30
z	1970	33
z	2010	35

"Wide" format

country	yr1960	yr1970	yr2010
x	10	13	15
y	20	23	25
z	30	33	35

Wide format is where we have a single row for every data point with multiple columns to hold the values of various attributes.

Long format is where, for each data point we have as many rows as the number of attributes and each row contains the value of a particular attribute for a given data point.

Melt -- Converting wide data to long Data.

In [55]:
```
# melt -> simple example branch
# wide to long
pd.DataFrame({'cse':[120]})
```

Out[55]:

	cse
0	120

In [56]:
```
pd.DataFrame({'cse':[120]}).melt()
```

Out[56]:

	variable	value
0	cse	120

In [57]:
```
# melt -> branch with year
pd.DataFrame({'cse':[120],'ece':[100],'mech':[50]}).melt()
```

Out[57]:

	variable	value
0	cse	120
1	ece	100
2	mech	50

In [58]:
```
# we can name the varibale and value
pd.DataFrame({'cse':[120],'ece':[100],'mech':[50]}).melt(var_name='branch',value_name='num_students')
```

Out[58]:

	branch	num_students
0	cse	120
1	ece	100
2	mech	50

In [59]:
```
pd.DataFrame(
    {
        'branch':['cse','ece','mech'],
        '2020':[100,150,60],
        '2021':[120,130,80],
        '2022':[150,140,70]
    }
)
```

Out[59]:

	branch	2020	2021	2022
0	cse	100	120	150
1	ece	150	130	140
2	mech	60	80	70

In [60]:
```
pd.DataFrame(
    {
        'branch':['cse','ece','mech'],
        '2020':[100,150,60],
        '2021':[120,130,80],
        '2022':[150,140,70]
    }
).melt()
```

Out[60]:

	variable	value
0	branch	cse
1	branch	ece
2	branch	mech
3	2020	100
4	2020	150
5	2020	60
6	2021	120
7	2021	130
8	2021	80
9	2022	150
10	2022	140
11	2022	70

In [61]:
```
# dont include 'branch' to rows
pd.DataFrame(
    {
        'branch':['cse','ece','mech'],
        '2020':[100,150,60],
        '2021':[120,130,80],
        '2022':[150,140,70]
    }
).melt(id_vars=['branch'])
```

Out[61]:

	branch	variable	value
0	cse	2020	100
1	ece	2020	150
2	mech	2020	60
3	cse	2021	120
4	ece	2021	130
5	mech	2021	80
6	cse	2022	150
7	ece	2022	140
8	mech	2022	70

the **melt()** method is used to reshape a DataFrame from wide to long format. This means that the columns of the DataFrame are converted into rows, and the values in the columns are converted into columns.

In [62]:
```
# adding variable and value names.
pd.DataFrame(
    {
        'branch':['cse','ece','mech'],
        '2020':[100,150,60],
        '2021':[120,130,80],
        '2022':[150,140,70]
    }
).melt(id_vars=['branch'],var_name='year',value_name='students')
```

Out[62]:

	branch	year	students
0	cse	2020	100
1	ece	2020	150
2	mech	2020	60
3	cse	2021	120
4	ece	2021	130
5	mech	2021	80
6	cse	2022	150
7	ece	2022	140
8	mech	2022	70

In [63]:
```
# melt ---> Real world examples.

deaths =pd.read_csv("time_series_covid19_deaths_global.csv")
confirm = pd.read_csv("time_series_covid19_confirmed_global.csv")
```

In [64]:
```
deaths.head(2)
```

Out[64]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	...	12/24/22	12/25/22	12/26/22	12/27/22
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	7845	7846	7846	7846
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	3595	3595	3595	3595

2 rows × 1081 columns

In [65]:
```
deaths.shape
```

Out[65]: (289, 1081)

In [66]:
```
deaths =deaths.melt(id_vars=['Province/State','Country/Region','Lat','Long'],var_name='date',value_name='no. of deaths'
```

In [67]:
```
# After converting columns into rows,
# which is converting wide format to long format using 'melt'
deaths.shape
```

Out[67]: (311253, 6)

In [75]:
```
deaths.head()
```

Out[75]:

	Province/State	Country/Region	Lat	Long	date	no. of deaths
0	NaN	Afghanistan	33.93911	67.709953	1/22/20	0
1	NaN	Albania	41.15330	20.168300	1/22/20	0
2	NaN	Algeria	28.03390	1.659600	1/22/20	0
3	NaN	Andorra	42.50630	1.521800	1/22/20	0
4	NaN	Angola	-11.20270	17.873900	1/22/20	0

In [68]:
```
confirm.head(2)
```

Out[68]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	...	12/24/22	12/25/22	12/26/22	12/27/22
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	...	207310	207399	207438	207460
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	...	333749	333749	333751	333751

2 rows × 1081 columns

In [69]:
```
confirm.shape
```

Out[69]: (289, 1081)

In [73]:
```
confirm =confirm.melt(id_vars=['Province/State','Country/Region','Lat','Long'],var_name='date',value_name='no. of confi
```

In [76]:
```
confirm.head()
```

Out[76]:

	Province/State	Country/Region	Lat	Long	date	no. of confirmed
0	NaN	Afghanistan	33.93911	67.709953	1/22/20	0
1	NaN	Albania	41.15330	20.168300	1/22/20	0
2	NaN	Algeria	28.03390	1.659600	1/22/20	0
3	NaN	Andorra	42.50630	1.521800	1/22/20	0
4	NaN	Angola	-11.20270	17.873900	1/22/20	0

In [74]:
```
# After converting columns into rows,
# which is converting wide format to long format using 'melt'
confirm.shape
```

Out[74]: (311253, 6)

In [77]:
```
# Now merge both data frames as per desire
confirm.merge(deaths, on =['Province/State','Country/Region','Lat','Long','date'])
```

Out[77]:

	Province/State	Country/Region	Lat	Long	date	no. of confirmed	no. of deaths
0	NaN	Afghanistan	33.939110	67.709953	1/22/20	0	0
1	NaN	Albania	41.153300	20.168300	1/22/20	0	0
2	NaN	Algeria	28.033900	1.659600	1/22/20	0	0
3	NaN	Andorra	42.506300	1.521800	1/22/20	0	0
4	NaN	Angola	-11.202700	17.873900	1/22/20	0	0
...	...	...	...	...	...	...	...
311248	NaN	West Bank and Gaza	31.952200	35.233200	1/2/23	703228	5708
311249	NaN	Winter Olympics 2022	39.904200	116.407400	1/2/23	535	0
311250	NaN	Yemen	15.552727	48.516388	1/2/23	11945	2159
311251	NaN	Zambia	-13.133897	27.849332	1/2/23	334661	4024
311252	NaN	Zimbabwe	-19.015438	29.154857	1/2/23	259981	5637

311253 rows × 7 columns

In [80]:
```
esired columns
eaths, on =['Province/State','Country/Region','Lat','Long','date'])[['Country/Region','date','no. of confirmed','no. of
```

Out[80]:

	Country/Region	date	no. of confirmed	no. of deaths
0	Afghanistan	1/22/20	0	0
1	Albania	1/22/20	0	0
2	Algeria	1/22/20	0	0
3	Andorra	1/22/20	0	0
4	Angola	1/22/20	0	0
...	...	...	...	...
311248	West Bank and Gaza	1/2/23	703228	5708
311249	Winter Olympics 2022	1/2/23	535	0
311250	Yemen	1/2/23	11945	2159
311251	Zambia	1/2/23	334661	4024
311252	Zimbabwe	1/2/23	259981	5637

311253 rows × 4 columns

Pivot table -- Converting Long data to wide data.

the **Pivot table** takes simple column wise data as input, and groups as the entire Into 2 dimensional table that provides a multi dimensional summarization of the data.

Pivot table generally used on categorical data

In [81]:
```
import seaborn as sns
```

In [83]:
```
df = sns.load_dataset('tips')
df.head()
```

Out[83]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [85]:
```
# On gender basis average total bill
df.groupby('sex')['total_bill'].mean()
```

Out[85]:
```
sex
Male      20.744076
Female    18.056897
Name: total_bill, dtype: float64
```

In [88]:
```
# On gender basis. Who smokes more? On average.
df.groupby(['sex','smoker'])['total_bill'].mean().unstack()
```

Out[88]:

smoker	Yes	No
sex		
Male	22.284500	19.791237
Female	17.977879	18.105185

In [89]:
```
# Using Pivot table method

df.pivot_table(index ='sex',columns='smoker',values = 'total_bill')
```

Out[89]:

smoker	Yes	No
sex		
Male	22.284500	19.791237
Female	17.977879	18.105185

In [90]:
```
# Aggregate function.
# Print, the total amount smokers of the bill, Not mean Or average,
df.pivot_table(index ='sex',columns='smoker',values = 'total_bill',aggfunc='sum')
```

Out[90]:

smoker	Yes	No
sex		
Male	1337.07	1919.75
Female	593.27	977.68

In [91]:
```
# count of people
df.pivot_table(index ='sex',columns='smoker',values = 'total_bill',aggfunc='count')
```

Out[91]:

smoker	Yes	No
sex		
Male	60	97
Female	33	54

In [92]:
```
# standard deviation

df.pivot_table(index ='sex',columns='smoker',values = 'total_bill',aggfunc='std')
```

Out[92]:

smoker	Yes	No
sex		
Male	9.911845	8.726566
Female	9.189751	7.286455

In [93]:
```
# All columns together --- gives average

df.pivot_table(index='sex',columns='smoker')
```

Out[93]:

	size		tip		total_bill	
smoker	Yes	No	Yes	No	Yes	No
sex						
Male	2.500000	2.711340	3.051167	3.113402	22.284500	19.791237
Female	2.242424	2.592593	2.931515	2.773519	17.977879	18.105185

In [95]:
```
# single column
df.pivot_table(index='sex',columns='smoker')['tip']
```

Out[95]:

smoker	Yes	No
sex		
Male	3.051167	3.113402
Female	2.931515	2.773519

In [96]:
```
df.pivot_table(index='sex',columns='smoker')['size']
```

Out[96]:

smoker	Yes	No
sex		
Male	2.500000	2.711340
Female	2.242424	2.592593

In [98]:
```
# Multi dimensional -5D
df.head(2)
```

Out[98]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3

In [100]:
```
# 5D - 5 Dimensional data
df.pivot_table(index=['sex','smoker'],columns=['day','time'],values='total_bill')
```

Out[100]:

day		Thur		Fri		Sat	Sun
time		Lunch	Dinner	Lunch	Dinner	Dinner	Dinner
sex	smoker						
Male	Yes	19.171000	NaN	11.386667	25.892	21.837778	26.141333
	No	18.486500	NaN	NaN	17.475	19.929063	20.403256
Female	Yes	19.218571	NaN	13.260000	12.200	20.266667	16.540000
	No	15.899167	18.78	15.980000	22.750	19.003846	20.824286

In [102]:
```
df.pivot_table(index=['sex','smoker'],columns=['day','time'])
```

Out[102]:

	tip										total_bill				
	Fri		Sat	Sun	Thur		Fri		Sat	Sun	Thur		Fri		Sat
Dinner	Lunch	Dinner	Dinner	Dinner	Lunch	Dinner	Lunch	Dinner	Dinner	Dinner	Lunch	Dinner	Lunch	Dinner	Dinner
NaN	1.666667	2.4	2.629630	2.600000	3.058000	NaN	1.90	3.246	2.879259	3.521333	19.171000	NaN	11.386667	25.892	21.837778
NaN	NaN	2.0	2.656250	2.883721	2.941500	NaN	NaN	2.500	3.256563	3.115349	18.486500	NaN	NaN	17.475	19.929063
NaN	2.000000	2.0	2.200000	2.500000	2.990000	NaN	2.66	2.700	2.868667	3.500000	19.218571	NaN	13.260000	12.200	20.266667
2.0	3.000000	2.0	2.307692	3.071429	2.437083	3.0	3.00	3.250	2.724615	3.329286	15.899167	18.78	15.980000	22.750	19.003846

In [103]:
```
df.pivot_table(index=['sex','smoker'],columns=['day','time'],aggfunc={'size':'mean','tip':'max','total_bill':'sum'})
```

Out[103]:

		size						tip						total_bill			
	day	Thur		Fri		Sat	Sun	Thur		Fri		Sat	Sun	Thur		Fri	
	time	Lunch	Dinner	Lunch	Dinner	Dinner	Dinner	Lunch	Dinner	Lunch	Dinner	Dinner	Dinner	Lunch	Dinner	Lunch	Di
sex	smoker																
Male	Yes	2.300000	NaN	1.666667	2.4	2.629630	2.600000	5.00	NaN	2.20	4.73	10.00	6.5	191.71	0.00	34.16	12
	No	2.500000	NaN	NaN	2.0	2.656250	2.883721	6.70	NaN	NaN	3.50	9.00	6.0	369.73	0.00	0.00	3
Female	Yes	2.428571	NaN	2.000000	2.0	2.200000	2.500000	5.00	NaN	3.48	4.30	6.50	4.0	134.53	0.00	39.78	4
	No	2.500000	2.0	3.000000	2.0	2.307692	3.071429	5.17	3.0	3.00	3.25	4.67	5.2	381.58	18.78	15.98	2

In [106]:
```
# Margins.
df.pivot_table(index='sex',columns= 'smoker',values ='total_bill',aggfunc='sum',margins=True)
```

Out[106]:

smoker	Yes	No	All
sex			
Male	1337.07	1919.75	3256.82
Female	593.27	977.68	1570.95
All	1930.34	2897.43	4827.77

In [108]:
```
# Plotting Graphs.
expense= pd.read_csv("expense_data.csv")
```

In [110]:
```
expense.head(2)
```

Out[110]:

	Date	Account	Category	Subcategory	Note	INR	Income/Expense	Note.1	Amount	Currency	Account.1
0	3/2/2022 10:11	CUB - online payment	Food	NaN	Brownie	50.0	Expense	NaN	50.0	INR	50.0
1	3/2/2022 10:11	CUB - online payment	Other	NaN	To lended people	300.0	Expense	NaN	300.0	INR	300.0

In [115]:
```
# Categories
expense['Category'].value_counts()
```

Out[115]:
```
Food                156
Other                60
Transportation       31
Apparel               7
Household             6
Allowance             6
Social Life           5
Education             1
Salary                1
Self-development      1
Beauty                1
Gift                  1
Petty cash            1
Name: Category, dtype: int64
```

In [117]:
```
expense.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 277 entries, 0 to 276
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Date            277 non-null    object 
 1   Account         277 non-null    object 
 2   Category        277 non-null    object 
 3   Subcategory     0 non-null      float64
 4   Note            273 non-null    object 
 5   INR             277 non-null    float64
 6   Income/Expense  277 non-null    object 
 7   Note.1          0 non-null      float64
 8   Amount          277 non-null    float64
 9   Currency        277 non-null    object 
 10  Account.1       277 non-null    float64
dtypes: float64(5), object(6)
memory usage: 23.9+ KB
```

In [119]:
```
# Converting integer to daytime format
expense['Date'] = pd.to_datetime(expense['Date'])
```

In [121]:
```
expense.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 277 entries, 0 to 276
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype         
---  ------          --------------  -----         
 0   Date            277 non-null    datetime64[ns]
 1   Account         277 non-null    object        
 2   Category        277 non-null    object        
 3   Subcategory     0 non-null      float64       
 4   Note            273 non-null    object        
 5   INR             277 non-null    float64       
 6   Income/Expense  277 non-null    object        
 7   Note.1          0 non-null      float64       
 8   Amount          277 non-null    float64       
 9   Currency        277 non-null    object        
 10  Account.1       277 non-null    float64       
dtypes: datetime64[ns](1), float64(5), object(5)
memory usage: 23.9+ KB
```

In [123]:
```
# Extracting month from the date column

expense['Date'].dt.month_name()
```

Out[123]:
```
0         March
1         March
2         March
3         March
4         March
         ...
272    November
273    November
274    November
275    November
276    November
Name: Date, Length: 277, dtype: object
```

In [124]:
```
expense['month']= expense['Date'].dt.month_name()
```

In [125]:
```
expense.head(2)
```

Out[125]:

	Date	Account	Category	Subcategory	Note	INR	Income/Expense	Note.1	Amount	Currency	Account.1	month
0	2022-03-02 10:11:00	CUB - online payment	Food	NaN	Brownie	50.0	Expense	NaN	50.0	INR	50.0	March
1	2022-03-02 10:11:00	CUB - online payment	Other	NaN	To lended people	300.0	Expense	NaN	300.0	INR	300.0	March

In [126]:
```
# Using pivot table
expense.pivot_table(index ='month', columns='Category', values ='INR', aggfunc='sum')
```

Out[126]:

Category	Allowance	Apparel	Beauty	Education	Food	Gift	Household	Other	Petty cash	Salary	Self-development	Social Life	Transportation
month													
December	11000.0	2590.0	196.0	NaN	6440.72	NaN	4800.0	1790.0	NaN	NaN	400.0	513.72	914.0
February	NaN	798.0	NaN	NaN	5579.85	NaN	2808.0	20000.0	NaN	NaN	NaN	1800.00	5078.8
January	1000.0	NaN	NaN	1400.0	9112.51	NaN	4580.0	13178.0	NaN	8000.0	NaN	200.00	2850.0
March	NaN	NaN	NaN	NaN	195.00	NaN	NaN	900.0	NaN	NaN	NaN	NaN	30.0
November	2000.0	NaN	NaN	NaN	3174.40	115.0	NaN	2000.0	3.0	NaN	NaN	NaN	331.0

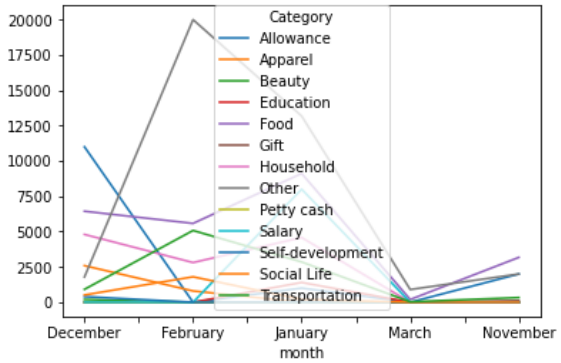
In [128]:
```
# fill values of NAN
expense.pivot_table(index ='month', columns='Category', values ='INR', aggfunc='sum',fill_value =0)
```

Out[128]:

Category	Allowance	Apparel	Beauty	Education	Food	Gift	Household	Other	Petty cash	Salary	Self-development	Social Life	Transportation
month													
December	11000	2590	196	0	6440.72	0	4800	1790	0	0	400	513.72	914.0
February	0	798	0	0	5579.85	0	2808	20000	0	0	0	1800.00	5078.8
January	1000	0	0	1400	9112.51	0	4580	13178	0	8000	0	200.00	2850.0
March	0	0	0	0	195.00	0	0	900	0	0	0	0.00	30.0
November	2000	0	0	0	3174.40	115	0	2000	3	0	0	0.00	331.0

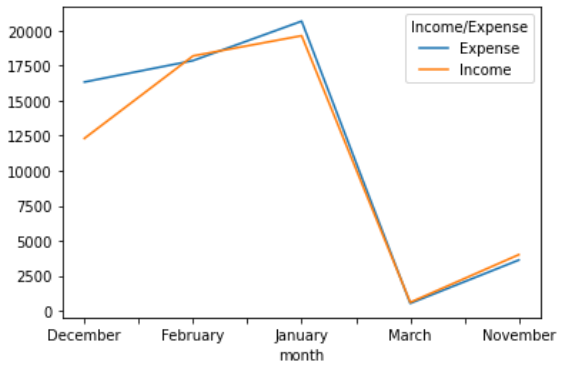
In [131]:
```
# plot
expense.pivot_table(index ='month', columns='Category', values ='INR', aggfunc='sum',fill_value =0).plot()
```

Out[131]: <AxesSubplot:xlabel='month'>



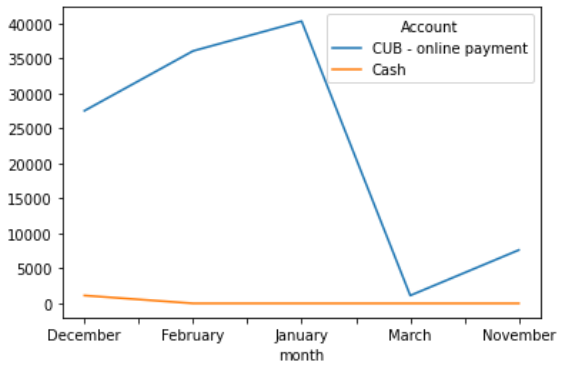
In [132]:
```
expense.pivot_table(index ='month', columns='Income/Expense', values ='INR', aggfunc='sum',fill_value =0).plot()
```

Out[132]: <AxesSubplot:xlabel='month'>



In [133]:
```
expense.pivot_table(index ='month', columns='Account', values ='INR', aggfunc='sum',fill_value =0).plot()
```

Out[133]: <AxesSubplot:xlabel='month'>



In [ ]:
```

```

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

What are vectorized operations

vectorized operations are a way to perform operations on entire arrays of data at once, which is faster than doing them one at a time.

In [2]:
```
a = np.array([1,2,3,4])
a * 4
```

Out[2]:
```
array([ 4,  8, 12, 16])
```

In [3]:
```
# problem in vectorized opertions in vanilla python
s = ['cat','mat',None,'rat']
[i.startswith('c') for i in s]
## Throws an error , because Startswith only works on strings
```

In [4]:
```
# How pandas solves this issue?

s = pd.Series(['cat','mat',None,'rat'])
s
```

Out[4]:
```
0     cat
1     mat
2    None
3     rat
dtype: object
```

here , **str = string accesor**

In [5]:
```
s.str.startswith('c') # Fast and optimized for larger datasets.
```

Out[5]:
```
0     True
1    False
2     None
3    False
dtype: object
```

In [6]:
```
# import titanic dataset
df =pd.read_csv("titanic.csv")
```

In [7]:
```
df.head(1)
```

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	NaN	

In [8]:
```
df['Name']
```

Out[8]:
```
0                                Braund, Mr. Owen Harris
1      Cumings, Mrs. John Bradley (Florence Briggs Th...
2                                 Heikkinen, Miss. Laina
3           Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                               Allen, Mr. William Henry
                             ...
886                              Montvila, Rev. Juozas
887                       Graham, Miss. Margaret Edith
888           Johnston, Miss. Catherine Helen "Carrie"
889                              Behr, Mr. Karl Howell
890                                Dooley, Mr. Patrick
Name: Name, Length: 891, dtype: object
```

Common Functions

lower/upper/capitalize/title

In [9]:
```
# Upper
df['Name'].str.upper() # converts into Capital Words
```

Out[9]:
```
0                                BRAUND, MR. OWEN HARRIS
1      CUMINGS, MRS. JOHN BRADLEY (FLORENCE BRIGGS TH...
2                                 HEIKKINEN, MISS. LAINA
3           FUTRELLE, MRS. JACQUES HEATH (LILY MAY PEEL)
4                               ALLEN, MR. WILLIAM HENRY
                             ...
886                              MONTVILA, REV. JUOZAS
887                       GRAHAM, MISS. MARGARET EDITH
888           JOHNSTON, MISS. CATHERINE HELEN "CARRIE"
889                              BEHR, MR. KARL HOWELL
890                                DOOLEY, MR. PATRICK
Name: Name, Length: 891, dtype: object
```

In [10]:
```
# Lower
df['Name'].str.lower() # converts into small Words
```

Out[10]:
```
0                                 braund, mr. owen harris
1       cumings, mrs. john bradley (florence briggs th...
2                                  heikkinen, miss. laina
3            futrelle, mrs. jacques heath (lily may peel)
4                                allen, mr. william henry
                              ...
886                                 montvila, rev. juozas
887                          graham, miss. margaret edith
888              johnston, miss. catherine helen "carrie"
889                                 behr, mr. karl howell
890                                   dooley, mr. patrick
Name: Name, Length: 891, dtype: object
```

In [11]:
```
# title
df['Name'].str.title() # converts into starting letter of Word to Capital
```

Out[11]:
```
0                                 Braund, Mr. Owen Harris
1       Cumings, Mrs. John Bradley (Florence Briggs Th...
2                                  Heikkinen, Miss. Laina
3            Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                                Allen, Mr. William Henry
                              ...
886                                 Montvila, Rev. Juozas
887                          Graham, Miss. Margaret Edith
888              Johnston, Miss. Catherine Helen "Carrie"
889                                 Behr, Mr. Karl Howell
890                                   Dooley, Mr. Patrick
Name: Name, Length: 891, dtype: object
```

In [12]:
```
# lets try to find the longest name In the passengers
df['Name'].str.len().max()
```

Out[12]:
```
82
```

In [13]:
```
df['Name'][df['Name'].str.len()== 82]
```

Out[13]:
```
307    Penasco y Castellana, Mrs. Victor de Satode (M...
Name: Name, dtype: object
```

In [14]:
```
df['Name'][df['Name'].str.len()== 82].values[0]
```

Out[14]:
```
'Penasco y Castellana, Mrs. Victor de Satode (Maria Josefa Perez de Soto y Va
llejo)'
```

strip

In [16]:
```
'          jack  '.strip()
```

Out[16]:
```
'jack'
```

In [17]:
```
df['Name'].str.strip() # removes spaces
```

Out[17]:
```
0                                Braund, Mr. Owen Harris
1      Cumings, Mrs. John Bradley (Florence Briggs Th...
2                                 Heikkinen, Miss. Laina
3           Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                               Allen, Mr. William Henry
                             ...
886                               Montvila, Rev. Juozas
887                        Graham, Miss. Margaret Edith
888            Johnston, Miss. Catherine Helen "Carrie"
889                               Behr, Mr. Karl Howell
890                                 Dooley, Mr. Patrick
Name: Name, Length: 891, dtype: object
```

split

In [19]:
```
# split
df['Name'].str.split(',')
```

Out[19]:
```
0                             [Braund,  Mr. Owen Harris]
1      [Cumings,  Mrs. John Bradley (Florence Briggs ...
2                              [Heikkinen,  Miss. Laina]
3        [Futrelle,  Mrs. Jacques Heath (Lily May Peel)]
4                            [Allen,  Mr. William Henry]
                             ...
886                            [Montvila,  Rev. Juozas]
887                      [Graham,  Miss. Margaret Edith]
888          [Johnston,  Miss. Catherine Helen "Carrie"]
889                              [Behr,  Mr. Karl Howell]
890                                [Dooley,  Mr. Patrick]
Name: Name, Length: 891, dtype: object
```

In [21]:
```
# Split -> get
df['Name'].str.split(',').str.get(0)
```

Out[21]:
```
0         Braund
1       Cumings
2      Heikkinen
3       Futrelle
4          Allen
         ...
886     Montvila
887       Graham
888     Johnston
889         Behr
890       Dooley
Name: Name, Length: 891, dtype: object
```

In [22]:
```
df['Name'].str.split(',').str.get(1)
```

Out[22]:
```
0                                  Mr. Owen Harris
1        Mrs. John Bradley (Florence Briggs Thayer)
2                                     Miss. Laina
3               Mrs. Jacques Heath (Lily May Peel)
4                                  Mr. William Henry
                          ...
886                                   Rev. Juozas
887                          Miss. Margaret Edith
888                Miss. Catherine Helen "Carrie"
889                                   Mr. Karl Howell
890                                   Mr. Patrick
Name: Name, Length: 891, dtype: object
```

In [23]:
```
df['last_name'] = df['Name'].str.split(',').str.get(0)
```

In [25]:
```
df.head(1)
```

Out[25]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	NaN	

In [29]:
```
# it is used to split the Name column of the DataFrame df into two columns
# FirstName and LastName.

df['Name'].str.split(',').str.get(1).str.strip().str.split(' ',n=1, expand=Tru
```

Out[29]:

	0	1
0	Mr.	Owen Harris
1	Mrs.	John Bradley (Florence Briggs Thayer)
2	Miss.	Laina
3	Mrs.	Jacques Heath (Lily May Peel)
4	Mr.	William Henry
...	...	...
886	Rev.	Juozas
887	Miss.	Margaret Edith
888	Miss.	Catherine Helen "Carrie"
889	Mr.	Karl Howell
890	Mr.	Patrick

891 rows × 2 columns

In [30]:
```
df[['title','firstname']]= df['Name'].str.split(',').str.get(1).str.strip().st
```

In [31]:
```
df.head(1)
```

Out[31]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	NaN	

In [33]:
```
# Number of titles
df['title'].value_counts()
```

Out[33]:
```
Mr.          517
Miss.        182
Mrs.         125
Master.       40
Dr.            7
Rev.           6
Mlle.          2
Major.         2
Col.           2
the            1
Capt.          1
Ms.            1
Sir.           1
Lady.          1
Mme.           1
Don.           1
Jonkheer.      1
Name: title, dtype: int64
```

replace

In [36]:
```
df['title'] = df['title'].str.replace('Ms.','Miss.')
df['title'] = df['title'].str.replace('Mlle.','Miss.')
```

```
C:\Users\user\AppData\Local\Temp/ipykernel_15952/1805277261.py:1: FutureWarni
ng: The default value of regex will change from True to False in a future ver
sion.
  df['title'] = df['title'].str.replace('Ms.','Miss.')
C:\Users\user\AppData\Local\Temp/ipykernel_15952/1805277261.py:2: FutureWarni
ng: The default value of regex will change from True to False in a future ver
sion.
  df['title'] = df['title'].str.replace('Mlle.','Miss.')
```

In [37]:
```
df['title'].value_counts()
```

Out[37]:
```
Mr.          517
Miss.        185
Mrs.         125
Master.       40
Dr.            7
Rev.           6
Major.         2
Col.           2
Don.           1
Mme.           1
Lady.          1
Sir.           1
Capt.          1
the            1
Jonkheer.      1
Name: title, dtype: int64
```

Filtering

In [38]:
```
# startswith/endswith
df[df['firstname'].str.startswith('A')]
```

Out[38]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
13	14	0	3	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750
22	23	1	3	McGowan, Miss. Anna "Annie"	female	15.0	0	0	330923	8.0292
35	36	0	1	Holverson, Mr. Alexander Oskar	male	42.0	1	0	113789	52.0000
38	39	0	3	Vander Planke, Miss. Augusta Maria	female	18.0	2	0	345764	18.0000
61	62	1	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0000
...	...	...	...	...	...	...	...	...	...	...
842	843	1	1	Serepeca, Miss. Augusta	female	30.0	0	0	113798	31.0000
845	846	0	3	Abbing, Mr. Anthony	male	42.0	0	0	C.A. 5547	7.5500
866	867	1	2	Duran y More, Miss. Asuncion	female	27.0	1	0	SC/PARIS 2149	13.8583
875	876	1	3	Najib, Miss. Adele Kiamie "Jane"	female	15.0	0	0	2667	7.2250
876	877	0	3	Gustafsson, Mr. Alfred Ossian	male	20.0	0	0	7534	9.8458

95 rows × 15 columns

In [40]:
```
# endswith
df[df['firstname'].str.endswith('z')]
```

Out[40]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabi
69	70	0	3	Kink, Mr. Vincenz	male	26.0	2	0	315151	8.6625	Na
679	680	1	1	Cardeza, Mr. Thomas Drake Martinez	male	36.0	0	1	PC 17755	512.3292	B5 B5 B5
721	722	0	3	Jensen, Mr. Svend Lauritz	male	17.0	1	0	350048	7.0542	Na

In [41]:
```
# isdigit/isalpha...
df[df['firstname'].str.isdigit()]
```

Out[41]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarke

regex

In [42]:
```
# applying regex
# contains
# search john -> both case
df[df['firstname'].str.contains('john',case=False)]
```

Out[42]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	7
41	42	0	2	Turpin, Mrs. William John Robert (Dorothy Ann ...	female	27.0	1	0	11668	2
45	46	0	3	Rogers, Mr. William John	male	NaN	0	0	S.C./A.4. 23567	
98	99	1	2	Doling, Mrs. John T (Ada Julia Bone)	female	34.0	0	1	231919	2

In [44]:
```
# find lastnames with start and end char vowel ( aeiou)
df[df['last_name'].str.contains('^[^aeiouAEIOU].+[^aeiouAEIOU]$')]
```

Out[44]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625
...	...	...	...	...	...	...	...	...	...	...
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.0500
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500

671 rows × 15 columns

slicing

In [45]:
```
df['Name'].str[:4] # first 4 characters
```

Out[45]:
```
0      Brau
1      Cumi
2      Heik
3      Futr
4      Alle
       ...
886    Mont
887    Grah
888    John
889    Behr
890    Dool
Name: Name, Length: 891, dtype: object
```

In [46]:
```
df['Name'].str[::2] # alternate characters
```

Out[46]:
```
0                    Ban,M.Oe ars
1      Cmns r.Jh rde Foec rgsTae)
2                     Hiknn is an
3          Ftel,Ms aqe et Ll a el
4                    Aln r ila er
                  ...
886                   Mnvl,Rv uzs
887                Gaa,Ms.Mrae dt
888          Jhso,Ms.CteieHln"are
889                   Bh,M.Kr oel
890                    Doe,M.Ptik
Name: Name, Length: 891, dtype: object
```

In [47]:
```
df['Name'].str[::-1] # reverse
```

Out[47]:
```
0                                sirraH newO .rM ,dnuarB
1      )reyahT sggirB ecnerolF( yeldarB nhoJ .srM ,sg...
2                                  aniaL .ssiM ,nenikkieH
3           )leeP yaM yliL( htaeH seuqcaJ .srM ,ellertuF
4                               yrneH mailliW .rM ,nellA
                              ...
886                                sazouJ .veR ,alivtnoM
887                          htidE teragraM .ssiM ,maharG
888             "eirraC" neleH enirehtaC .ssiM ,notsnhoJ
889                                llewoH lraK .rM ,rheB
890                                  kcirtaP .rM ,yelooD
Name: Name, Length: 891, dtype: object
```

In [ ]:
```

```

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

Timestamp Object

Time stamps reference particular moments in time (e.g., Oct 24th, 2022 at 7:00pm)

Vectorized date and time operations are a powerful tool for working with date and time data. They can be used to quickly and easily perform a wide variety of operations on date and time data.

Creating Timestamp objects

In [2]:
```
# creating a timestamp
pd.Timestamp('2023/05/12')
# This time stamp contains year-month- Day ,hour-minute-Second
```

Out[2]:
```
Timestamp('2023-05-12 00:00:00')
```

In [3]:
```
# type
type(pd.Timestamp('2023/05/12'))
```

Out[3]:
```
pandas._libs.tslibs.timestamps.Timestamp
```

In [4]:
```
# Variations
pd.Timestamp('2023-05-12')
```

Out[4]:
```
Timestamp('2023-05-12 00:00:00')
```

In [5]:
```
pd.Timestamp('2023,05,12')
```

Out[5]:
```
Timestamp('2023-12-01 00:00:00')
```

In [6]:
```
pd.Timestamp('2023.05.12')
```

Out[6]:
```
Timestamp('2023-05-12 00:00:00')
```

In [7]:
```
# only year
pd.Timestamp('2023') # It Automatically assigns First day of the year.
```

Out[7]:
```
Timestamp('2023-01-01 00:00:00')
```

In [8]:
```
# Using text
pd.Timestamp('12th May 2023')
```

Out[8]:
```
Timestamp('2023-05-12 00:00:00')
```

In [9]:
```
# Provide time also
pd.Timestamp('12th May 2023 4:40PM')
```

Out[9]:
```
Timestamp('2023-05-12 16:40:00')
```

In [10]:
```
# using Python's datetime object
import datetime as dt

x = pd.Timestamp(dt.datetime(2023,5,12,4,42,56))
x
```

Out[10]:
```
Timestamp('2023-05-12 04:42:56')
```

In [12]:
```
# Fetching attributes
x.year
```

Out[12]:
```
2023
```

In [15]:
```
x.day
```

Out[15]:
```
12
```

In [17]:
```
x.time()
```

Out[17]:
```
datetime.time(4, 42, 56)
```

In [18]:
```
x.month
```

Out[18]:
```
5
```

why separate objects to handle data and time when python already has datetime functionality?

- syntax wise datetime is very convenient
- But the performance takes a hit while working with huge data. List vs Numpy Array
- The weaknesses of Python's datetime format inspired the NumPy team to add a set of native time series data type to NumPy.
- The datetime64 dtype encodes dates as 64-bit integers, and thus allows arrays of dates to be represented very compactly.

In [19]:
```
import numpy as np
date = np.array('2023-05-12', dtype=np.datetime64)
date
```

Out[19]:
```
array('2023-05-12', dtype='datetime64[D]')
```

In [21]:
```
# We can operate vector operations on Date.
date + np.arange(20)
```

Out[21]:
```
array(['2023-05-12', '2023-05-13', '2023-05-14', '2023-05-15',
       '2023-05-16', '2023-05-17', '2023-05-18', '2023-05-19',
       '2023-05-20', '2023-05-21', '2023-05-22', '2023-05-23',
       '2023-05-24', '2023-05-25', '2023-05-26', '2023-05-27',
       '2023-05-28', '2023-05-29', '2023-05-30', '2023-05-31'],
      dtype='datetime64[D]')
```

- Because of the uniform type in NumPy datetime64 arrays, this type of operation can be accomplished much more quickly than if we were working directly with Python's datetime objects, especially as arrays get large
- Pandas **Timestamp** object combines the ease-of-use of python datetime with the efficient storage and vectorized interface of numpy.datetime64
- From a group of these Timestamp objects, Pandas can construct a DatetimeIndex that can be used to index data in a Series or DataFrame

DatetimeIndex Object

A collection of pandas timestamp

In [22]:
```
# using strings
pd.DatetimeIndex(['2023/05/12','2023/01/01','2025/01/22'])
```

Out[22]:
```
DatetimeIndex(['2023-05-12', '2023-01-01', '2025-01-22'], dtype='datetime64[ns]', freq=None)
```

In [23]:
```
pd.DatetimeIndex(['2023/05/12','2023/01/01','2025/01/22'])[0]
```

Out[23]:
```
Timestamp('2023-05-12 00:00:00')
```

In [25]:
```
# type
type(pd.DatetimeIndex(['2023/05/12','2023/01/01','2025/01/22']))
```

Out[25]:
```
pandas.core.indexes.datetimes.DatetimeIndex
```

To store a **single date**, we use Timestamp.
And to store **multiple date** and time we use date time index.

In [26]:
```
# using python datetime object
pd.DatetimeIndex([dt.datetime(2023,5,12),dt.datetime(2023,1,1),dt.datetime(2025,1,1)])
```

Out[26]:
```
DatetimeIndex(['2023-05-12', '2023-01-01', '2025-01-01'], dtype='datetime64[ns]', freq=None)
```

In [27]:
```
# using pd.timestamps
dt_index = pd.DatetimeIndex([pd.Timestamp(2023,1,1),pd.Timestamp(2022,1,1),pd.Timestamp(2021,1,1)]
```

In [28]:
```
dt_index
```

Out[28]:
```
DatetimeIndex(['2023-01-01', '2022-01-01', '2021-01-01'], dtype='datetime64[ns]', freq=None)
```

In [29]:
```
# using datatimeindex as series index

pd.Series([1,2,3],index=dt_index)
```

Out[29]:
```
2023-01-01    1
2022-01-01    2
2021-01-01    3
dtype: int64
```

date_range function

In [33]:
```
# generate daily dates in a given range
pd.date_range(start='2023/5/12',end='2023/6/12',freq='D')
```

Out[33]:
```
DatetimeIndex(['2023-05-12', '2023-05-13', '2023-05-14', '2023-05-15',
               '2023-05-16', '2023-05-17', '2023-05-18', '2023-05-19',
               '2023-05-20', '2023-05-21', '2023-05-22', '2023-05-23',
               '2023-05-24', '2023-05-25', '2023-05-26', '2023-05-27',
               '2023-05-28', '2023-05-29', '2023-05-30', '2023-05-31',
               '2023-06-01', '2023-06-02', '2023-06-03', '2023-06-04',
               '2023-06-05', '2023-06-06', '2023-06-07', '2023-06-08',
               '2023-06-09', '2023-06-10', '2023-06-11', '2023-06-12'],
              dtype='datetime64[ns]', freq='D')
```

In [34]:
```
# Alternate days
pd.date_range(start='2023/5/12',end='2023/6/12',freq='2D')
```

Out[34]:
```
DatetimeIndex(['2023-05-12', '2023-05-14', '2023-05-16', '2023-05-18',
               '2023-05-20', '2023-05-22', '2023-05-24', '2023-05-26',
               '2023-05-28', '2023-05-30', '2023-06-01', '2023-06-03',
               '2023-06-05', '2023-06-07', '2023-06-09', '2023-06-11'],
              dtype='datetime64[ns]', freq='2D')
```

In [35]:
```
# 2 days gap
pd.date_range(start='2023/5/12',end='2023/6/12',freq='3D')
```

Out[35]:
```
DatetimeIndex(['2023-05-12', '2023-05-15', '2023-05-18', '2023-05-21',
               '2023-05-24', '2023-05-27', '2023-05-30', '2023-06-02',
               '2023-06-05', '2023-06-08', '2023-06-11'],
              dtype='datetime64[ns]', freq='3D')
```

In [36]:
```
# B -> business days (MON- FRI)
pd.date_range(start='2023/5/12',end='2023/6/12',freq='B')
```

Out[36]:
```
DatetimeIndex(['2023-05-12', '2023-05-15', '2023-05-16', '2023-05-17',
               '2023-05-18', '2023-05-19', '2023-05-22', '2023-05-23',
               '2023-05-24', '2023-05-25', '2023-05-26', '2023-05-29',
               '2023-05-30', '2023-05-31', '2023-06-01', '2023-06-02',
               '2023-06-05', '2023-06-06', '2023-06-07', '2023-06-08',
               '2023-06-09', '2023-06-12'],
              dtype='datetime64[ns]', freq='B')
```

In [37]:
```
# W -> one week per day (SUN)
pd.date_range(start='2023/5/12',end='2023/6/12',freq='w')
```

Out[37]:
```
DatetimeIndex(['2023-05-14', '2023-05-21', '2023-05-28', '2023-06-04',
               '2023-06-11'],
              dtype='datetime64[ns]', freq='W-SUN')
```

In [38]:
```
# if you want specific Day (THU)
pd.date_range(start='2023/5/12',end='2023/6/12',freq='w-THU')
```

Out[38]:
```
DatetimeIndex(['2023-05-18', '2023-05-25', '2023-06-01', '2023-06-08'], dtype='datetime64[ns]', f
req='W-THU')
```

In [39]:
```
# H -> Hourly data(factor)
pd.date_range(start='2023/5/12',end='2023/6/12',freq='H')
```

Out[39]:
```
DatetimeIndex(['2023-05-12 00:00:00', '2023-05-12 01:00:00',
               '2023-05-12 02:00:00', '2023-05-12 03:00:00',
               '2023-05-12 04:00:00', '2023-05-12 05:00:00',
               '2023-05-12 06:00:00', '2023-05-12 07:00:00',
               '2023-05-12 08:00:00', '2023-05-12 09:00:00',
               ...
               '2023-06-11 15:00:00', '2023-06-11 16:00:00',
               '2023-06-11 17:00:00', '2023-06-11 18:00:00',
               '2023-06-11 19:00:00', '2023-06-11 20:00:00',
               '2023-06-11 21:00:00', '2023-06-11 22:00:00',
               '2023-06-11 23:00:00', '2023-06-12 00:00:00'],
              dtype='datetime64[ns]', length=745, freq='H')
```

In [41]:
```
# For every six hours
pd.date_range(start='2023/5/12',end='2023/6/12',freq='6H')
```

Out[41]:
```
DatetimeIndex(['2023-05-12 00:00:00', '2023-05-12 06:00:00',
               '2023-05-12 12:00:00', '2023-05-12 18:00:00',
               '2023-05-13 00:00:00', '2023-05-13 06:00:00',
               '2023-05-13 12:00:00', '2023-05-13 18:00:00',
               '2023-05-14 00:00:00', '2023-05-14 06:00:00',
               ...
               '2023-06-09 18:00:00', '2023-06-10 00:00:00',
               '2023-06-10 06:00:00', '2023-06-10 12:00:00',
               '2023-06-10 18:00:00', '2023-06-11 00:00:00',
               '2023-06-11 06:00:00', '2023-06-11 12:00:00',
               '2023-06-11 18:00:00', '2023-06-12 00:00:00'],
              dtype='datetime64[ns]', length=125, freq='6H')
```

In [42]:
```
# M -> Month end
pd.date_range(start='2023/5/12',end='2023/6/12',freq='M')
```

Out[42]:
```
DatetimeIndex(['2023-05-31'], dtype='datetime64[ns]', freq='M')
```

In [47]:
```
# MS -> Month start
pd.date_range(start='2023/5/12',end='2028/6/12',freq='MS')
```

Out[47]:
```
DatetimeIndex(['2023-06-01', '2023-07-01', '2023-08-01', '2023-09-01',
               '2023-10-01', '2023-11-01', '2023-12-01', '2024-01-01',
               '2024-02-01', '2024-03-01', '2024-04-01', '2024-05-01',
               '2024-06-01', '2024-07-01', '2024-08-01', '2024-09-01',
               '2024-10-01', '2024-11-01', '2024-12-01', '2025-01-01',
               '2025-02-01', '2025-03-01', '2025-04-01', '2025-05-01',
               '2025-06-01', '2025-07-01', '2025-08-01', '2025-09-01',
               '2025-10-01', '2025-11-01', '2025-12-01', '2026-01-01',
               '2026-02-01', '2026-03-01', '2026-04-01', '2026-05-01',
               '2026-06-01', '2026-07-01', '2026-08-01', '2026-09-01',
               '2026-10-01', '2026-11-01', '2026-12-01', '2027-01-01',
               '2027-02-01', '2027-03-01', '2027-04-01', '2027-05-01',
               '2027-06-01', '2027-07-01', '2027-08-01', '2027-09-01',
               '2027-10-01', '2027-11-01', '2027-12-01', '2028-01-01',
               '2028-02-01', '2028-03-01', '2028-04-01', '2028-05-01',
               '2028-06-01'],
              dtype='datetime64[ns]', freq='MS')
```

In [46]:
```
# A -> Year end
pd.date_range(start='2023/5/12',end='2030/6/12',freq='A')
```

Out[46]:
```
DatetimeIndex(['2023-12-31', '2024-12-31', '2025-12-31', '2026-12-31',
               '2027-12-31', '2028-12-31', '2029-12-31'],
              dtype='datetime64[ns]', freq='A-DEC')
```

In [49]:
```
# using periods(number of results)
pd.date_range(start='2023/5/12',periods =30,freq='D')
```

Out[49]:
```
DatetimeIndex(['2023-05-12', '2023-05-13', '2023-05-14', '2023-05-15',
               '2023-05-16', '2023-05-17', '2023-05-18', '2023-05-19',
               '2023-05-20', '2023-05-21', '2023-05-22', '2023-05-23',
               '2023-05-24', '2023-05-25', '2023-05-26', '2023-05-27',
               '2023-05-28', '2023-05-29', '2023-05-30', '2023-05-31',
               '2023-06-01', '2023-06-02', '2023-06-03', '2023-06-04',
               '2023-06-05', '2023-06-06', '2023-06-07', '2023-06-08',
               '2023-06-09', '2023-06-10'],
              dtype='datetime64[ns]', freq='D')
```

In [50]:
```
# Hour (using periods)
pd.date_range(start='2023/5/12',periods =30,freq='H')
```

Out[50]:
```
DatetimeIndex(['2023-05-12 00:00:00', '2023-05-12 01:00:00',
               '2023-05-12 02:00:00', '2023-05-12 03:00:00',
               '2023-05-12 04:00:00', '2023-05-12 05:00:00',
               '2023-05-12 06:00:00', '2023-05-12 07:00:00',
               '2023-05-12 08:00:00', '2023-05-12 09:00:00',
               '2023-05-12 10:00:00', '2023-05-12 11:00:00',
               '2023-05-12 12:00:00', '2023-05-12 13:00:00',
               '2023-05-12 14:00:00', '2023-05-12 15:00:00',
               '2023-05-12 16:00:00', '2023-05-12 17:00:00',
               '2023-05-12 18:00:00', '2023-05-12 19:00:00',
               '2023-05-12 20:00:00', '2023-05-12 21:00:00',
               '2023-05-12 22:00:00', '2023-05-12 23:00:00',
               '2023-05-13 00:00:00', '2023-05-13 01:00:00',
               '2023-05-13 02:00:00', '2023-05-13 03:00:00',
               '2023-05-13 04:00:00', '2023-05-13 05:00:00'],
              dtype='datetime64[ns]', freq='H')
```

In [51]:
```
# 6 Hours (using periods)
pd.date_range(start='2023/5/12',periods =30,freq='6H')
```

Out[51]:
```
DatetimeIndex(['2023-05-12 00:00:00', '2023-05-12 06:00:00',
               '2023-05-12 12:00:00', '2023-05-12 18:00:00',
               '2023-05-13 00:00:00', '2023-05-13 06:00:00',
               '2023-05-13 12:00:00', '2023-05-13 18:00:00',
               '2023-05-14 00:00:00', '2023-05-14 06:00:00',
               '2023-05-14 12:00:00', '2023-05-14 18:00:00',
               '2023-05-15 00:00:00', '2023-05-15 06:00:00',
               '2023-05-15 12:00:00', '2023-05-15 18:00:00',
               '2023-05-16 00:00:00', '2023-05-16 06:00:00',
               '2023-05-16 12:00:00', '2023-05-16 18:00:00',
               '2023-05-17 00:00:00', '2023-05-17 06:00:00',
               '2023-05-17 12:00:00', '2023-05-17 18:00:00',
               '2023-05-18 00:00:00', '2023-05-18 06:00:00',
               '2023-05-18 12:00:00', '2023-05-18 18:00:00',
               '2023-05-19 00:00:00', '2023-05-19 06:00:00'],
              dtype='datetime64[ns]', freq='6H')
```

In [52]:
```
# Month (using periods)
pd.date_range(start='2023/5/12',periods =30,freq='M')
```

Out[52]:
```
DatetimeIndex(['2023-05-31', '2023-06-30', '2023-07-31', '2023-08-31',
               '2023-09-30', '2023-10-31', '2023-11-30', '2023-12-31',
               '2024-01-31', '2024-02-29', '2024-03-31', '2024-04-30',
               '2024-05-31', '2024-06-30', '2024-07-31', '2024-08-31',
               '2024-09-30', '2024-10-31', '2024-11-30', '2024-12-31',
               '2025-01-31', '2025-02-28', '2025-03-31', '2025-04-30',
               '2025-05-31', '2025-06-30', '2025-07-31', '2025-08-31',
               '2025-09-30', '2025-10-31'],
              dtype='datetime64[ns]', freq='M')
```

to_datetime function

converts an existing objects to pandas timestamp/datetimeindex object

In [59]:
```
# simple series example
s = pd.Series(['2023/5/12','2022/1/1','2021/2/1'])
pd.to_datetime(s).dt.year # converting string to datetime
```

Out[59]:
```
0    2023
1    2022
2    2021
dtype: int64
```

In [60]:
```
pd.to_datetime(s).dt.day
```

Out[60]:
```
0    12
1     1
2     1
dtype: int64
```

In [61]:
```
pd.to_datetime(s).dt.day_name()
```

Out[61]:
```
0      Friday
1    Saturday
2      Monday
dtype: object
```

In [62]:
```
pd.to_datetime(s).dt.month_name()
```

Out[62]:
```
0         May
1     January
2    February
dtype: object
```

In [63]:
```
# with errors -> coerce

s = pd.Series(['2023/1/1','2022/1/1','2021/130/1'])
pd.to_datetime(s,errors='coerce') #NaT = Not a Time
```

Out[63]:
```
0   2023-01-01
1   2022-01-01
2          NaT
dtype: datetime64[ns]
```

In [64]:
```
pd.to_datetime(s,errors='coerce').dt.year
```

Out[64]:
```
0    2023.0
1    2022.0
2       NaN
dtype: float64
```

In [65]:
```
pd.to_datetime(s,errors='coerce').dt.month_name()
```

Out[65]:
```
0    January
1    January
2        NaN
dtype: object
```

Real World example

In [66]:
```
df = pd.read_csv("expense_data.csv")
```

In [69]:
```
df.head()
```

Out[69]:

	Date	Account	Category	Subcategory	Note	INR	Income/Expense	Note.1	Amount	Currency	Account.1
0	3/2/2022 10:11	CUB - online payment	Food	NaN	Brownie	50.0	Expense	NaN	50.0	INR	50.0
1	3/2/2022 10:11	CUB - online payment	Other	NaN	To lended people	300.0	Expense	NaN	300.0	INR	300.0
2	3/1/2022 19:50	CUB - online payment	Food	NaN	Dinner	78.0	Expense	NaN	78.0	INR	78.0
3	3/1/2022 18:56	CUB - online payment	Transportation	NaN	Metro	30.0	Expense	NaN	30.0	INR	30.0
4	3/1/2022 18:22	CUB - online payment	Food	NaN	Snacks	67.0	Expense	NaN	67.0	INR	67.0

In [70]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 277 entries, 0 to 276
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Date            277 non-null    object 
 1   Account         277 non-null    object 
 2   Category        277 non-null    object 
 3   Subcategory     0 non-null      float64
 4   Note            273 non-null    object 
 5   INR             277 non-null    float64
 6   Income/Expense  277 non-null    object 
 7   Note.1          0 non-null      float64
 8   Amount          277 non-null    float64
 9   Currency        277 non-null    object 
 10  Account.1       277 non-null    float64
dtypes: float64(5), object(6)
memory usage: 23.9+ KB
```

In [72]:
```
# converting object to date time type
df['Date'] = pd.to_datetime(df['Date'])
```

In [73]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 277 entries, 0 to 276
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype         
---  ------          --------------  -----         
 0   Date            277 non-null    datetime64[ns]
 1   Account         277 non-null    object        
 2   Category        277 non-null    object        
 3   Subcategory     0 non-null      float64       
 4   Note            273 non-null    object        
 5   INR             277 non-null    float64       
 6   Income/Expense  277 non-null    object        
 7   Note.1          0 non-null      float64       
 8   Amount          277 non-null    float64       
 9   Currency        277 non-null    object        
 10  Account.1       277 non-null    float64       
dtypes: datetime64[ns](1), float64(5), object(5)
memory usage: 23.9+ KB
```

dt accessor

Accessor object for datetimelike properties of the Series values.

In [75]:
```
df['Date'].dt.year
```

Out[75]:
```
0      2022
1      2022
2      2022
3      2022
4      2022
       ... 
272    2021
273    2021
274    2021
275    2021
276    2021
Name: Date, Length: 277, dtype: int64
```

In [76]:
```
df['Date'].dt.month
```

Out[76]:
```
0       3
1       3
2       3
3       3
4       3
       ..
272    11
273    11
274    11
275    11
276    11
Name: Date, Length: 277, dtype: int64
```

In [77]:
```
df['Date'].dt.month_name()
```

Out[77]:
```
0         March
1         March
2         March
3         March
4         March
         ...
272    November
273    November
274    November
275    November
276    November
Name: Date, Length: 277, dtype: object
```

In [80]:
```
df['Date'].dt.day_name()
```

Out[80]:
```
0      Wednesday
1      Wednesday
2        Tuesday
3        Tuesday
4        Tuesday
         ...
272       Monday
273       Monday
274       Sunday
275       Sunday
276       Sunday
Name: Date, Length: 277, dtype: object
```

In [86]:
```
df['Date'].dt.is_month_end
```

Out[86]:
```
0      False
1      False
2      False
3      False
4      False
       ...
272    False
273    False
274    False
275    False
276    False
Name: Date, Length: 277, dtype: bool
```

In [87]:
```
df['Date'].dt.is_year_end
```

Out[87]:
```
0      False
1      False
2      False
3      False
4      False
       ...
272    False
273    False
274    False
275    False
276    False
Name: Date, Length: 277, dtype: bool
```

In [90]:
```
df['Date'].dt.is_quarter_end
```

Out[90]:
```
0      False
1      False
2      False
3      False
4      False
       ...
272    False
273    False
274    False
275    False
276    False
Name: Date, Length: 277, dtype: bool
```

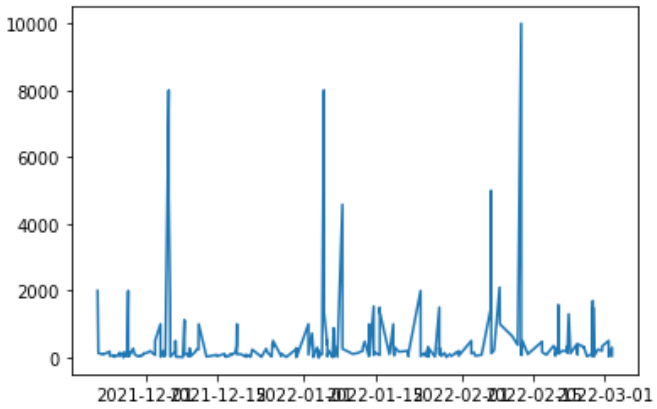
In [91]:
```
df['Date'].dt.is_quarter_start
```

Out[91]:
```
0      False
1      False
2      False
3      False
4      False
       ...
272    False
273    False
274    False
275    False
276    False
Name: Date, Length: 277, dtype: bool
```

In [94]:
```
## Plot Graph
import matplotlib.pyplot as plt
```

In [95]:
```
plt.plot(df['Date'],df['INR'])
```

Out[95]:
```
[<matplotlib.lines.Line2D at 0x18193cb1790>]
```



In [96]:
```
# Money spent day name wise (bar chart)
df['day_name'] = df['Date'].dt.day_name()
```

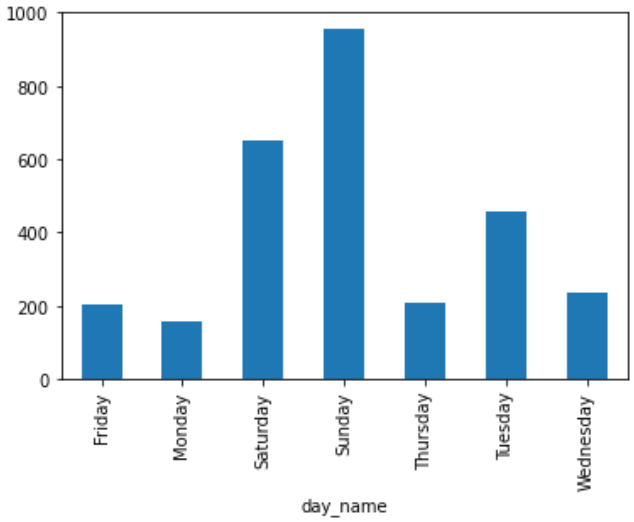
In [97]:
```
df.head()
```

Out[97]:

	Date	Account	Category	Subcategory	Note	INR	Income/Expense	Note.1	Amount	Currency	Account.1
0	2022-03-02 10:11:00	CUB - online payment	Food	NaN	Brownie	50.0	Expense	NaN	50.0	INR	50.0
1	2022-03-02 10:11:00	CUB - online payment	Other	NaN	To lended people	300.0	Expense	NaN	300.0	INR	300.0
2	2022-03-01 19:50:00	CUB - online payment	Food	NaN	Dinner	78.0	Expense	NaN	78.0	INR	78.0
3	2022-03-01 18:56:00	CUB - online payment	Transportation	NaN	Metro	30.0	Expense	NaN	30.0	INR	30.0
4	2022-03-01 18:22:00	CUB - online payment	Food	NaN	Snacks	67.0	Expense	NaN	67.0	INR	67.0

In [99]:
```
df.groupby('day_name')['INR'].mean().plot(kind='bar')
```

Out[99]:
```
<AxesSubplot:xlabel='day_name'>
```



In [100]:
```
# Money spent month name wise (pie chart)
df['month_name'] = df['Date'].dt.month_name()
```

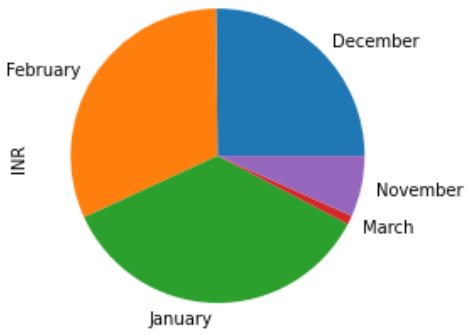
In [101]:
```
df.head()
```

Out[101]:

	Date	Account	Category	Subcategory	Note	INR	Income/Expense	Note.1	Amount	Currency	Account.1
0	2022-03-02 10:11:00	CUB - online payment	Food	NaN	Brownie	50.0	Expense	NaN	50.0	INR	50.0
1	2022-03-02 10:11:00	CUB - online payment	Other	NaN	To lended people	300.0	Expense	NaN	300.0	INR	300.0
2	2022-03-01 19:50:00	CUB - online payment	Food	NaN	Dinner	78.0	Expense	NaN	78.0	INR	78.0
3	2022-03-01 18:56:00	CUB - online payment	Transportation	NaN	Metro	30.0	Expense	NaN	30.0	INR	30.0
4	2022-03-01 18:22:00	CUB - online payment	Food	NaN	Snacks	67.0	Expense	NaN	67.0	INR	67.0

In [102]:
```
df.groupby('month_name')['INR'].sum().plot(kind = 'pie')
```

Out[102]:
```
<AxesSubplot:ylabel='INR'>
```



In [109]:
```
# Average
df.groupby('month_name')['INR'].mean().plot(kind ='bar')
```

Out[109]:
```
<AxesSubplot:xlabel='month_name'>
```