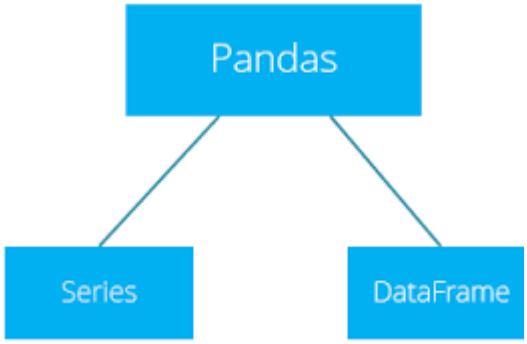
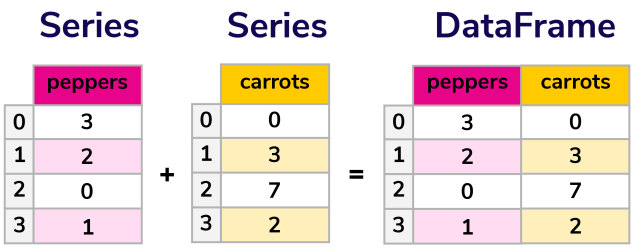
pandas

Pandas is an open-source data manipulation and analysis library for the Python programming language. It provides easy-to-use data structures and data analysis tools for working with structured data, such as tabular data (like spreadsheets or SQL tables). The name "pandas" is derived from the term "panel data," which is a type of multi-dimensional data set commonly used in statistics and econometrics.

Pandas is particularly well-suited for tasks such as data cleaning, data transformation, and data analysis. It offers two primary data structures:





Learn Python

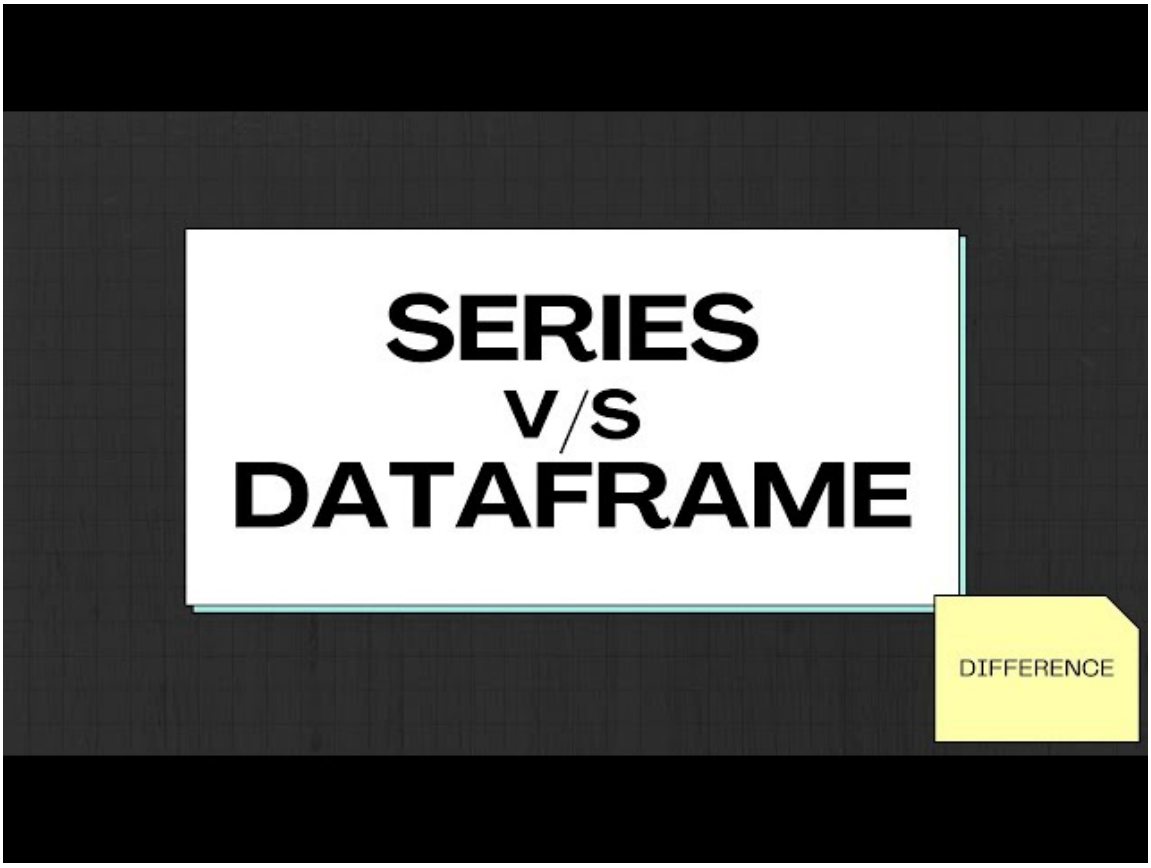
1. **Series**: A one-dimensional data structure that is essentially a labeled array, similar to a single column or row of data in a DataFrame.
2. **DataFrame**: A two-dimensional, tabular data structure resembling a spreadsheet with rows and columns. Each column can have a different data type, such as integers, floating-point numbers, strings, or dates.

Pandas provides a wide range of functions and methods for data manipulation and analysis, including:

- Data cleaning: handling missing data, data imputation, and data alignment.
- Data filtering and selection.
- Aggregation and summarization of data.
- Data merging and joining.
- Time series data manipulation.
- Reading and writing data from/to various file formats, such as CSV, Excel, SQL databases, and more.

Pandas is an essential tool for data scientists, analysts, and anyone working with data in Python. It is often used in conjunction with other libraries, such as NumPy for numerical computations and Matplotlib or Seaborn for data visualization.

Data Structures in Pandas:



1. **DataFrame:**

   - A DataFrame is a two-dimensional, tabular data structure that resembles a spreadsheet or SQL table. It consists of rows and columns.
   - Columns in a DataFrame are known as Series objects, and each Series can have a different data type (e.g., integers, floats, strings, dates).
   - Rows and columns are both labeled, allowing for easy indexing and retrieval of data.
   - DataFrames can be thought of as a collection of Series objects that share the same index.

   Example of creating a DataFrame:

In [ ]:
```
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'San Francisco', 'Los Angeles']}

df = pd.DataFrame(data)
```

1. **Series:**
   - A Series is a one-dimensional data structure that can be thought of as a single column or row from a DataFrame.
   - Each element in a Series is associated with a label, called an index.
   - Series can hold various data types, including numbers, text, and dates.

In [ ]:
```
import pandas as pd

ages = pd.Series([25, 30, 35], name='Age')
```

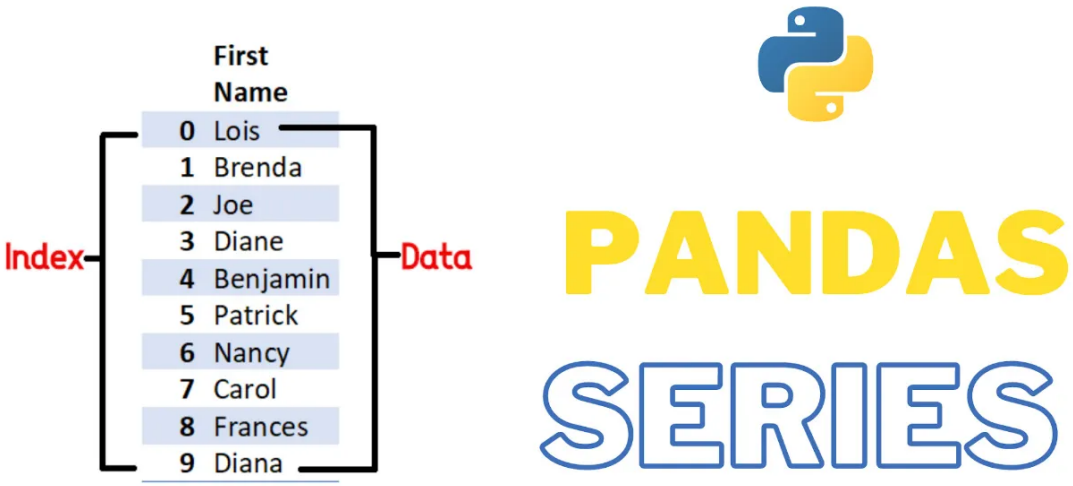
Operations and Functionality:

Pandas provides a wide range of operations and functionality for working with data, including:

1. **Data Cleaning:**
   - Handling missing data: Pandas provides methods like `isna()`, `fillna()`, and `dropna()` to deal with missing values.
   - Data imputation: You can fill missing values with meaningful data using methods like `fillna()` or statistical techniques.
2. **Data Selection and Filtering:**
   - You can select specific rows and columns, filter data based on conditions, and use boolean indexing to retrieve relevant data.
3. **Data Aggregation and Summarization:**
   - Pandas allows you to perform aggregate functions on data, such as `sum()`, `mean()`, `count()`, and more.
   - Grouping and aggregating data based on specific criteria is made easy with the `groupby()` method.
4. **Data Merging and Joining:**
   - You can merge data from multiple DataFrames using functions like `merge()` and `concat()`.
   - This is particularly useful when working with multiple data sources.
5. **Time Series Data Manipulation:**
   - Pandas has built-in support for working with time series data, making it simple to perform operations on time-based data.
6. **Reading and Writing Data:**
   - Pandas can read data from various file formats, including CSV, Excel, SQL databases, JSON, and more, using functions like `read_csv()`, `read_excel()`, and `read_sql()`.
   - It can also write DataFrames back to these formats using functions like `to_csv()` and `to_excel()`.

In summary, Pandas is a powerful Python library for data manipulation and analysis that simplifies working with structured data, making it a valuable tool for anyone dealing with data in Python. Its flexibility and extensive functionality make it an essential part of the data science toolkit.

Introduction to Pandas Series in Python



A pandas Series is a one-dimensional data structure in the pandas library, which is a popular Python library for data manipulation and analysis. It can be thought of as a labeled array or a column in a spreadsheet or a single column in a SQL table. Each element in a Series is associated with a label or index, allowing for easy and efficient data manipulation and analysis.

Key features of a pandas Series include:

1. Homogeneous Data: All elements in a Series must be of the same data type, such as integers, floats, strings, or even more complex data structures like other Series or dataframes.

2. Labels: Each element in a Series is associated with a label or an index. You can think of the index as a unique identifier for each element in the Series. The labels can be integers, strings, or other types.

3. Powerful Data Operations: Pandas Series allows you to perform various operations like filtering, slicing, mathematical operations, and more on the data elements efficiently.

You can create a pandas Series from various data sources, such as Python lists, NumPy arrays, or dictionaries. Here's an example of creating a Series from a Python list:

In [ ]:
```
import pandas as pd

data = [1, 2, 3, 4, 5]
series = pd.Series(data)

print(series)
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

You can access data in a Series using its labels or positions, and you can perform various operations like filtering, aggregating, and applying functions to the data. Series are often used as building blocks for more complex data structures like pandas DataFrames, which are essentially collections of Series organized in a tabular structure.

Importing Pandas:

Importing the pandas library is done using the `import pandas as pd` statement. It's considered best practice to also import numpy when working with pandas since numpy is often used internally for data manipulation.

In [ ]:
```
import numpy as np # it is a best prctise with pandas always import numpy as well
import pandas as pd
```

Series From List:

- You can create a Series from a list, which can contain various data types. For example, you can create a Series from a list of strings, integers, or any other data type.

In [ ]:
```
# string

country = ['India', 'Pakistan', 'USA']

pd.Series(country)
```

Out[ ]:
```
0       India
1    Pakistan
2         USA
dtype: object
```

In [ ]:
```
# integers
runs = [67,38,35]

runs_series = pd.Series(runs)
print(runs_series)
```

```
0    67
1    38
2    35
dtype: int64
```

Series From List (Custom Index):

- You can specify custom index labels for a Series when creating it from a list. This allows you to associate each element with a specific label.

In [ ]:
```
# custom Index
marks = [67,54,89,100]
subjects = ['Math','English','SocialScience','Marathi']

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

Out[ ]:
```
Math              67
English           54
SocialScience     89
Marathi          100
dtype: int64
```

Series From List (Setting a Name):

- You can set a name for the Series when creating it. This name can be used to label the Series, making it more descriptive

In [ ]:
```
# setting a name of series

Marks = pd.Series(marks,index=subjects,name="Exam_Marks")
Marks
```

Out[ ]:
```
Math              67
English           54
SocialScience     89
Marathi          100
Name: Exam_Marks, dtype: int64
```

Series From Dictionary:

- A pandas Series can also be created from a dictionary. The keys of the dictionary become the index labels, and the values become the data elements in the Series.

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

marks_series = pd.Series(marks,name='marks')
marks_series
```

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

In [ ]:
```
marks
```

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

Series Attributes (Size):

- The `.size` attribute of a Series returns the number of elements in the Series.

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

Out[ ]:
```
4
```

Series Attributes (Data Type - dtype):

- The `.dtype` attribute of a Series returns the data type of the elements in the Series. In this case, it's 'int64', indicating integers.

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

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

Series Attributes (Name):

- The `.name` attribute of a Series allows you to set and retrieve the name of the Series. It can make the Series more descriptive.

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

Out[ ]:
```
'marks'
```

Series Attributes (is_unique):

- The `.is_unique` attribute checks if all elements in the Series are unique. If there are duplicates, it returns `False`.

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

pd.Series([1,1,2,3,4,5]).is_unique
```

Out[ ]:
```
False
```

Series Attributes (Index):

- The `.index` attribute returns the index labels associated with the Series, which are stored in an Index object.

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

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

Series Attributes (Values):

- The `.values` attribute returns the actual data elements of the Series, stored in a NumPy array.

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

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

These features and attributes make pandas Series a versatile and powerful data structure for working with one-dimensional data, making it a fundamental tool for data manipulation and analysis in Python.

Working with Series in Pandas: Cricket and Bollywood Datasets

Importing Libraries:

- The code starts by importing the necessary Python libraries, including pandas (as 'pd') and NumPy (as 'np').

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

In [ ]:
```
import warnings
warnings.filterwarnings("ignore")
```

Reading a CSV File with Pandas:

- The code uses the `pd.read_csv()` function to read a CSV file located at '/content/kohli_ipl.csv'. It sets the 'match_no' column as the index and uses the `squeeze=True` parameter to read the data as a pandas Series.

In [ ]:
```
vk = pd.read_csv('/content/kohli_ipl.csv',index_col='match_no',squeeze=True)
vk
```

Out[ ]:
```
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
```

Displaying the 'vk' Series:

- The code then displays the 'vk' Series, which represents data related to cricket matches with the match number as the index and the number of runs scored by a player.

Reading Another CSV File:

- The code reads another CSV file located at '/content/bollywood.csv' and assigns it to the 'movies' Series. It sets the 'movie' column as the index.

In [ ]:
```
movies = pd.read_csv('/content/bollywood.csv',index_col='movie',squeeze=True)
movies
```

Out[ ]:
```
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
```

Displaying the 'movies' Series:

- The 'movies' Series is displayed, containing data related to Bollywood movies with the movie title as the index and the lead actor's name as the data.

Series methods

In [ ]:
```
# head and tail

# `head()`: Displays the first 5 rows of the 'vk' Series.
vk.head()
```

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

In [ ]:
```
# head(10): Displays the first 10 rows of the 'vk' Series.
vk.head(10)
```

Out[ ]:
```
match_no
1      1
2     23
3     13
4     12
5      1
6      9
7     34
8      0
9     21
10     3
Name: runs, dtype: int64
```

In [ ]:
```
# tail -> last 5 rows
vk.tail()
```

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

In [ ]:
```
# `sample()`: Retrieves a random sample from the 'movies' Series.
movies.sample()
```

Out[ ]:
```
movie
Dhund (2003 film)    Amar Upadhyaya
Name: lead, dtype: object
```

In [ ]:
```
# `sample(5)`: Retrieves 5 random samples from the 'movies' Series.
movies.sample(5)
```

Out[ ]:
```
movie
Halla Bol                      Ajay Devgn
Shaadi No. 1                 Fardeen Khan
Karma Aur Holi             Rati Agnihotri
Patiala House (film)         Rishi Kapoor
Chaalis Chauraasi        Naseeruddin Shah
Name: lead, dtype: object
```

In [ ]:
```
# `value_counts()`: Counts the number of occurrences of each lead actor in the 'mov
movies.value_counts()
```

Out[ ]:
```
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 [ ]:
```
# `sort_values()`: Sorts the 'vk' Series in ascending order.
vk.sort_values()
```

Out[ ]:
```
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 [ ]:
```
# `sort_values(ascending=False)`: Sorts the 'vk' Series in descending order.
vk.sort_values(ascending=False).head(1).values[0]
```

Out[ ]:
```
113
```

In [ ]:
```
#`sort_index(ascending=False)`: Sorts the 'movies' Series by index in descending or
vk.sort_values(ascending=False)
```

Out[ ]:
```
match_no
128    113
126    109
123    108
164    100
120    100
       ...
93       0
211      0
130      0
8        0
135      0
Name: runs, Length: 215, dtype: int64
```

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

Out[ ]:
```
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
```

In [ ]:
```
# inplace -> for permanant change
movies.sort_index(ascending=False,inplace=True)
```

In-Place Sorting:

- The code shows how to perform in-place sorting by using the `inplace=True` argument with the `sort_index()` method for the 'movies' Series.

In [ ]:
```
movies
```

Out[ ]:
```
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

In [ ]:
```
# `count()`: Counts the number of non-null elements in the 'vk' Series
vk.count()
```

Out[ ]:
```
215
```

In [ ]:
```
# `sum()`: Calculates the total runs scored by the player in the 'vk' Series
vk.sum()
```

Out[ ]:
```
6634
```

In [ ]:
```
# `mean()`: Calculates the mean (average) value of the 'vk' Series.
# `median()`: Calculates the median (middle) value of the 'vk' Series.
# `mode()`: Calculates the mode (most frequent value) of the 'vk' Series.'''

print(vk.mean())
print('---------')
print(vk.median())
print('---------')
print(movies.mode())
print('---------')
print(vk.std())
print('---------')
print(vk.var())
```

```
30.855813953488372
---------
24.0
---------
0    Akshay Kumar
Name: lead, dtype: object
---------
26.22980132830278
---------
688.0024777222343
```

In [ ]:
```
# min/max
vk.min()
```

Out[ ]:
```
0
```

In [ ]:
```
vk.max()
```

Out[ ]:
```
113
```

In [ ]:
```
# `describe()`: Provides summary statistics for the 'vk' Series, including count, m
vk.describe()
```

Out[ ]:
```
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
```

Some Important Series Methods

1. astype
2. between
3. clip
4. drop_duplicates
5. isnull
6. dropna
7. fillna
8. isin
9. apply
10. copy
11. plot

In [ ]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
# import dataset
vk = pd.read_csv('/content/kohli_ipl.csv',index_col='match_no',squeeze=True)
vk
```

Out[ ]:
```
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 [ ]:
```
movies = pd.read_csv('/content/bollywood.csv',index_col='movie',squeeze=True)
movies
```

Out[ ]:
```
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
```

astype:

- The `astype` method is used to change the data type of the elements in a Pandas Series. In your example, you used it to change the data type of the 'vk' Series from 'int64' to 'int16', which can reduce memory usage if you're dealing with large datasets.

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

Out[ ]:
```
3456
```

In [ ]:
```
vk.astype('int16')
```

Out[ ]:
```
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 [ ]:
```
sys.getsizeof(vk.astype('int16'))
```

Out[ ]:
```
2166
```

between:

- The `between` method is used to filter a Series to include only elements that fall within a specified range. In your example, you used it to filter the 'vk' Series to include only values between 51 and 99.

In [ ]:
```
# between
vk.between(51,99)
```

Out[ ]:
```
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
```

In [ ]:
```
# between
vk[vk.between(51,99)].size
```

Out[ ]:
```
43
```

clip:

- The `clip` method is used to limit the values in a Series to a specified range. It replaces values that are below the lower bound with the lower bound and values above the upper bound with the upper bound. This can be useful for handling outliers or ensuring data falls within a certain range.

In [ ]:
```
# clip
vk
```

Out[ ]:
```
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 [ ]:
```
vk.clip(50,80)
```

Out[ ]:
```
match_no
1      50
2      50
3      50
4      50
5      50
       ..
211    50
212    50
213    73
214    50
215    50
Name: runs, Length: 215, dtype: int64
```

drop_duplicates:

- The `drop_duplicates` method is used to remove duplicate values from a Series. It returns a new Series with only the unique values. In your example, you used it with the 'temp' Series to remove duplicate values.

In [ ]:
```
# drop_duplicates
temp = pd.Series([1,1,2,2,3,3,4,4])
temp
```

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

In [ ]:
```
temp.duplicated().sum()
```

Out[ ]:
```
4
```

In [ ]:
```
temp.drop_duplicates()
```

Out[ ]:
```
0    1
2    2
4    3
6    4
dtype: int64
```

isnull:

- The `isnull` method is used to check for missing or NaN (Not-a-Number) values in a Series. It returns a Boolean Series where 'True' indicates missing values and 'False' indicates non-missing values. In your example, you used it to find missing values in the 'temp' Series.

In [ ]:
```
# isnull
temp = pd.Series([1,2,3,np.nan,5,6,np.nan,8,np.nan,10])
temp
```

Out[ ]:
```
0     1.0
1     2.0
2     3.0
3     NaN
4     5.0
5     6.0
6     NaN
7     8.0
8     NaN
9    10.0
dtype: float64
```

In [ ]:
```
temp.isnull().sum()
```

Out[ ]:
```
3
```

dropna:

- The `dropna` method is used to remove missing values from a Series. It returns a new Series with the missing values removed. In your example, you used it to remove missing values from the 'temp' Series.

In [ ]:
```
# dropna
temp.dropna()
```

Out[ ]:
```
0     1.0
1     2.0
2     3.0
4     5.0
5     6.0
7     8.0
9    10.0
dtype: float64
```

fillna:

- The `fillna` method is used to fill missing values in a Series with specified values. It can be used to replace missing data with a specific value, such as the mean of the non-

missing values. In your example, you filled missing values in the 'temp' Series with the mean of the non-missing values.

In [ ]:
```
# fillna
temp.fillna(temp.mean())
```

Out[ ]:
```
0     1.0
1     2.0
2     3.0
3     5.0
4     5.0
5     6.0
6     5.0
7     8.0
8     5.0
9    10.0
dtype: float64
```

isin:

- The `isin` method is used to filter a Series to include only elements that match a list of values. In your example, you used it to filter the 'vk' Series to include only values that match either 49 or 99.

In [ ]:
```
# isin

vk[vk.isin([49,99])]
```

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

apply:

- The `apply` method is used to apply a function to each element of a Series. In your example, you applied a lambda function to the 'movies' Series to extract the first word of each element and convert it to uppercase.

In [ ]:
```
# apply
movies
```

Out[ ]:
```
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 [ ]:
```
movies.apply(lambda x:x.split()[0].upper())
```

Out[ ]:
```
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                     SHAH
Aankhen (2002 film)                     AMITABH
Saathiya (film)                            VIVEK
Company (film)                             AJAY
Awara Paagal Deewana                     AKSHAY
Name: lead, Length: 1500, dtype: object
```

copy:

- The `copy` method is used to create a copy of a Series. This copy is separate from the original Series, and any modifications to the copy won't affect the original Series. In your example, you created a copy of the 'vk' Series using the 'copy' method and modified the copy without affecting the original Series.

In [ ]:
```
# copy

vk
```

Out[ ]:
```
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 [ ]:
```
new = vk.head().copy()
```

In [ ]:
```
new
```

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

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

In [ ]:
```
new
```

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

In [ ]:
```
vk.head()
```

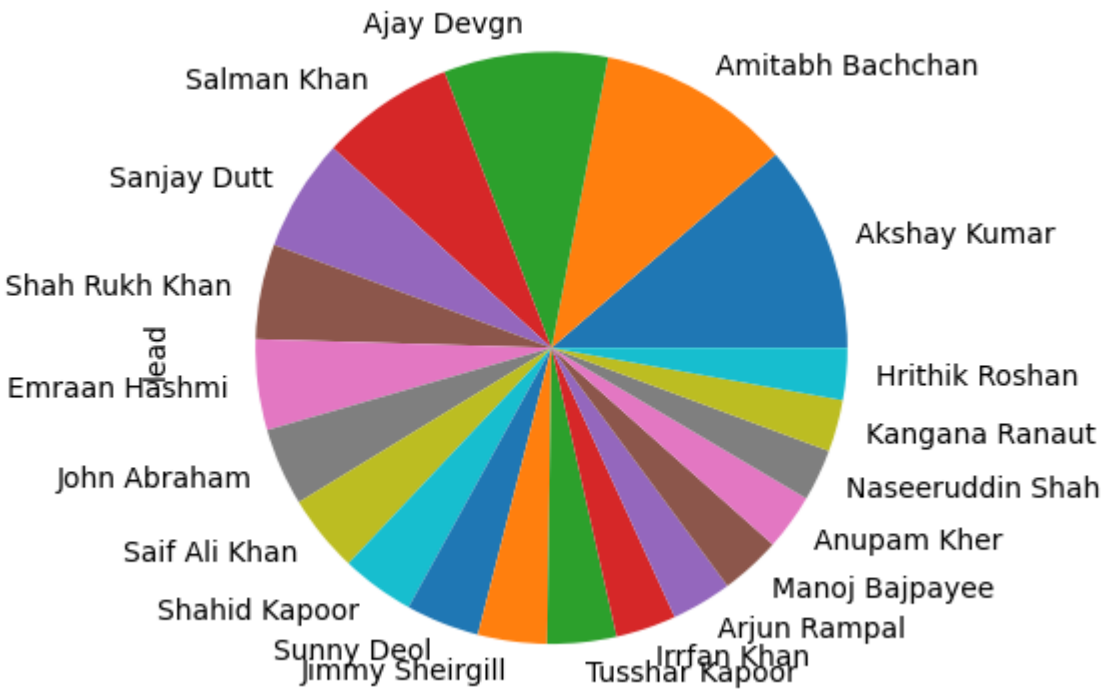
Out[ ]:
```
match_no
1     1
2    23
3    13
4    12
5     1
Name: runs, dtype: int64
```

plot:

- The `plot` method is used to create visualizations of data in a Series. You can specify the type of plot (e.g., 'line', 'bar', 'pie') and customize various plot attributes. In your example, you used it to create a pie chart of the top 20 most common values in the 'movies' Series.

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

Out[ ]:
```
<Axes: ylabel='lead'>
```



Introduction to DataFrames in Pandas

In [ ]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

Creating DataFrames:

- DataFrames can be created in various ways. You demonstrated creating DataFrames using lists and dictionaries. Lists represent rows, and dictionaries represent columns.

Using List

In [ ]:
```
# using lists

student_data = [
   [100,80,10],
    [90,70,7],
    [120,100,14],
    [80,50,2]
]

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

Out[ ]:

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

Using Dictionary

In [ ]:
```
# using dictionary

student_dict = {
    'name':['nitish','ankit','rupesh','rishabh','amit','ankita'],
    'iq':[100,90,120,80,0,0],
    'marks':[80,70,100,50,0,0],
    'package':[10,7,14,2,0,0]
}

students = pd.DataFrame(student_dict)
students
```

Out[ ]:

	name	iq	marks	package
0	nitish	100	80	10
1	ankit	90	70	7
2	rupesh	120	100	14
3	rishabh	80	50	2
4	amit	0	0	0
5	ankita	0	0	0

Reading Data from CSV:

- You can also create DataFrames by reading data from CSV files using the `pd.read_csv()` function.

In [ ]:
```
# using read_scv

ipl = pd.read_csv('ipl-matches.csv')
ipl
```

Out[ ]:

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

950 rows × 20 columns

In [ ]:
```
movies = pd.read_csv('movies.csv')
```

In [ ]:
```
movies
```

Out[ ]:

	title_x	imdb_id	poster_path	
0	Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/
1	Battalion 609	tt9472208	NaN	https://en.wikipedia.
2	The Accidental Prime Minister (film)	tt6986710	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/v
3	Why Cheat India	tt8108208	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org
4	Evening Shadows	tt6028796	NaN	https://en.wikipedia.org/
...	...	...	...	
1624	Tera Mera Saath Rahen	tt0301250	https://upload.wikimedia.org/wikipedia/en/2/2b...	https://en.wikipedia.org/w
1625	Yeh Zindagi Ka Safar	tt0298607	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/w
1626	Sabse Bada Sukh	tt0069204	NaN	https://en.wikipedia.org/
1627	Daaka	tt10833860	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wi
1628	Humsafar	tt2403201	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipe

1629 rows × 18 columns

Attributes of DataFrames:

- DataFrames have several attributes that provide information about their structure and content:

In [ ]:
```
#  `shape` returns the number of rows and columns in the DataFrame.
ipl.shape
```

Out[ ]:
```
(950, 20)
```

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

Out[ ]:
```
(1629, 18)
```

In [ ]:
```
# `dtypes` displays the data types of each column.
ipl.dtypes
```

Out[ ]:
```
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 [ ]:
```
movies.dtypes
```

Out[ ]:
```
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 [ ]:
```
# index` shows the row index information.
movies.index
```

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

In [ ]:
```
# `columns` provides the column names.
movies.columns
```

Out[ ]:
```
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 [ ]:
```
ipl.columns
```

Out[ ]:
```
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 [ ]:
```
students.columns
```

Out[ ]:
```
Index(['name', 'iq', 'marks', 'package'], dtype='object')
```

In [ ]:
```
# `values` gives a NumPy array of the data in the DataFrame.
students.values
```

Out[ ]:
```
array([['nitish', 100, 80, 10],
       ['ankit', 90, 70, 7],
       ['rupesh', 120, 100, 14],
       ['rishabh', 80, 50, 2],
       ['amit', 0, 0, 0],
       ['ankita', 0, 0, 0]], dtype=object)
```

Viewing Data:

- To view the data in a DataFrame, you can use methods like `head()`, `tail()`, and `sample()` to see the first few rows, last few rows, or random sample rows, respectively.

In [ ]:
```
# head and tail
movies.head()
```

Out[ ]:

	title_x	imdb_id	poster_path	
0	Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/U
1	Battalion 609	tt9472208	NaN	https://en.wikipedia.org/w
2	The Accidental Prime Minister (film)	tt6986710	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/T
3	Why Cheat India	tt8108208	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/
4	Evening Shadows	tt6028796	NaN	https://en.wikipedia.org/wiki/E

In [ ]:
```
# sample
ipl.sample(5)
```

Out[ ]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	To
455	829781	Kolkata	2015-05-04	2015	38	Kolkata Knight Riders	Sunrisers Hyderabad	Eden Gardens	H
52	1304068	Mumbai	2022-04-12	2022	22	Chennai Super Kings	Royal Challengers Bangalore	Dr DY Patil Sports Academy, Mumbai	Ch
686	548322	Pune	2012-04-14	2012	16	Pune Warriors	Chennai Super Kings	Subrata Roy Sahara Stadium	Su
681	548327	Bangalore	2012-04-17	2012	21	Royal Challengers Bangalore	Pune Warriors	M Chinnaswamy Stadium	
752	501221	Mumbai	2011-04-22	2011	25	Mumbai Indians	Chennai Super Kings	Wankhede Stadium	Su

Information about DataFrames:

- You can obtain information about a DataFrame using the `info()` method, which provides data types, non-null counts, and memory usage.

In [ ]:
```
# 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
```

- For numerical columns, you can use the `describe()` method to get statistics like count, mean, standard deviation, min, max, and quartiles.

In [ ]:
```
# describe -> only give on numerical columns
movies.describe()
```

Out[ ]:

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

Out[ ]:

	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

Missing Data:

- The `isnull()` function helps check for missing data (NaN values) in a DataFrame.
- The `sum()` function can be used to count the missing values in each column.

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

Out[ ]:
```
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
```

Duplicated Rows:

- You can check for duplicate rows in a DataFrame using the `duplicated()` function. It returns the number of duplicated rows.

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

Out[ ]:
```
0
```

In [ ]:
```
students.duplicated().sum()
```

Out[ ]:
```
0
```

Column Renaming:

- You can rename columns in a DataFrame using the `rename()` function. It can be performed temporarily or with permanent changes if you set the `inplace` parameter to `True`.

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

Out[ ]:

	name	iq	marks	package
0	nitish	100	80	10
1	ankit	90	70	7
2	rupesh	120	100	14
3	rishabh	80	50	2
4	amit	0	0	0
5	ankita	0	0	0

In [ ]:
```
students.rename(columns={'package':'package_lpa'})
```

Out[ ]:

	name	iq	marks	package_lpa
0	nitish	100	80	10
1	ankit	90	70	7
2	rupesh	120	100	14
3	rishabh	80	50	2
4	amit	0	0	0
5	ankita	0	0	0

In [ ]:
```
# if you want permanant changes
students.rename(columns={'package':'package_lpa'},inplace=True)
```

In [ ]:
```
students
```

Out[ ]:

	name	iq	marks	package_lpa
0	nitish	100	80	10
1	ankit	90	70	7
2	rupesh	120	100	14
3	rishabh	80	50	2
4	amit	0	0	0
5	ankita	0	0	0

These File provide a basic understanding of working with DataFrames in pandas, including creating, reading, exploring, and modifying them. It's important to note that these are fundamental operations, and pandas offers many more capabilities for data manipulation and analysis.

Pandas DataFrame Operations

In [ ]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

Certainly, here are some notes on the code you provided:

1.
2.
3.
4.

These are fundamental operations when working with Pandas DataFrames. They are useful for data manipulation and analysis, allowing you to extract specific information from your data.

Creating DataFrames:

The code demonstrates how to create Pandas DataFrames using different methods:

- Lists: You can create a DataFrame from a list of lists, where each inner list represents a row.
- Dictionaries: You can create a DataFrame from a dictionary where keys become column names.
- Reading from CSV: DataFrames can be created by reading data from a CSV file.

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

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

Out[ ]:

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

In [ ]:
```
# using dicts

student_dict = {
    'name':['nitish','ankit','rupesh','rishabh','amit','ankita'],
    'iq':[100,90,120,80,0,0],
```

```
    'marks':[80,70,100,50,0,0],
    'package':[10,7,14,2,0,0]
}

students = pd.DataFrame(student_dict)
students.set_index('name',inplace=True)
students
```

Out[ ]:

name	iq	marks	package
nitish	100	80	10
ankit	90	70	7
rupesh	120	100	14
rishabh	80	50	2
amit	0	0	0
ankita	0	0	0

In [ ]:

```
# using read_csv
movies = pd.read_csv('movies.csv')
movies
```

Out[ ]:

	title_x	imdb_id	poster_path	
0	Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/
1	Battalion 609	tt9472208	NaN	https://en.wikipedia.
2	The Accidental Prime Minister (film)	tt6986710	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/v
3	Why Cheat India	tt8108208	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org
4	Evening Shadows	tt6028796	NaN	https://en.wikipedia.org/
...	...	...	...	
1624	Tera Mera Saath Rahen	tt0301250	https://upload.wikimedia.org/wikipedia/en/2/2b...	https://en.wikipedia.org/w
1625	Yeh Zindagi Ka Safar	tt0298607	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/w
1626	Sabse Bada Sukh	tt0069204	NaN	https://en.wikipedia.org/
1627	Daaka	tt10833860	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wi
1628	Humsafar	tt2403201	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipe

1629 rows × 18 columns

In [ ]:
```
ipl = pd.read_csv('ipl-matches.csv')
ipl
```

Out[ ]:

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

950 rows × 20 columns

Selecting cols from a DataFrame

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

Out[ ]:
```
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
```

You can select specific columns from a DataFrame using square brackets. For instance, in the code, the `movies['title_x']` expression selects the 'title_x' column from the 'movies' DataFrame.

In [ ]:
```
ipl['Venue']
```

Out[ ]:
```
0               Narendra Modi Stadium, Ahmedabad
1               Narendra Modi Stadium, Ahmedabad
2                         Eden Gardens, Kolkata
3                         Eden Gardens, Kolkata
4                      Wankhede Stadium, Mumbai
                          ...
945                               Eden Gardens
946                            Wankhede Stadium
947                            Feroz Shah Kotla
948    Punjab Cricket Association Stadium, Mohali
949                          M Chinnaswamy Stadium
Name: Venue, Length: 950, dtype: object
```

In [ ]:
```
students['package']
```

Out[ ]:
```
name
nitish     10
ankit       7
rupesh     14
rishabh     2
amit        0
ankita      0
Name: package, dtype: int64
```

In [ ]:
```
# multiple cols
movies[['year_of_release','actors','title_x']]
```

Out[ ]:

	year_of_release	actors	title_x
0	2019	Vicky Kaushal|Paresh Rawal|Mohit Raina|Yami Ga...	Uri: The Surgical Strike
1	2019	Vicky Ahuja|Shoaib Ibrahim|Shrikant Kamat|Elen...	Battalion 609
2	2019	Anupam Kher|Akshaye Khanna|Aahana Kumra|Atul S...	The Accidental Prime Minister (film)
3	2019	Emraan Hashmi|Shreya Dhanwanthary|Snighdadeep ...	Why Cheat India
4	2018	Mona Ambegaonkar|Ananth Narayan Mahadevan|Deva...	Evening Shadows
...	...	...	...
1624	2001	Ajay Devgn|Sonali Bendre|Namrata Shirodkar|Pre...	Tera Mera Saath Rahen
1625	2001	Ameesha Patel|Jimmy Sheirgill|Nafisa Ali|Gulsh...	Yeh Zindagi Ka Safar
1626	2018	Vijay Arora|Asrani|Rajni Bala|Kumud Damle|Utpa...	Sabse Bada Sukh
1627	2019	Gippy Grewal|Zareen Khan|	Daaka
1628	2011	Fawad Khan|	Humsafar

1629 rows × 3 columns

In [ ]:
```
ipl[['Team1','Team2','WinningTeam']]
```

Out[ ]:

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

950 rows × 3 columns

Selecting Rows: You can select rows using `iloc` or `loc` methods:

- `iloc` - uses integer-based indexing, and you can select rows by their index positions.
- `loc` - uses label-based indexing, and you can select rows by their index labels.

In [ ]:
```
# single row
movies.iloc[5]
```

Out[ ]:
```
title_x                                                 Soni (film)
imdb_id                                                   tt6078866
poster_path          https://upload.wikimedia.org/wikipedia/en/thum...
wiki_link                   https://en.wikipedia.org/wiki/Soni_(film)
title_y                                                         Soni
original_title                                                  Soni
is_adult                                                           0
year_of_release                                                 2018
runtime                                                           97
genres                                                         Drama
imdb_rating                                                      7.2
imdb_votes                                                      1595
story                Soni  a young policewoman in Delhi  and her su...
summary              While fighting crimes against women in Delhi  ...
tagline                                                          NaN
actors               Geetika Vidya Ohlyan|Saloni Batra|Vikas Shukla...
wins_nominations                               3 wins & 5 nominations
release_date                                    18 January 2019 (USA)
Name: 5, dtype: object
```

In [ ]:
```
# multiple row
movies.iloc[:5]
```

Out[ ]:

	title_x	imdb_id	poster_path	
0	Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/U
1	Battalion 609	tt9472208	NaN	https://en.wikipedia.org/w
2	The Accidental Prime Minister (film)	tt6986710	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/T
3	Why Cheat India	tt8108208	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/
4	Evening Shadows	tt6028796	NaN	https://en.wikipedia.org/wiki/E

In [ ]:
```
# fancy indexing
movies.iloc[[0,4,5]]
```

Out[ ]:

	title_x	imdb_id	poster_path	
0	Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Uri
4	Evening Shadows	tt6028796	NaN	https://en.wikipedia.org/wiki/Eve
5	Soni (film)	tt6078866	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/

In [ ]:
```
# loc
students
```

Out[ ]:

name	iq	marks	package
nitish	100	80	10
ankit	90	70	7
rupesh	120	100	14
rishabh	80	50	2
amit	0	0	0
ankita	0	0	0

In [ ]:
```
students.loc['nitish']
```

Out[ ]:
```
iq         100
marks       80
package     10
Name: nitish, dtype: int64
```

In [ ]:
```
students.loc['nitish':'rishabh']
```

Out[ ]:

name	iq	marks	package
nitish	100	80	10
ankit	90	70	7
rupesh	120	100	14
rishabh	80	50	2

In [ ]:
```
students.loc[['nitish','ankita','rupesh']]
```

Out[ ]:

name	iq	marks	package
nitish	100	80	10
ankita	0	0	0
rupesh	120	100	14

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

Out[ ]:

name	iq	marks	package
nitish	100	80	10
rishabh	80	50	2
amit	0	0	0

Selecting both rows and cols

You can use the `iloc` and `loc` methods to select both rows and columns

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

Out[ ]:

	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...

`movies.iloc[0:3, 0:3]` selects the first three rows and first three columns of the 'movies' DataFrame.

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

Out[ ]:

	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...

These are fundamental operations when working with Pandas DataFrames. They are useful for data manipulation and analysis, allowing you to extract specific information from your data.

Exploring IPL and Movies Datasets with Pandas

Filtering a DataFrame

In [ ]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
ipl = pd.read_csv('ipl-matches.csv')
```

In [ ]:
```
ipl.head(3)
```

Out[ ]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossW
0	1312200	Ahmedabad	2022-05-29	2022	Final	Rajasthan Royals	Gujarat Titans	Narendra Modi Stadium, Ahmedabad	Raja R
1	1312199	Ahmedabad	2022-05-27	2022	Qualifier 2	Royal Challengers Bangalore	Rajasthan Royals	Narendra Modi Stadium, Ahmedabad	Raja R
2	1312198	Kolkata	2022-05-25	2022	Eliminator	Royal Challengers Bangalore	Lucknow Super Giants	Eden Gardens, Kolkata	Luc S G

Find all the final winners in ipl

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

Out[ ]:

	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 [ ]:
```
ipl[ipl['MatchNumber'] == 'Final'][['Season','WinningTeam']]
```

Out[ ]:

	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

How many super over finishes have occured ?

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

Out[ ]:
```
14
```

How many matches has csk won in kolkata?

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

Out[ ]:
```
5
```

Toss winner is match winner in percentage

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

Out[ ]:
```
51.473684210526315
```

Movies Dataset

In [ ]:
```
movies = pd.read_csv('movies.csv')
```

In [ ]:
```
movies.head(3)
```

Out[ ]:

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

Movies with rating higher than 8 and votes>10000

In [ ]:
```
movies[(movies['imdb_rating'] > 8.5) & (movies['imdb_votes'] > 10000)].shape[0]
```

Out[ ]:
```
0
```

Action movies with rating higher than 7.5

In [ ]:
```
mask1 = movies['genres'].str.contains('Action')
mask2 = movies['imdb_rating'] > 7.5

data = movies[mask1 & mask2]

data['title_x']
```

Out[ ]:
```
0            Uri: The Surgical Strike
41               Family of Thakurganj
84                           Mukkabaaz
106                             Raazi
110     Parmanu: The Story of Pokhran
112           Bhavesh Joshi Superhero
169                  The Ghazi Attack
219                  Raag Desh (film)
258                     Irudhi Suttru
280                         Laal Rang
297                       Udta Punjab
354                     Dangal (film)
362                 Bajrangi Bhaijaan
365            Baby (2015 Hindi film)
393        Detective Byomkesh Bakshy!
449                  Titli (2014 film)
536                      Haider (film)
589                       Vishwaroopam
625                        Madras Cafe
668            Paan Singh Tomar (film)
693                 Gangs of Wasseypur
694       Gangs of Wasseypur – Part 2
982                       Jodhaa Akbar
1039                 1971 (2007 film)
1058         Black Friday (2007 film)
1188                Omkara (2006 film)
1293                Sarkar (2005 film)
1294                             Sehar
1361                   Lakshya (film)
1432                        Gangaajal
1495                   Company (film)
1554       The Legend of Bhagat Singh
1607           Nayak (2001 Hindi film)
Name: title_x, dtype: object
```

Important Pandas Dataframe Methods

- Value_counts
- sort_values
- rank
- sort_index
- rename index -> rename
- unique and nunique
- isnull/notnull/hasnans
- dropna
- fillna
- drop_duplicates
- drop
- apply

In [ ]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

1 . value_counts

it is used in series as well as Dataframe in pandas

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

marks
```

Out[ ]:

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

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

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

In [ ]:
```
ipl = pd.read_csv('ipl-matches.csv')
```

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

Out[ ]:

	ID	City	Date	Season	MatchNumber	Team1	Team2	Venue	TossW
0	1312200	Ahmedabad	2022-05-29	2022	Final	Rajasthan Royals	Gujarat Titans	Narendra Modi Stadium, Ahmedabad	Raja R
1	1312199	Ahmedabad	2022-05-27	2022	Qualifier 2	Royal Challengers Bangalore	Rajasthan Royals	Narendra Modi Stadium, Ahmedabad	Raja R

In [ ]:
```
# find which player has won most player of the match -> in finals and qualifiers
ipl[~ipl['MatchNumber'].str.isdigit()]['Player_of_Match'].value_counts()
```

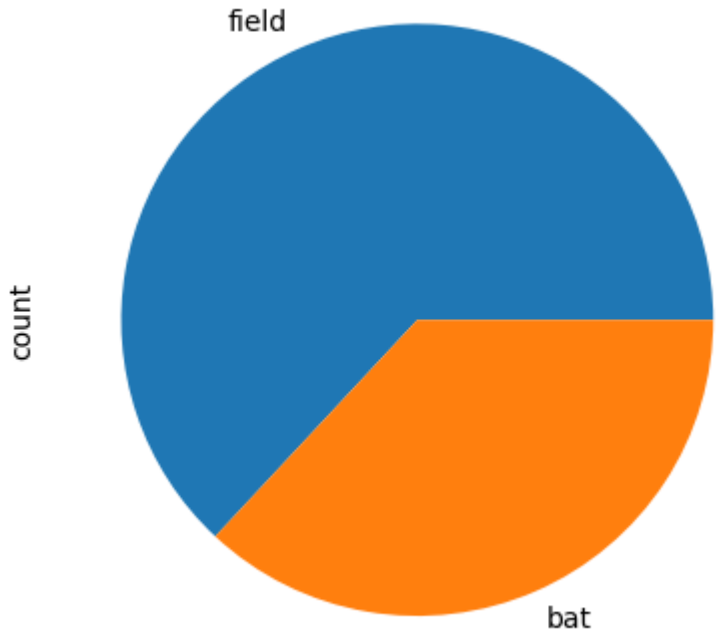
Out[ ]:
```
Player_of_Match
KA Pollard           3
F du Plessis         3
SK Raina             3
A Kumble             2
MK Pandey            2
YK Pathan            2
M Vijay              2
JJ Bumrah            2
AB de Villiers       2
SR Watson            2
HH Pandya            1
Harbhajan Singh      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: count, dtype: int64
```

In [ ]:
```
# plot Toss Pie plot
ipl['TossDecision'].value_counts()
```

Out[ ]:
```
TossDecision
field    599
bat      351
Name: count, dtype: int64
```

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

Out[ ]:
```
<Axes: ylabel='count'>
```



2. sort_values

In [ ]:
```
students = pd.DataFrame(
    {
        'name':['nitish','ankit','rupesh',np.nan,'mrityunjay',np.nan,'rishabh',np.r
        '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[ ]:

	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 [ ]:
```
students.sort_values('name')
```

Out[ ]:

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

In [ ]:
```
students.sort_values('name',na_position='first',ascending=False)
```

Out[ ]:

	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

3.sort_index(series and dataframe)

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

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

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

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

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

4.rename(dataframe) -> index

In [ ]:
```
movies = pd.read_csv('movies.csv')
```

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

Out[ ]:

	title_x	imdb_id	poster_path	
0	Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Uri:_

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

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

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

Out[ ]:

title_x	imdb	link
Uri: The Surgical Strike	tt8291224	https://upload.wikimedia.org/wikipedia/en/thum...	https://en.wikipedia.org/wiki/Uri:_Th

5. unique (Series) / nunique(series + dataframe) -> does not count nan -> dropna parameter

In [ ]:
```
marks_series.unique()
```

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

In [ ]:
```
ipl['Season'].nunique()
```

Out[ ]:
```
15
```

6. isnull(series + dataframe)

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

Out[ ]:
```
3    NaN
5    NaN
7    NaN
9    NaN
Name: name, dtype: object
```

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

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

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

Out[ ]:
```
True
```

In [ ]:
```
students
```

Out[ ]:

	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

7. dropna

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

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

In [ ]:
```
students
```

Out[ ]:

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

Out[ ]:

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

Out[ ]:

	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

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

Out[ ]:

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

Out[ ]:

	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

8. fillna(series + dataframe)

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

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

Out[ ]:
```
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 [ ]:
```
students['name'].fillna(method='bfill')
```

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

9. drop_duplicates(series + dataframe) -> works like and -> duplicated()

In [ ]:
```
# Create a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'Alice', 'David'],
        'Age': [25, 30, 22, 25, 28],
        'City': ['New York', 'San Francisco', 'Los Angeles', 'New York', 'Chicago']}
df = pd.DataFrame(data)

# Display the original DataFrame
print("Original DataFrame:")
print(df)

# Use the drop_duplicates method to remove duplicate rows based on all columns
df_no_duplicates = df.drop_duplicates()

# Display the DataFrame without duplicates
print("\nDataFrame after dropping duplicates:")
print(df_no_duplicates)
```

```
Original DataFrame:
      Name  Age           City
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   22    Los Angeles
3    Alice   25       New York
4    David   28        Chicago

DataFrame after dropping duplicates:
      Name  Age           City
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   22    Los Angeles
4    David   28        Chicago
```

10 .drop(series + dataframe)

In [ ]:
```
students
```

Out[ ]:

	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 [ ]:
```
students.drop(columns=['branch','cgpa'],inplace=True)
```

In [ ]:
```
students
```

Out[ ]:

	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

11. apply

In [ ]:
```
points_df = 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_df
```

Out[ ]:

	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 [ ]:
```
def euclidean(row):
  pt_A = row['1st point']
  pt_B = row['2nd point']

  return ((pt_A[0] - pt_B[0])**2 + (pt_A[1] - pt_B[1])**2)**0.5
```

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

Out[ ]:

	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 Object in Pandas(part-1): Groupby Foundation

In pandas, a `GroupBy` object is a crucial part of the data manipulation process, specifically for data aggregation and transformation. It is a result of the `groupby()` method applied to a pandas DataFrame, which allows you to group the data in the DataFrame based on one or more columns.

When you apply `groupby()` to a DataFrame, it creates a `GroupBy` object, which acts as a kind of intermediate step before applying aggregation functions or other operations to the grouped data. This intermediate step helps you perform operations on subsets of data based on the grouping criteria. Some common aggregation functions you can apply to a `GroupBy` object include `sum()`, `mean()`, `count()`, `max()`, `min()`, and more.

Here's a basic example of how you can create a `GroupBy` object and perform aggregation with it:

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

# Create a sample DataFrame
data = {
    'Category': ['A', 'B', 'A', 'B', 'A'],
    'Value': [10, 20, 15, 25, 30]
}

df = pd.DataFrame(data)

# Group the data by the 'Category' column
grouped = df.groupby('Category')
```

In [ ]:
```
# Calculate the sum of 'Value' for each group
sum_values = grouped['Value'].sum()
```

In [ ]:
```
# Display the result
print(sum_values)
```

```
Category
A    55
B    45
Name: Value, dtype: int64
```

In this example, we group the DataFrame `df` by the 'Category' column, creating a `GroupBy` object. Then, we calculate the sum of 'Value' for each group using the `sum()` method on the `GroupBy` object, resulting in a new DataFrame or Series with the aggregated values.

Practical Use

In [ ]:
```
movies = pd.read_csv('Data\Day35\imdb-top-1000.csv')
```

In [ ]:
```
movies.head(3)
```

Out[ ]:

	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
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

Applying builtin aggregation fuctions on groupby objects

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

In [ ]:
```
genres.sum(3)
```

Out[ ]:

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 [ ]:
```
genres.mean(3)
```

Out[ ]:

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

find the top 3 genres by total earning

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

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

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

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

find the genre with highest avg IMDB rating

In [ ]:
```
movies.groupby('Genre')['IMDB_Rating'].mean().sort_values(ascending=False).head(1)
```

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

find director with most popularity

In [ ]:
```
movies.groupby('Director')['No_of_Votes'].sum().sort_values(ascending=False).head(1
```

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

find number of movies done by each actor

In [ ]:
```
movies.groupby('Star1')['Series_Title'].count().sort_values(ascending=False)
```

Out[ ]:
```
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
```

A GroupBy object is a powerful tool for performing group-wise operations on data. It enables data analysts and scientists to gain insights into their data by aggregating, filtering, and transforming information based on specific grouping criteria. These operations are essential for understanding data patterns and making informed decisions.

GroupBy Attributes and Methods in Pandas

- len - find total number of groups
- size - find items in each group
- nth item - first()/nth/last items
- get_group - vs filtering
- groups
- describe
- sample
- nunique
- agg method
- apply -> builtin function

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

movies = pd.read_csv('Data\Day35\imdb-top-1000.csv')
```

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

1. len

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

Out[ ]:
```
14
```

2. nunique

In [ ]:
```
movies['Genre'].nunique()
```

Out[ ]:
```
14
```

3. size

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

Out[ ]:
```
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
```

4. nth

In [ ]:
```
genres = movies.groupby('Genre')
# genres.first()
# genres.last()
genres.nth(6)
```

Out[ ]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_of_V
16	Star Wars: Episode V - The Empire Strikes Back	1980	124	Action	8.7	Irvin Kershner	Mark Hamill	1159
27	Se7en	1995	127	Crime	8.6	David Fincher	Morgan Freeman	1445
32	It's a Wonderful Life	1946	130	Drama	8.6	Frank Capra	James Stewart	405
66	WALL·E	2008	98	Animation	8.4	Andrew Stanton	Ben Burtt	999
83	The Great Dictator	1940	125	Comedy	8.4	Charles Chaplin	Charles Chaplin	203
102	Braveheart	1995	178	Biography	8.3	Mel Gibson	Mel Gibson	959
118	North by Northwest	1959	136	Adventure	8.3	Alfred Hitchcock	Cary Grant	299
420	Sleuth	1972	138	Mystery	8.0	Joseph L. Mankiewicz	Laurence Olivier	44
724	Get Out	2017	104	Horror	7.7	Jordan Peele	Daniel Kaluuya	492

5. get_group

In [ ]:
```
genres.get_group('Fantasy')
```

Out[ ]:

	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	3
568	Nosferatu	1922	94	Fantasy	7.9	F.W. Murnau	Max Schreck	88794	4

In [ ]:
```
# with simple but its slow
movies[movies['Genre'] == 'Fantasy']
```

Out[ ]:

	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	3
568	Nosferatu	1922	94	Fantasy	7.9	F.W. Murnau	Max Schreck	88794	4

6. describe

In [ ]:
```
genres.describe()
```

Out[ ]:

	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

7. sample

In [ ]:
```
genres.sample(2,replace=True)
```

Out[ ]:

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_
648	The Boondock Saints	1999	108	Action	7.8	Troy Duffy	Willem Dafoe	
908	Kick-Ass	2010	117	Action	7.6	Matthew Vaughn	Aaron Taylor-Johnson	
193	The Gold Rush	1925	95	Adventure	8.2	Charles Chaplin	Charles Chaplin	
361	Blood Diamond	2006	143	Adventure	8.0	Edward Zwick	Leonardo DiCaprio	
61	Coco	2017	105	Animation	8.4	Lee Unkrich	Adrian Molina	
758	Papurika	2006	90	Animation	7.7	Satoshi Kon	Megumi Hayashibara	
328	Lion	2016	118	Biography	8.0	Garth Davis	Dev Patel	
159	A Beautiful Mind	2001	135	Biography	8.2	Ron Howard	Russell Crowe	
256	Underground	1995	170	Comedy	8.1	Emir Kusturica	Predrag 'Miki' Manojlovic	
667	Night on Earth	1991	129	Comedy	7.8	Jim Jarmusch	Winona Ryder	
441	The Killing	1956	84	Crime	8.0	Stanley Kubrick	Sterling Hayden	
288	Cool Hand Luke	1967	127	Crime	8.1	Stuart Rosenberg	Paul Newman	
773	Brokeback Mountain	2005	134	Drama	7.7	Ang Lee	Jake Gyllenhaal	
515	Mulholland Dr.	2001	147	Drama	7.9	David Lynch	Naomi Watts	
698	Willy Wonka & the Chocolate Factory	1971	100	Family	7.8	Mel Stuart	Gene Wilder	
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	
456	The Maltese Falcon	1941	100	Film-Noir	8.0	John Huston	Humphrey Bogart	
456	The Maltese Falcon	1941	100	Film-Noir	8.0	John Huston	Humphrey Bogart	
932	Saw	2004	103	Horror	7.6	James Wan	Cary Elwes	

	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	No_
876	The Invisible Man	1933	71	Horror	7.7	James Whale	Claude Rains	
420	Sleuth	1972	138	Mystery	8.0	Joseph L. Mankiewicz	Laurence Olivier	
119	Vertigo	1958	128	Mystery	8.3	Alfred Hitchcock	James Stewart	
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	
12	Il buono, il brutto, il cattivo	1966	161	Western	8.8	Sergio Leone	Clint Eastwood	
691	The Outlaw Josey Wales	1976	135	Western	7.8	Clint Eastwood	Clint Eastwood	

8. nunique()

In [ ]:
```
genres.nunique()
```

Out[ ]:

Genre	Series_Title	Released_Year	Runtime	IMDB_Rating	Director	Star1	No_of_Votes	Gross
Action	172	61	78	15	123	121	172	172
Adventure	72	49	58	10	59	59	72	72
Animation	82	35	41	11	51	77	82	82
Biography	88	44	56	13	76	72	88	88
Comedy	155	72	70	11	113	133	155	155
Crime	106	56	65	14	86	85	107	107
Drama	289	83	95	14	211	250	288	287
Family	2	2	2	1	2	2	2	2
Fantasy	2	2	2	2	2	2	2	2
Film-Noir	3	3	3	3	3	3	3	3
Horror	11	11	10	8	10	11	11	11
Mystery	12	11	10	8	10	11	12	12
Thriller	1	1	1	1	1	1	1	1
Western	4	4	4	4	2	2	4	4

9. agg method

In [ ]:
```
# passing dict
genres.agg(
    {
```

```
        'Runtime':'mean',
        'IMDB_Rating':'mean',
        'No_of_Votes':'sum',
        'Gross':'sum',
        'Metascore':'min'
    }
)
```

Out[ ]:

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 [ ]:

```
genres.agg(
    {
        'Runtime':['min','mean'],
        'IMDB_Rating':'mean',
        'No_of_Votes':['sum','max'],
        'Gross':'sum',
        'Metascore':'min'
    }
)
```

Out[ ]:

	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

10. apply -> builtin function

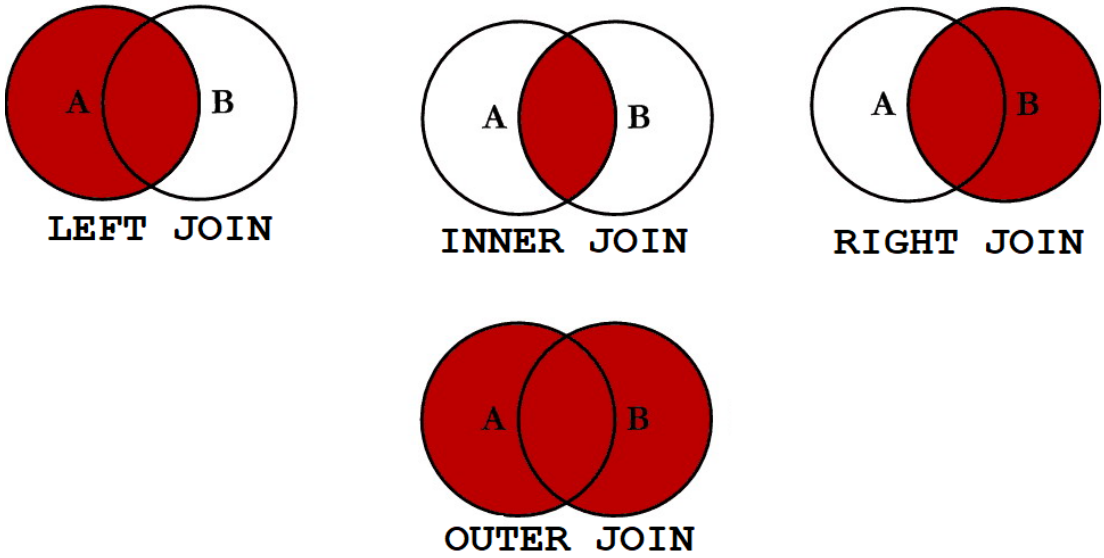
In [ ]:
```
genres.apply(min)
```

Out[ ]:

Genre	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1	N
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	

Joins In Pandas

In pandas, "joins" refer to the process of combining data from two or more DataFrames based on a common column or index. There are several types of joins available, which determine how rows are matched between DataFrames. Let's go into more detail about the different types of joins and how to perform them in pandas:



1. **Inner Join**:
   - An inner join returns only the rows that have matching keys in both DataFrames.
   - Use the `pd.merge()` function with the `how='inner'` parameter or use the `.merge()` method with the same parameter to perform an inner join.
   - Example:

```
merged_df = pd.merge(left_df, right_df, on='key', how='inner')
```

2. **Left Join (Left Outer Join)**:
   - A left join returns all the rows from the left DataFrame and the matching rows from the right DataFrame. Non-matching rows from the left DataFrame will also be included.
   - Use the `how='left'` parameter with `pd.merge()` or `.merge()` to perform a left join.
   - Example:

```
merged_df = pd.merge(left_df, right_df, on='key', how='left')
```

3. **Right Join (Right Outer Join)**:
   - A right join is the opposite of a left join. It returns all the rows from the right DataFrame and the matching rows from the left DataFrame. Non-matching rows from the right DataFrame will also be included.
   - Use the `how='right'` parameter with `pd.merge()` or `.merge()` to perform a right join.
   - Example:

```
merged_df = pd.merge(left_df, right_df, on='key', how='right')
```

4. **Full Outer Join**:

   - A full outer join returns all rows from both DataFrames, including both matching and non-matching rows.
   - Use the `how='outer'` parameter with `pd.merge()` or `.merge()` to perform a full outer join.
   - Example:

```
merged_df = pd.merge(left_df, right_df, on='key', how='outer')
```

5. **Join on Multiple Columns**:

   - You can perform joins on multiple columns by passing a list of column names to the `on` parameter.
   - Example:

```
merged_df = pd.merge(left_df, right_df, on=['key1', 'key2'],
how='inner')
```

6. **Join on Index**:

   - You can join DataFrames based on their indices using the `left_index` and `right_index` parameters set to `True`.
   - Example:

```
merged_df = pd.merge(left_df, right_df, left_index=True,
right_index=True, how='inner')
```

7. **Suffixes**:

   - If DataFrames have columns with the same name, you can specify suffixes to differentiate them in the merged DataFrame using the `suffixes` parameter.
   - Example:

```
merged_df = pd.merge(left_df, right_df, on='key', how='inner',
suffixes=('_left', '_right'))
```

Joins in pandas are a powerful way to combine and analyze data from multiple sources. It's important to understand the structure of your data and the requirements of your analysis to choose the appropriate type of join. You can also use the `.join()` method if you want to join DataFrames based on their indices or use `pd.concat()` to stack DataFrames without performing a join based on columns or indices.

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

Practical on Joins In Pandas

In [ ]:
```
# import some Dataset
courses = pd.read_csv('Data\Day37\courses.csv')


dec = pd.read_csv('Data\Day37\Dec.csv')
matches = pd.read_csv('Data\Day37\matches.csv')
delivery = pd.read_csv('Data\Day37\deliveries.csv')
```

In [ ]:
```
students = pd.read_csv('Data\Day37\student.csv')
```

In [ ]:
```
nov = pd.read_csv('Nov.csv')
```

Concat

In [ ]:
```
pd.concat([nov,dec],axis=1)
```

Out[ ]:

	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

In [ ]:
```
regs = pd.concat([nov,dec],ignore_index=True)
regs.head(2)
```

Out[ ]:

	student_id	course_id
0	23	1
1	15	5

Inner join

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

Out[ ]:

	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

left join

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

Out[ ]:

	course_id	course_name	price	student_id
0	1	python	2499	23.0
1	1	python	2499	18.0
2	1	python	2499	1.0
3	1	python	2499	15.0
4	1	python	2499	21.0

right join

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

Out[ ]:

	student_id	name	partner
0	1	Kailash Harjo	23
1	2	Esha Butala	1
2	3	Parveen Bhalla	3
3	4	Marlo Dugal	14
4	5	Kusum Bahri	6

outer join

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

Out[ ]:

	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

Pandas Merge Analysis of Student Enrollments and Course Revenues

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

In [ ]:
```
# import some Dataset
courses = pd.read_csv('Data\Day37\courses.csv')


dec = pd.read_csv('Data\Day37\Dec.csv')
matches = pd.read_csv('Data\Day37\matches.csv')
delivery = pd.read_csv('Data\Day37\deliveries.csv')
```

In [ ]:
```
students = pd.read_csv('Data\Day37\student.csv')
```

In [ ]:
```
nov = pd.read_csv('Nov.csv')
```

1. find total revenue generated

In [ ]:
```
regs = pd.concat([nov,dec],ignore_index=True)
regs.head()
```

Out[ ]:

	student_id	course_id
0	23	1
1	15	5
2	18	6
3	23	4
4	16	9

In [ ]:
```
total = regs.merge(courses,how='inner',on='course_id')['price'].sum()
total
```

Out[ ]:
```
154247
```

2. find month by month revenue

In [ ]:
```
temp_df = pd.concat([nov,dec],keys=['Nov','Dec']).reset_index()
temp_df.merge(courses,on='course_id').groupby('level_0')['price'].sum()
```

Out[ ]:
```
level_0
Dec    65072
Nov    89175
Name: price, dtype: int64
```

3. Print the registration table

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

Out[ ]:

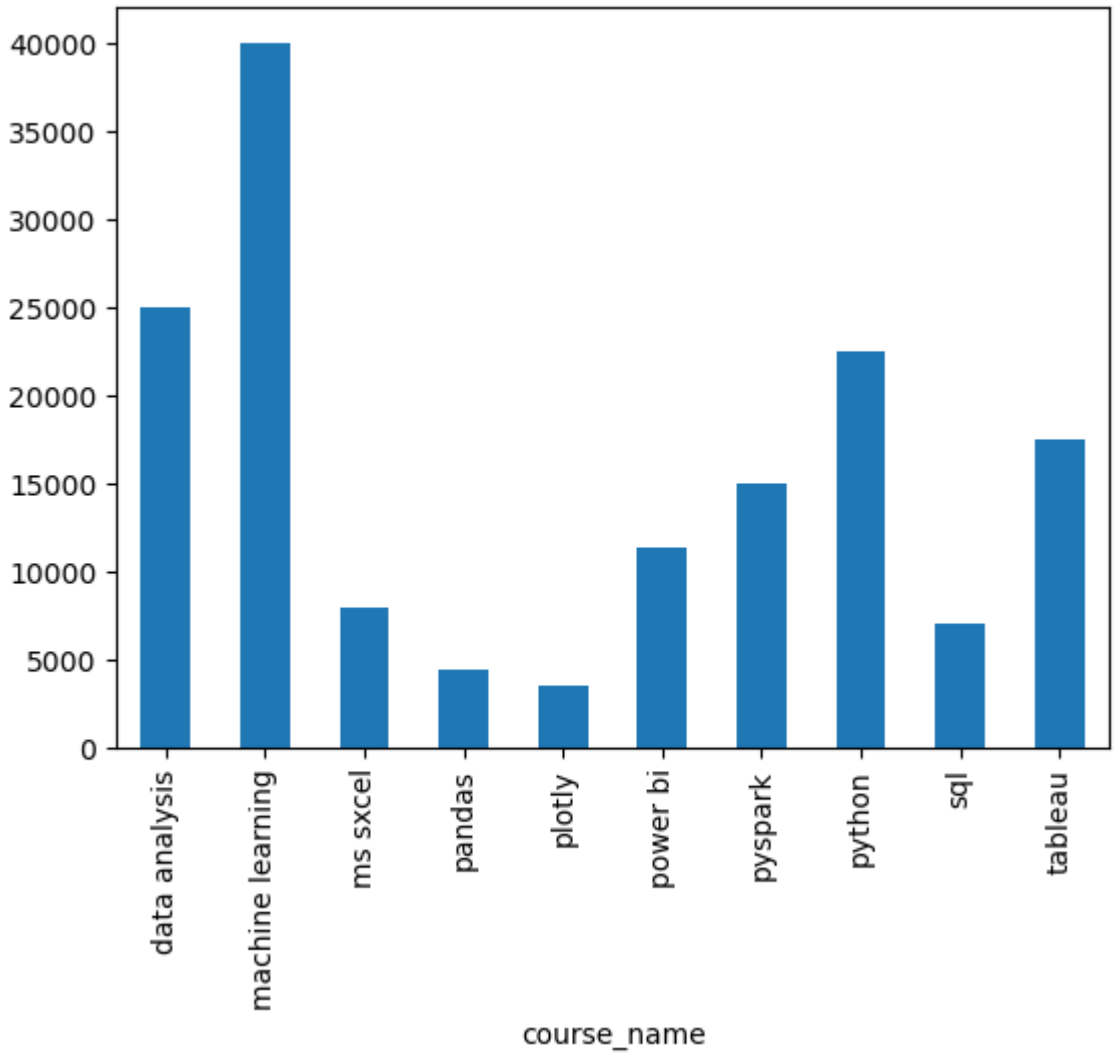
	name	course_name	price
0	Chhavi Lachman	python	2499
1	Preet Sha	python	2499
2	Fardeen Mahabir	python	2499
3	Kailash Harjo	python	2499
4	Seema Kota	python	2499
5	Shashank D'Alia	python	2499
6	Radha Dutt	python	2499
7	Pranab Natarajan	python	2499
8	Chhavi Lachman	machine learning	9999
9	Qabeel Raman	machine 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

	name	course_name	price
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

4. Plot bar chart for revenue/course

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

Out[ ]:
```
<Axes: xlabel='course_name'>
```



5. find students who enrolled in both the months

In [ ]:
```
common_student_id = np.intersect1d(nov['student_id'],dec['student_id'])
common_student_id
```

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

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

Out[ ]:

	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

6. find course that got no enrollment

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

Out[ ]:

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

7. find students who did not enroll into any courses

In [ ]:
```
student_id_list = np.setdiff1d(students['student_id'],regs['student_id'])
students[students['student_id'].isin(student_id_list)].shape[0]

(10/28)*100
```

Out[ ]:
```
35.714285714285715
```

8. Print student name -> partner name for all enrolled students

In [ ]:
```
students.merge(students,how='inner',left_on='partner',right_on='student_id')[['name
```

Out[ ]:

	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	Shashank D'Alia	Esha Butala

9. find top 3 students who did most number enrollments

In [ ]:
```
regs.merge(students,on='student_id').groupby(['student_id','name'])['name'].count()
```

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

10. find top 3 students who spent most amount of money on courses

In [ ]:
```
regs.merge(students,on='student_id').merge(courses,on='course_id').groupby(['studer
```

Out[ ]:
```
student_id  name
23          Chhavi Lachman      22594
14          Pranab Natarajan    15096
19          Qabeel Raman        13498
Name: price, dtype: int64
```

MultiIndex Object in Pandas

In pandas, a multiindex (or multi-level index) object is a way to create hierarchical indexes for Series and DataFrames. This means you can have multiple levels of row and column labels, allowing you to organize and structure your data in a more complex way. MultiIndexes are especially useful when dealing with datasets that have multiple dimensions or levels of categorization.

Here's a brief explanation of multiindex objects in pandas:

1. MultiIndex in Series:

- In a Series, a multiindex allows you to have multiple levels of row labels.
- You can think of it as having subcategories or subgroups for the data in your Series.
- To create a multiindex Series, you can use the `pd.MultiIndex.from_tuples`, `pd.MultiIndex.from_arrays`, or other constructors.

How to create multiindex object

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

In [ ]:
```
# 1. pd.MultiIndex.from_tuples()
index_val = [('cse',2019),('cse',2020),('cse',2021),('cse',2022),('ece',2019),('ece
multiindex = pd.MultiIndex.from_tuples(index_val)
```

In [ ]:
```
multiindex
```

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

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

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

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

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

In [ ]:
```
# how to fetch items from such a series
s['cse']
```

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

2. MultiIndex in DataFrames:

- In a DataFrame, a multiindex allows you to have hierarchical row and column labels.
- You can think of it as having multiple levels of row and column headers, which is useful when dealing with multi-dimensional data.
- To create a multiindex DataFrame, you can use `pd.MultiIndex.from_tuples`, `pd.MultiIndex.from_arrays`, or construct it directly when creating the DataFrame.

In [ ]:
```
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[ ]:

		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 [ ]:
```
# multiindex df from columns perspective
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','studer
)

branch_df2
```

Out[ ]:

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

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

In [ ]:
```
# Multiindex df in terms of both cols and index

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','studer
)

branch_df3
```

Out[ ]:

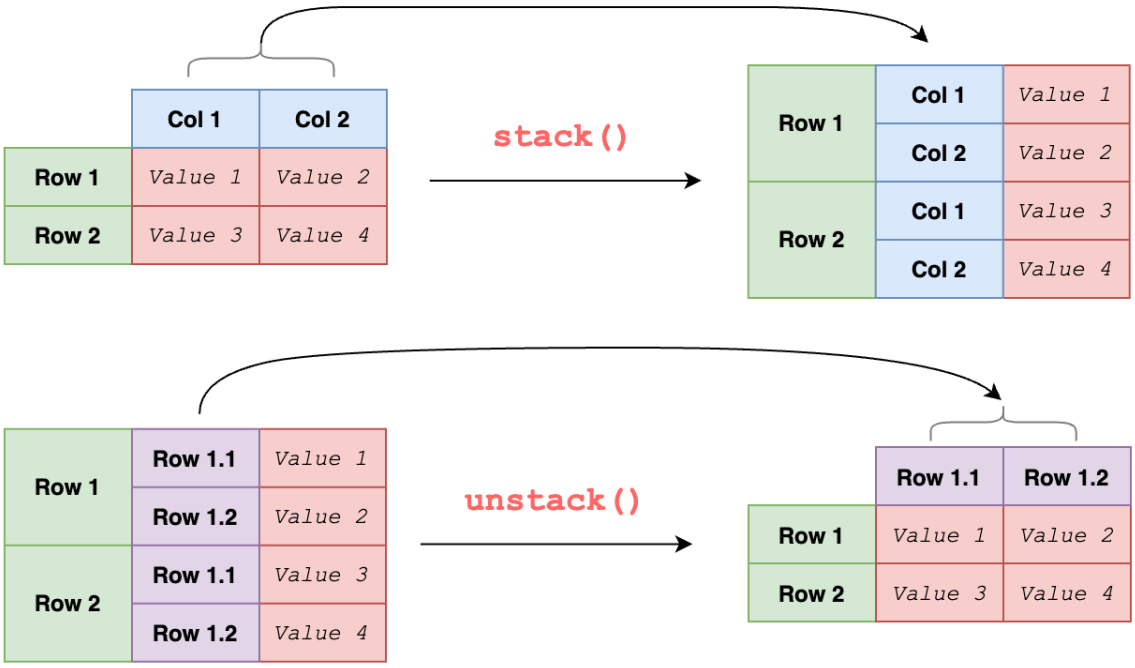
		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2019	1	2	0	0
	2020	3	4	0	0
	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0
	2020	11	12	0	0
	2021	13	14	0	0
	2022	15	16	0	0

MultiIndexes allow you to represent and manipulate complex, multi-level data structures efficiently in pandas, making it easier to work with and analyze data that has multiple dimensions or hierarchies. You can perform various operations and selections on multiindex objects to access and manipulate specific levels of data within your Series or DataFrame.

Stacking and Unstacking in MultiIndex Object in Pandas

In pandas, a multi-index object is a way to represent hierarchical data structures within a DataFrame or Series. Multi-indexing allows you to have multiple levels of row or column indices, providing a way to organize and work with complex, structured data.

"Stacking" and "unstacking" are operations that you can perform on multi-indexed DataFrames to change the arrangement of the data, essentially reshaping the data between a wide and a long format (or vice versa).



1. Stacking:

- Stacking is the process of "melting" or pivoting the innermost level of column labels to become the innermost level of row labels.
- This operation is typically used when you want to convert a wide DataFrame with multi-level columns into a long format.
- You can use the `.stack()` method to perform stacking. By default, it will stack the innermost level of columns.

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

In [ ]:
```
# Create a DataFrame with multi-level columns
df = pd.DataFrame(np.random.rand(4, 4), columns=[['A', 'A', 'B', 'B'], ['X', 'Y', '
print(df)
```

```
          A                   B
          X         Y         X         Y
0  0.960684  0.118538  0.900984  0.485585
1  0.946716  0.049913  0.444658  0.991469
2  0.656110  0.158270  0.759727  0.203801
3  0.360581  0.965035  0.797212  0.102426
```

In [ ]:
```
# Stack the innermost level of columns
stacked_df = df.stack()
print(stacked_df)
```

```
            A         B
0 X  0.960684  0.900984
  Y  0.118538  0.485585
1 X  0.946716  0.444658
  Y  0.049913  0.991469
2 X  0.656110  0.759727
  Y  0.158270  0.203801
3 X  0.360581  0.797212
  Y  0.965035  0.102426
```

2. **Unstacking**:

- Unstacking is the reverse operation of stacking. It involves pivoting the innermost level of row labels to become the innermost level of column labels.
- You can use the `.unstack()` method to perform unstacking. By default, it will unstack the innermost level of row labels.

Example:

In [ ]:
```
# Unstack the innermost level of row labels
unstacked_df = stacked_df.unstack()
print(unstacked_df)
```

```
          A                   B
          X         Y         X         Y
0  0.960684  0.118538  0.900984  0.485585
1  0.946716  0.049913  0.444658  0.991469
2  0.656110  0.158270  0.759727  0.203801
3  0.360581  0.965035  0.797212  0.102426
```

You can specify the level you want to stack or unstack by passing the `level` parameter to the `stack()` or `unstack()` methods. For example:

In [ ]:
```
# Stack the second level of columns
stacked_df = df.stack(level=1)
stacked_df
```

Out[ ]:

		A	B
0	**X**	0.960684	0.900984
	Y	0.118538	0.485585
1	**X**	0.946716	0.444658
	Y	0.049913	0.991469
2	**X**	0.656110	0.759727
	Y	0.158270	0.203801
3	**X**	0.360581	0.797212
	Y	0.965035	0.102426

In [ ]:
```
# Unstack the first level of row labels
unstacked_df = stacked_df.unstack(level=0)
unstacked_df
```

Out[ ]:

	A				B			
	0	**1**	**2**	**3**	**0**	**1**	**2**	**3**
X	0.960684	0.946716	0.65611	0.360581	0.900984	0.444658	0.759727	0.797212
Y	0.118538	0.049913	0.15827	0.965035	0.485585	0.991469	0.203801	0.102426

In [ ]:
```
index_val = [('cse',2019),('cse',2020),('cse',2021),('cse',2022),('ece',2019),('ece
multiindex = pd.MultiIndex.from_tuples(index_val)
multiindex.levels[1]
```

Out[ ]:
```
Index([2019, 2020, 2021, 2022], dtype='int64')
```

In [ ]:
```
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[ ]:

		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 [ ]:
```
# multiindex df from columns perspective
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','studer
)

branch_df2
```

Out[ ]:

	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 [ ]:
```
branch_df1
```

Out[ ]:

		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 [ ]:
```
branch_df1.unstack().unstack()
```

Out[ ]:
```
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
```

In [ ]:
```
branch_df1.unstack().stack()
```

Out[ ]:

		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 [ ]:
```
branch_df2
```

Out[ ]:

	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 [ ]:
```
branch_df2.stack()
```

Out[ ]:

		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 [ ]:
```
branch_df2.stack().stack()
```

Out[ ]:
```
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
```

Stacking and unstacking can be very useful when you need to reshape your data to make it more suitable for different types of analysis or visualization. They are common operations in data manipulation when working with multi-indexed DataFrames in pandas.

Working with MultiIndex DataFrames

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

Creating MutiIndex Dataframe

In [ ]:
```
index_val = [('cse',2019),('cse',2020),('cse',2021),('cse',2022),('ece',2019),('ece
multiindex = pd.MultiIndex.from_tuples(index_val)
multiindex.levels
```

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

In [ ]:
```
branch_df = 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','studer
)

branch_df
```

Out[ ]:

		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

Basic Checks

In [ ]:
```
# HEAD
branch_df.head()
```

Out[ ]:

		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 [ ]:
```
# Tail
branch_df.tail()
```

Out[ ]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	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 [ ]:
```
#shape
branch_df.shape
```

Out[ ]:
```
(8, 4)
```

In [ ]:
```
# info
branch_df.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: 632.0+ bytes
```

In [ ]:
```
# duplicated
branch_df.duplicated().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# isnull
branch_df.isnull().sum()
```

Out[ ]:
```
delhi   avg_package    0
        students       0
mumbai  avg_package    0
        students       0
dtype: int64
```

How to Extract

In [ ]:
```
# extracting single row
branch_df.loc[('cse',2022)]
```

Out[ ]:
```
delhi   avg_package    7
        students       8
mumbai  avg_package    0
        students       0
Name: (cse, 2022), dtype: int64
```

In [ ]:
```
branch_df
```

Out[ ]:

		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 [ ]:
```
# extract multiple rows
branch_df.loc[('cse',2021):('ece',2021)]
```

Out[ ]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	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

In [ ]:
```
# using iloc
branch_df.iloc[2:5]
```

Out[ ]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2021	5	6	0	0
	2022	7	8	0	0
ece	2019	9	10	0	0

In [ ]:
```
branch_df.iloc[2:8:2]
```

Out[ ]:

		delhi		mumbai	
		avg_package	students	avg_package	students
cse	2021	5	6	0	0
ece	2019	9	10	0	0
	2021	13	14	0	0

In [ ]:
```
# extacting cols
branch_df['delhi']['students']
```

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

In [ ]:
```
branch_df.iloc[:,1:3]
```

Out[ ]:

		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 [ ]:
```
# Extracting both
branch_df.iloc[[0,4],[1,2]]
```

Out[ ]:

		delhi	mumbai
		students	avg_package
cse	2019	2	0
ece	2019	10	0

Sorting

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

Out[ ]:

		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 [ ]:
```
branch_df.sort_index(ascending=[False,True])
```

Out[ ]:

		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 [ ]:
```
branch_df.sort_index(level=0,ascending=[False])
```

Out[ ]:

		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 [ ]:
```
# multiindex dataframe(col) -> transpose
branch_df.transpose()
```

Out[ ]:

		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 [ ]:
```
# swaplevel
branch_df.swaplevel(axis=1)
```

Out[ ]:

		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

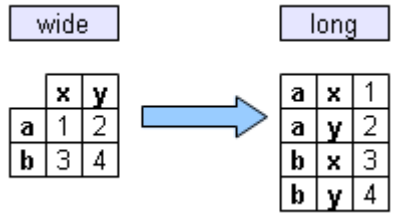
In [ ]:
```
branch_df.swaplevel()
```

Out[ ]:

		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

Long vs. Wide Data Formats in Data Analysis

Long Vs Wide Data



"Long" and "wide" are terms often used in data analysis and data reshaping in the context of data frames or tables, typically in software like R or Python. They describe two different ways of organizing and structuring data.

1. Long Format (also called "Tidy" or "Melted"):

- In the long format, each row typically represents an individual observation or data point, and each column represents a variable or attribute.
- This format is useful when you have a dataset where you want to store multiple measurements or values for the same individuals or entities.
- Long data is often more convenient for various types of statistical analysis, and it can be easier to filter, subset, and manipulate the data.

2. Wide Format (also called "Spread" or "Pivoted"):

- In the wide format, each row represents an individual, and multiple variables are stored in separate columns.
- This format is useful when you have a dataset with a few variables and you want to see the values for each individual at a glance, which can be more readable for humans.
- Wide data can be useful for simple summaries and initial data exploration.

Here's a simple example to illustrate the difference:

Long Format:

ID	Variable	Value
1	Age	25
1	Height	175
1	Weight	70
2	Age	30
2	Height	160
2	Weight	60

Wide Format:

ID	Age	Height	Weight
1	25	175	70
2	30	160	60

Converting data between long and wide formats is often necessary depending on the specific analysis or visualization tasks you want to perform. In software like R and Python, there are functions and libraries available for reshaping data between these formats, such as `tidyr` in R and `pivot` functions in Python's pandas library for moving from wide to long format, and `gather` in R and `melt` in pandas for moving from long to wide format.

Data Conversion : melt

- wide is converted into long

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

In [ ]:
```
pd.DataFrame({'cse':[120]})
```

Out[ ]:

	cse
0	120

In [ ]:
```
pd.DataFrame({'cse':[120]}).melt()
```

Out[ ]:

	variable	value
0	cse	120

In [ ]:
```
# melt -> branch with year
pd.DataFrame({'cse':[120],'ece':[100],'mech':[50]})
```

Out[ ]:

	cse	ece	mech
0	120	100	50

In [ ]:
```
# melt -> branch with year
pd.DataFrame({'cse':[120],'ece':[100],'mech':[50]}).melt()
```

Out[ ]:

	variable	value
0	cse	120
1	ece	100
2	mech	50

In [ ]:
```
# var_name and value_name
pd.DataFrame({'cse':[120],'ece':[100],'mech':[50]}).melt(var_name='branch',value_na
```

Out[ ]:

	branch	num_students
0	cse	120
1	ece	100
2	mech	50

In [ ]:
```
pd.DataFrame(
    {
        'branch':['cse','ece','mech'],
        '2020':[100,150,60],
        '2021':[120,130,80],
        '2022':[150,140,70]
    }
)
```

Out[ ]:

	branch	2020	2021	2022
0	cse	100	120	150
1	ece	150	130	140
2	mech	60	80	70

In [ ]:
```
pd.DataFrame(
    {
        'branch':['cse','ece','mech'],
        '2020':[100,150,60],
        '2021':[120,130,80],
        '2022':[150,140,70]
    }
).melt()
```

Out[ ]:

	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 [ ]:
```
# id_vars ->  we don't want to convert
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[ ]:

	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

Real-World Example:

- In the context of COVID-19 data, data for deaths and confirmed cases are initially stored in wide formats.
- The data is converted to long format, making it easier to conduct analyses.
- In the long format, each row represents a specific location, date, and the corresponding number of deaths or confirmed cases. This format allows for efficient merging and analysis, as it keeps related data in one place and facilitates further data exploration.

In [ ]:
```
# melt -> real world example
death = pd.read_csv('Data\Day42\death_covid.csv')
confirm = pd.read_csv('Data\Day42\Confirmed_covid.csv')
```

In [ ]:
```
death.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/2
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	
1	NaN	Albania	41.15330	20.168300	0	0	0	0	
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	

5 rows × 1081 columns

In [ ]:
```
confirm.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/2
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	
1	NaN	Albania	41.15330	20.168300	0	0	0	0	
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	

5 rows × 1081 columns

In [ ]:
```
death = death.melt(id_vars=['Province/State','Country/Region','Lat','Long'],var_nam
confirm = confirm.melt(id_vars=['Province/State','Country/Region','Lat','Long'],var
```

In [ ]:
```
death.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	date	num_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 [ ]:
```
confirm.head()
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	date	num_cases
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 [ ]:
```
confirm.merge(death,on=['Province/State','Country/Region','Lat','Long','date'])
```

Out[ ]:

	Province/State	Country/Region	Lat	Long	date	num_cases	num_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 [ ]:
```
confirm.merge(death,on=['Province/State','Country/Region','Lat','Long','date'])[['C
```

Out[ ]:

	Country/Region	date	num_cases	num_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

The choice between long and wide data formats depends on the nature of the dataset and the specific analysis or visualization tasks you want to perform. Converting data between these formats can help optimize data organization for different analytical needs.

Pivot Table in Python

In Python, you can create pivot tables using libraries like pandas or NumPy, which are commonly used for data manipulation and analysis.

Now, let's create a simple pivot table:

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
```

In [ ]:
```
# Sample data
data = {
    'Date': ['2023-01-01', '2023-01-01', '2023-01-02', '2023-01-02'],
    'Product': ['A', 'B', 'A', 'B'],
    'Sales': [100, 200, 150, 250],
}
```

In [ ]:
```
# Create a pandas DataFrame
df = pd.DataFrame(data)
```

In [ ]:
```
# Create a pivot table
pivot_table = pd.pivot_table(df, values='Sales', index='Date', columns='Product', a
```

In [ ]:
```
print(pivot_table)
```

```
Product       A    B
Date
2023-01-01  100  200
2023-01-02  150  250
```

In this example, we first create a DataFrame using sample data. Then, we use the `pd.pivot_table` function to create a pivot table. Here's what each argument does:

- `df`: The DataFrame containing the data.
- `values`: The column for which you want to aggregate values (in this case, 'Sales').
- `index`: The column that you want to use as the rows in the pivot table (in this case, 'Date').
- `columns`: The column that you want to use as the columns in the pivot table (in this case, 'Product').
- `aggfunc`: The aggregation function to apply when there are multiple values that need to be combined. You can use functions like 'sum', 'mean', 'max', etc., depending on your requirements.

Real-world Examples

In [ ]:
```
df = sns.load_dataset('tips')
```

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

Out[ ]:

	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 [ ]:
```
df.pivot_table(index='sex',columns='smoker',values='total_bill')
```

Out[ ]:

smoker	Yes	No
sex		
Male	22.284500	19.791237
Female	17.977879	18.105185

In [ ]:
```
## aggfunc# aggfunc
df.pivot_table(index='sex',columns='smoker',values='total_bill',aggfunc='std')
```

Out[ ]:

smoker	Yes	No
sex		
Male	9.911845	8.726566
Female	9.189751	7.286455

In [ ]:
```
# all cols together
df.pivot_table(index='sex',columns='smoker')['size']
```

```
<ipython-input-7-dd5735ad60ca>:2: FutureWarning: pivot_table dropped a column beca
use it failed to aggregate. This behavior is deprecated and will raise in a future
version of pandas. Select only the columns that can be aggregated.
  df.pivot_table(index='sex',columns='smoker')['size']
```

Out[ ]:

smoker	Yes	No
sex		
Male	2.500000	2.711340
Female	2.242424	2.592593

In [ ]:
```
# multidimensional
df.pivot_table(index=['sex','smoker'],columns=['day','time'],aggfunc={'size':'mean'
```

Out[ ]:

		size									
	day	Thur		Fri		Sat	Sun	Thur			
	time	Lunch	Dinner	Lunch	Dinner	Dinner	Dinner	Lunch	Dinner	Lunch	D
sex	smoker										
Male	Yes	2.300000	NaN	1.666667	2.4	2.629630	2.600000	5.00	NaN	2.20	
	No	2.500000	NaN	NaN	2.0	2.656250	2.883721	6.70	NaN	NaN	
Female	Yes	2.428571	NaN	2.000000	2.0	2.200000	2.500000	5.00	NaN	3.48	
	No	2.500000	2.0	3.000000	2.0	2.307692	3.071429	5.17	3.0	3.00	

In [ ]:
```
# margins
df.pivot_table(index='sex',columns='smoker',values='total_bill',aggfunc='sum',margi
```

Out[ ]:

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

Plotting graph

In [ ]:
```
df = pd.read_csv('Data\Day43\expense_data.csv')
```

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

Out[ ]:

	Date	Account	Category	Subcategory	Note	INR	Income/Expense	Note.1	Amou
0	3/2/2022 10:11	CUB - online payment	Food	NaN	Brownie	50.0	Expense	NaN	5
1	3/2/2022 10:11	CUB - online payment	Other	NaN	To lended people	300.0	Expense	NaN	30
2	3/1/2022 19:50	CUB - online payment	Food	NaN	Dinner	78.0	Expense	NaN	7
3	3/1/2022 18:56	CUB - online payment	Transportation	NaN	Metro	30.0	Expense	NaN	3
4	3/1/2022 18:22	CUB - online payment	Food	NaN	Snacks	67.0	Expense	NaN	6

In [ ]:
```
df['Category'].value_counts()
```

Out[ ]:
```
Category
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: count, dtype: int64
```

In [ ]:
```
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 [ ]:
```
df['Date'] = pd.to_datetime(df['Date'])
```

In [ ]:
```
df['month'] = df['Date'].dt.month_name()
```

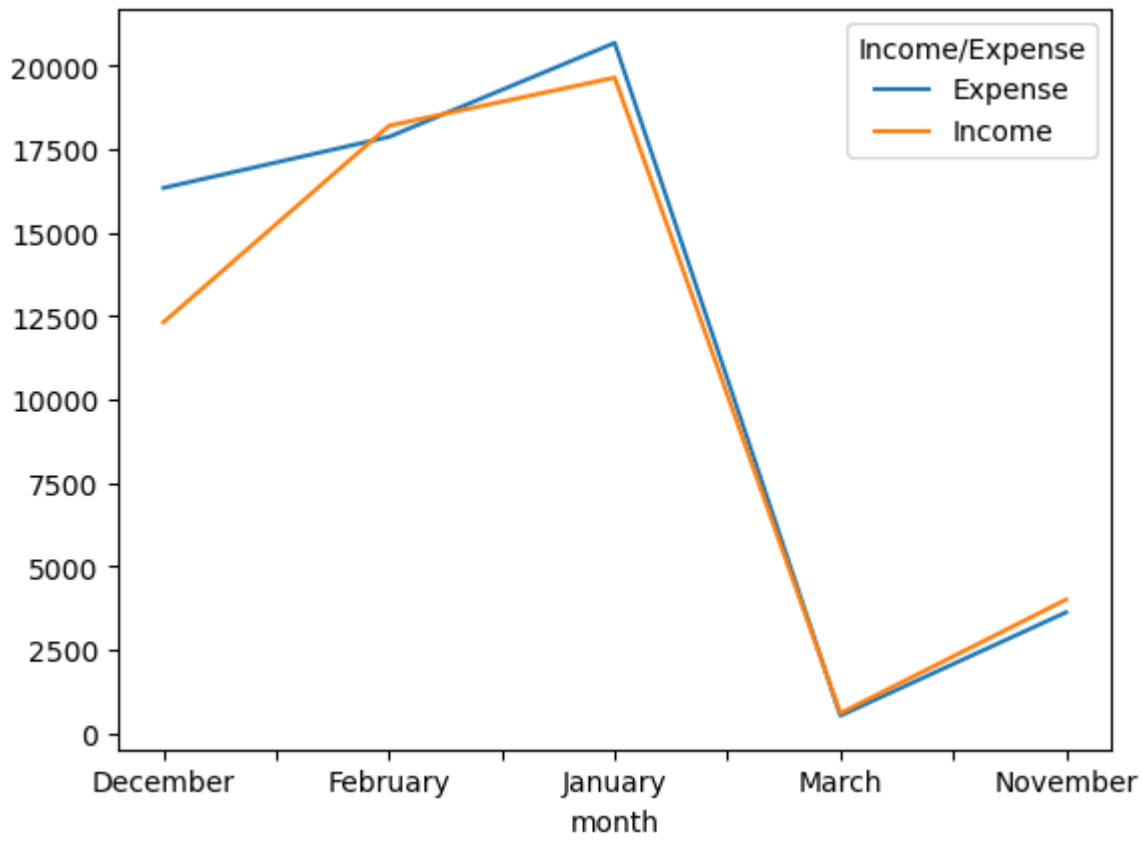
In [ ]:
```
df.head()
```

Out[ ]:

	Date	Account	Category	Subcategory	Note	INR	Income/Expense	Note.1	Amou
0	2022-03-02 10:11:00	CUB - online payment	Food	NaN	Brownie	50.0	Expense	NaN	50
1	2022-03-02 10:11:00	CUB - online payment	Other	NaN	To lended people	300.0	Expense	NaN	300
2	2022-03-01 19:50:00	CUB - online payment	Food	NaN	Dinner	78.0	Expense	NaN	78
3	2022-03-01 18:56:00	CUB - online payment	Transportation	NaN	Metro	30.0	Expense	NaN	30
4	2022-03-01 18:22:00	CUB - online payment	Food	NaN	Snacks	67.0	Expense	NaN	67

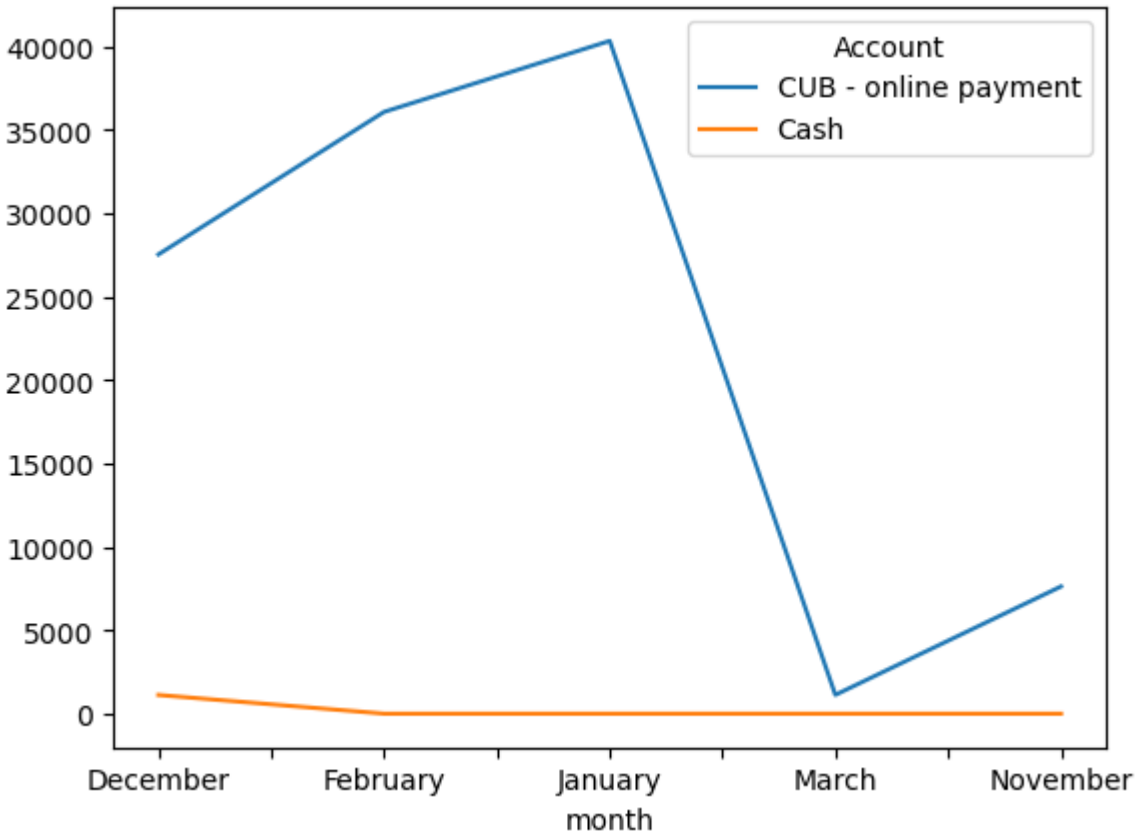
In [ ]:
```
df.pivot_table(index='month',columns='Income/Expense',values='INR',aggfunc='sum',fi
```

Out[ ]:
```
<Axes: xlabel='month'>
```



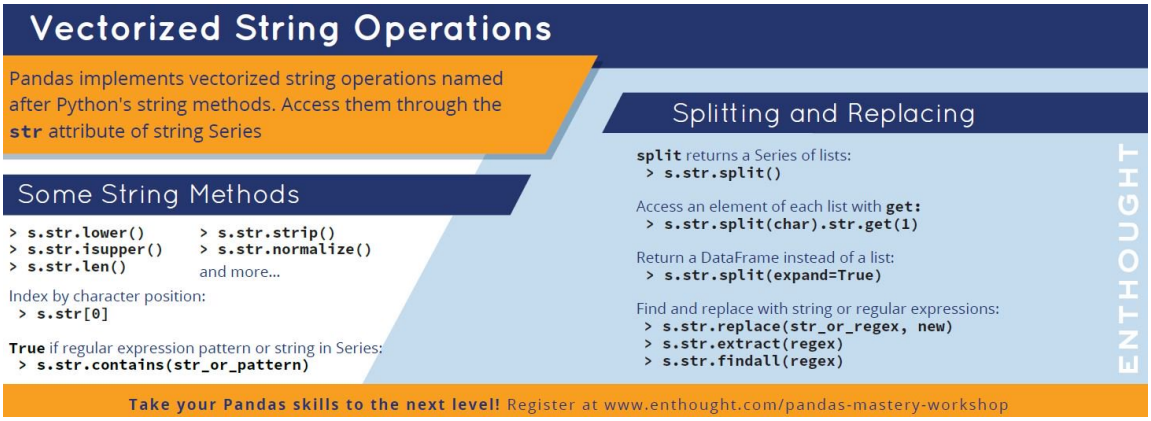
In [ ]:
```
df.pivot_table(index='month',columns='Account',values='INR',aggfunc='sum',fill_valu
```

Out[ ]:
```
<Axes: xlabel='month'>
```


Account
CUB - online payment
Cash
40000
35000
30000
25000
20000
15000
10000
5000
0
December
February
January
March
November
month

Vectorized String Opearations in Pandas

- Vectorized string operations in Pandas refer to the ability to apply string operations to a series or dataframe of strings in a single operation, rather than looping over each string individually. This can be achieved using the str attribute of a Series or DataFrame object, which provides a number of vectorized string methods.
- Vectorized string operations in Pandas refer to the ability to apply string functions and operations to entire arrays of strings (columns or Series containing strings) without the need for explicit loops or iteration. This is made possible by Pandas' integration with the NumPy library, which allows for efficient element-wise operations.
- When you have a Pandas DataFrame or Series containing string data, you can use various string methods that are applied to every element in the column simultaneously. This can significantly improve the efficiency and readability of your code. Some of the commonly used vectorized string operations in Pandas include methods like `.str.lower()`, `.str.upper()`, `.str.strip()`, `.str.replace()`, and many more.
- Vectorized string operations not only make your code more concise and readable but also often lead to improved performance compared to explicit for-loops, especially when dealing with large datasets.



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

In [ ]:
```
s = pd.Series(['cat','mat',None,'rat'])
```

In [ ]:
```
# str ->  string accessor
s.str.startswith('c')
```

Out[ ]:
```
0     True
1    False
2     None
3    False
dtype: object
```

Real-world Dataset - Titanic Dataset

In [ ]:
```
df = pd.read_csv('Data\Day44\Titanic.csv')
```

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

Out[ ]:

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

In [ ]:
```
df['Name']
```

Out[ ]:
```
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

In [ ]:
```
# lower/upper/capitalize/title
df['Name'].str.upper()
df['Name'].str.capitalize()
df['Name'].str.title()
```

Out[ ]:
```
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 [ ]:
```
# len

df['Name'][df['Name'].str.len()]
```

Out[ ]:
```
23                      Sloper, Mr. William Thompson
51                      Nosworthy, Mr. Richard Cater
22                       McGowan, Miss. Anna "Annie"
44                     Devaney, Miss. Margaret Delia
24                     Palsson, Miss. Torborg Danira
                           ...
21                          Beesley, Mr. Lawrence
28                     O'Dwyer, Miss. Ellen "Nellie"
40    Ahlin, Mrs. Johan (Johanna Persdotter Larsson)
21                          Beesley, Mr. Lawrence
19                       Masselmani, Mrs. Fatima
Name: Name, Length: 891, dtype: object
```

In [ ]:
```
df['Name'][df['Name'].str.len() == 82].values[0]
```

Out[ ]:
```
'Penasco y Castellana, Mrs. Victor de Satode (Maria Josefa Perez de Soto y Vallej
o)'
```

In [ ]:
```
# strip
df['Name'].str.strip()
```

Out[ ]:
```
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 [ ]:
```
# split -> get
df['lastname'] = df['Name'].str.split(',').str.get(0)
df.head()
```

Out[ ]:

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

In [ ]:
```
df[['title','firstname']] = df['Name'].str.split(',').str.get(1).str.strip().str.sp
df.head()
```

Out[ ]:

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

In [ ]:
```
df['title'].value_counts()
```

Out[ ]:
```
title
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: count, dtype: int64
```

In [ ]:
```
# replace
df['title'] = df['title'].str.replace('Ms.','Miss.')
df['title'] = df['title'].str.replace('Mlle.','Miss.')
```

In [ ]:
```
df['title'].value_counts()
```

Out[ ]:
```
title
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: count, dtype: int64
```

filtering

In [ ]:
```
# startswith/endswith
df[df['firstname'].str.endswith('A')]
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
64	65	0	1	Stewart, Mr. Albert A	male	NaN	0	0	PC 17605	27.7208	NaN
303	304	1	2	Keane, Miss. Nora A	female	NaN	0	0	226593	12.3500	E101

In [ ]:
```
# isdigit/isalpha...
df[df['firstname'].str.isdigit()]
```

Out[ ]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

slicing

In [ ]:
```
df['Name'].str[::-1]
```

Out[ ]:
```
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
```

Date and Time in Pandas

In Pandas, you can work with dates and times using the `datetime` data type. Pandas provides several data structures and functions for handling date and time data, making it convenient for time series data analysis.

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

1. `Timestamp`:

This represents a single timestamp and is the fundamental data type for time series data in Pandas.

Time stamps reference particular moments in time (e.g., Oct 24th, 2022 at 7:00pm)

In [ ]:
```
# Creating a Timestamp object
pd.Timestamp('2023/1/5')
```

Out[ ]:
```
Timestamp('2023-01-05 00:00:00')
```

In [ ]:
```
# variations
pd.Timestamp('2023-1-5')
pd.Timestamp('2023, 1, 5')
```

Out[ ]:
```
Timestamp('2023-01-05 00:00:00')
```

In [ ]:
```
# only year
pd.Timestamp('2023')
```

Out[ ]:
```
Timestamp('2023-01-01 00:00:00')
```

In [ ]:
```
# using text
pd.Timestamp('5th January 2023')
```

Out[ ]:
```
Timestamp('2023-01-05 00:00:00')
```

In [ ]:
```
# providing time also
pd.Timestamp('5th January 2023 9:21AM')
```

Out[ ]:
```
Timestamp('2023-01-05 09:21:00')
```

Using Python Datetime Object

In [ ]:
```
# using datetime.datetime object
import datetime as dt

x = pd.Timestamp(dt.datetime(2023,1,5,9,21,56))
x
```

Out[ ]:
```
Timestamp('2023-01-05 09:21:56')
```

In [ ]:
```
# fetching attributes
x.year
```

Out[ ]:
```
2023
```

In [ ]:
```
x.month
```

Out[ ]:
```
1
```

In [ ]:
```
x.day
x.hour
x.minute
x.second
```

Out[ ]:
```
56
```

why separate objects to handle data and time when python already has datetime functionality?

Python's `datetime` module provides a comprehensive way to work with dates and times, and it is a fundamental part of Python's standard library. However, Pandas introduces separate objects to handle dates and times for several reasons:

1. **Efficiency**: The `datetime` module in Python is flexible and comprehensive, but it may not be as efficient when dealing with large datasets. Pandas' datetime objects are optimized for performance and are designed for working with data, making them more suitable for operations on large time series datasets.

2. **Data Alignment**: Pandas focuses on data manipulation and analysis, so it provides tools for aligning data with time-based indices and working with irregular time series. This is particularly useful in financial and scientific data analysis.

3. **Convenience**: Pandas provides a high-level API for working with time series data, which can make your code more concise and readable. It simplifies common operations such as resampling, aggregation, and filtering.

4. **Integration with DataFrames**: Pandas seamlessly integrates its date and time objects with DataFrames. This integration allows you to easily create, manipulate, and analyze time series data within the context of your data analysis tasks.

5. **Time Zones**: Pandas has built-in support for handling time zones and daylight saving time, making it more suitable for working with global datasets and international time series data.

6. **Frequency-Based Data**: Pandas introduces concepts like `Period` and `PeriodIndex` to work with fixed-frequency time data, which is common in various applications, such as financial time series analysis.

While Python's `datetime` module is powerful and flexible, it is a general-purpose module and is not specifically optimized for data analysis and manipulation. Pandas complements Python's `datetime` module by providing a more data-centric and efficient approach to working with dates and times, especially within the context of data analysis and time series data. This separation of functionality allows Pandas to offer a more streamlined and efficient experience when dealing with time-related data in the realm of data science and data analysis.

2. `DatetimeIndex`:

This is an index that consists of `Timestamp` objects. It is used to create time series data in Pandas DataFrames.

In [ ]:
```
# from strings
pd.DatetimeIndex(['2023/1/1','2022/1/1','2021/1/1'])
```

Out[ ]:
```
DatetimeIndex(['2023-01-01', '2022-01-01', '2021-01-01'], dtype='datetime64[ns]',
freq=None)
```

In [ ]:
```
# from strings
type(pd.DatetimeIndex(['2023/1/1','2022/1/1','2021/1/1']))
```

Out[ ]:
```
pandas.core.indexes.datetimes.DatetimeIndex
```

In [ ]:
```
# using python datetime object
pd.DatetimeIndex([dt.datetime(2023,1,1),dt.datetime(2022,1,1),dt.datetime(2021,1,1)
```

Out[ ]:
```
DatetimeIndex(['2023-01-01', '2022-01-01', '2021-01-01'], dtype='datetime64[ns]',
freq=None)
```

In [ ]:
```
# using pd.timestamps
dt_index = pd.DatetimeIndex([pd.Timestamp(2023,1,1),pd.Timestamp(2022,1,1),pd.Times
```

In [ ]:
```
dt_index
```

Out[ ]:
```
DatetimeIndex(['2023-01-01', '2022-01-01', '2021-01-01'], dtype='datetime64[ns]',
freq=None)
```

In [ ]:
```
# using datatimeindex as series index

pd.Series([1,2,3],index=dt_index)
```

Out[ ]:
```
2023-01-01    1
2022-01-01    2
2021-01-01    3
dtype: int64
```

3. date_range function

In [ ]:
```
# generate daily dates in a given range
pd.date_range(start='2023/1/5',end='2023/2/28',freq='D')
```

Out[ ]:
```
DatetimeIndex(['2023-01-05', '2023-01-06', '2023-01-07', '2023-01-08',
               '2023-01-09', '2023-01-10', '2023-01-11', '2023-01-12',
               '2023-01-13', '2023-01-14', '2023-01-15', '2023-01-16',
               '2023-01-17', '2023-01-18', '2023-01-19', '2023-01-20',
               '2023-01-21', '2023-01-22', '2023-01-23', '2023-01-24',
               '2023-01-25', '2023-01-26', '2023-01-27', '2023-01-28',
               '2023-01-29', '2023-01-30', '2023-01-31', '2023-02-01',
               '2023-02-02', '2023-02-03', '2023-02-04', '2023-02-05',
               '2023-02-06', '2023-02-07', '2023-02-08', '2023-02-09',
               '2023-02-10', '2023-02-11', '2023-02-12', '2023-02-13',
               '2023-02-14', '2023-02-15', '2023-02-16', '2023-02-17',
               '2023-02-18', '2023-02-19', '2023-02-20', '2023-02-21',
               '2023-02-22', '2023-02-23', '2023-02-24', '2023-02-25',
               '2023-02-26', '2023-02-27', '2023-02-28'],
              dtype='datetime64[ns]', freq='D')
```

In [ ]:
```
# alternate days in a given range
pd.date_range(start='2023/1/5',end='2023/2/28',freq='2D')
```

Out[ ]:
```
DatetimeIndex(['2023-01-05', '2023-01-07', '2023-01-09', '2023-01-11',
               '2023-01-13', '2023-01-15', '2023-01-17', '2023-01-19',
               '2023-01-21', '2023-01-23', '2023-01-25', '2023-01-27',
               '2023-01-29', '2023-01-31', '2023-02-02', '2023-02-04',
               '2023-02-06', '2023-02-08', '2023-02-10', '2023-02-12',
               '2023-02-14', '2023-02-16', '2023-02-18', '2023-02-20',
               '2023-02-22', '2023-02-24', '2023-02-26', '2023-02-28'],
              dtype='datetime64[ns]', freq='2D')
```

In [ ]:
```
# B -> business days
pd.date_range(start='2023/1/5',end='2023/2/28',freq='B')
```

Out[ ]:
```
DatetimeIndex(['2023-01-05', '2023-01-06', '2023-01-09', '2023-01-10',
               '2023-01-11', '2023-01-12', '2023-01-13', '2023-01-16',
               '2023-01-17', '2023-01-18', '2023-01-19', '2023-01-20',
               '2023-01-23', '2023-01-24', '2023-01-25', '2023-01-26',
               '2023-01-27', '2023-01-30', '2023-01-31', '2023-02-01',
               '2023-02-02', '2023-02-03', '2023-02-06', '2023-02-07',
               '2023-02-08', '2023-02-09', '2023-02-10', '2023-02-13',
               '2023-02-14', '2023-02-15', '2023-02-16', '2023-02-17',
               '2023-02-20', '2023-02-21', '2023-02-22', '2023-02-23',
               '2023-02-24', '2023-02-27', '2023-02-28'],
              dtype='datetime64[ns]', freq='B')
```

In [ ]:
```
# W -> one week per day
pd.date_range(start='2023/1/5',end='2023/2/28',freq='W-THU')
```

Out[ ]:
```
DatetimeIndex(['2023-01-05', '2023-01-12', '2023-01-19', '2023-01-26',
               '2023-02-02', '2023-02-09', '2023-02-16', '2023-02-23'],
              dtype='datetime64[ns]', freq='W-THU')
```

In [ ]:
```
# H -> Hourly data(factor)
pd.date_range(start='2023/1/5',end='2023/2/28',freq='6H')
```

Out[ ]:
```
DatetimeIndex(['2023-01-05 00:00:00', '2023-01-05 06:00:00',
               '2023-01-05 12:00:00', '2023-01-05 18:00:00',
               '2023-01-06 00:00:00', '2023-01-06 06:00:00',
               '2023-01-06 12:00:00', '2023-01-06 18:00:00',
               '2023-01-07 00:00:00', '2023-01-07 06:00:00',
               ...
               '2023-02-25 18:00:00', '2023-02-26 00:00:00',
               '2023-02-26 06:00:00', '2023-02-26 12:00:00',
               '2023-02-26 18:00:00', '2023-02-27 00:00:00',
               '2023-02-27 06:00:00', '2023-02-27 12:00:00',
               '2023-02-27 18:00:00', '2023-02-28 00:00:00'],
              dtype='datetime64[ns]', length=217, freq='6H')
```

In [ ]:
```
# M -> Month end
pd.date_range(start='2023/1/5',end='2023/2/28',freq='M')
```

Out[ ]:
```
DatetimeIndex(['2023-01-31', '2023-02-28'], dtype='datetime64[ns]', freq='M')
```

In [ ]:
```
# A -> Year end
pd.date_range(start='2023/1/5',end='2030/2/28',freq='A')
```

Out[ ]:
```
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 [ ]:
```
# MS -> Month start
pd.date_range(start='2023/1/5',end='2023/2/28',freq='MS')
```

Out[ ]:
```
DatetimeIndex(['2023-02-01'], dtype='datetime64[ns]', freq='MS')
```

4. to_datetime function

converts an existing objects to pandas timestamp/datetimeindex object

In [ ]:
```
# simple series example

s = pd.Series(['2023/1/1','2022/1/1','2021/1/1'])
pd.to_datetime(s).dt.day_name()
```

Out[ ]:
```
0      Sunday
1    Saturday
2      Friday
dtype: object
```

In [ ]:
```
# with errors
s = pd.Series(['2023/1/1','2022/1/1','2021/130/1'])
pd.to_datetime(s,errors='coerce').dt.month_name()
```

Out[ ]:
```
0    January
1    January
2        NaN
dtype: object
```

In [ ]:
```
df = pd.read_csv('Data\Day43\expense_data.csv')
df.shape
```

Out[ ]:
```
(277, 11)
```

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

Out[ ]:

	Date	Account	Category	Subcategory	Note	INR	Income/Expense	Note.1	Amou
0	3/2/2022 10:11	CUB - online payment	Food	NaN	Brownie	50.0	Expense	NaN	5
1	3/2/2022 10:11	CUB - online payment	Other	NaN	To lended people	300.0	Expense	NaN	30
2	3/1/2022 19:50	CUB - online payment	Food	NaN	Dinner	78.0	Expense	NaN	7
3	3/1/2022 18:56	CUB - online payment	Transportation	NaN	Metro	30.0	Expense	NaN	3
4	3/1/2022 18:22	CUB - online payment	Food	NaN	Snacks	67.0	Expense	NaN	6

In [ ]:
```
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 [ ]:
```
df['Date'] = pd.to_datetime(df['Date'])
```

In [ ]:
```
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
```

5. `dt accessor`

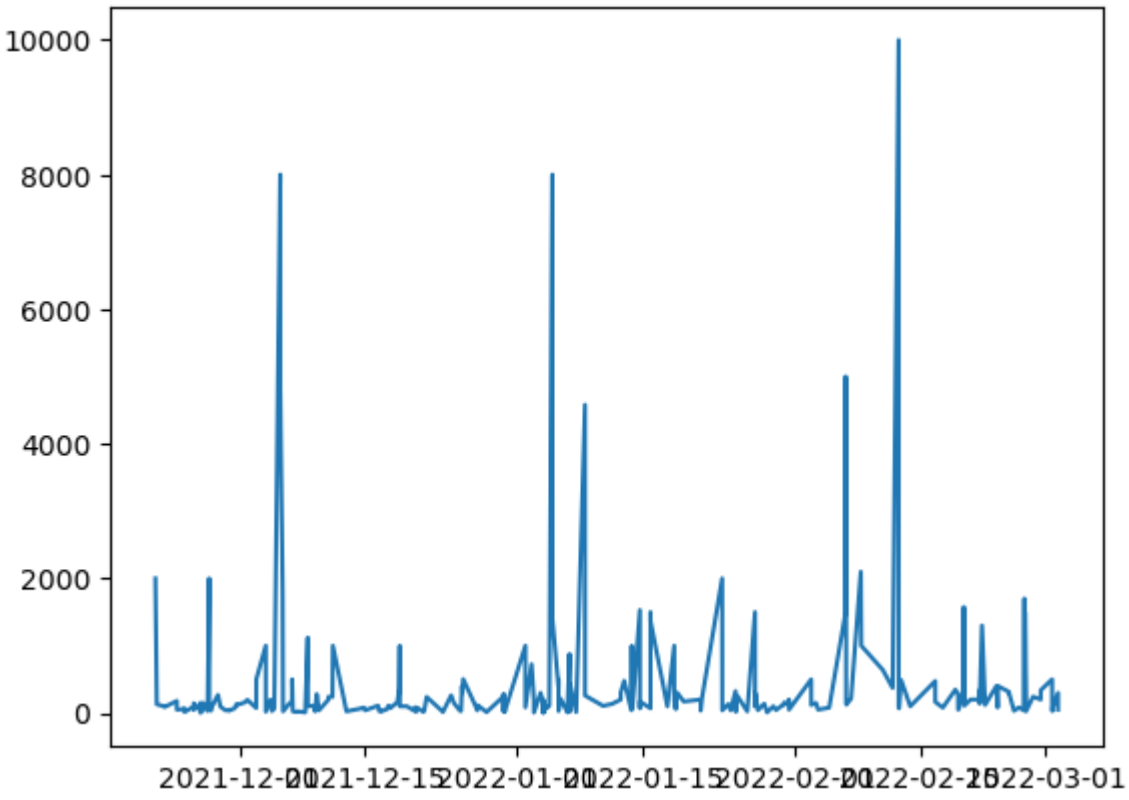
Accessor object for datetimelike properties of the Series values.

In [ ]:
```
df['Date'].dt.is_quarter_start
```

Out[ ]:
```
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 [ ]:
```
# plot graph
import matplotlib.pyplot as plt
plt.plot(df['Date'],df['INR'])
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x181faeba430>]
```



In [ ]:
```
# day name wise bar chart/month wise bar chart

df['day_name'] = df['Date'].dt.day_name()
```

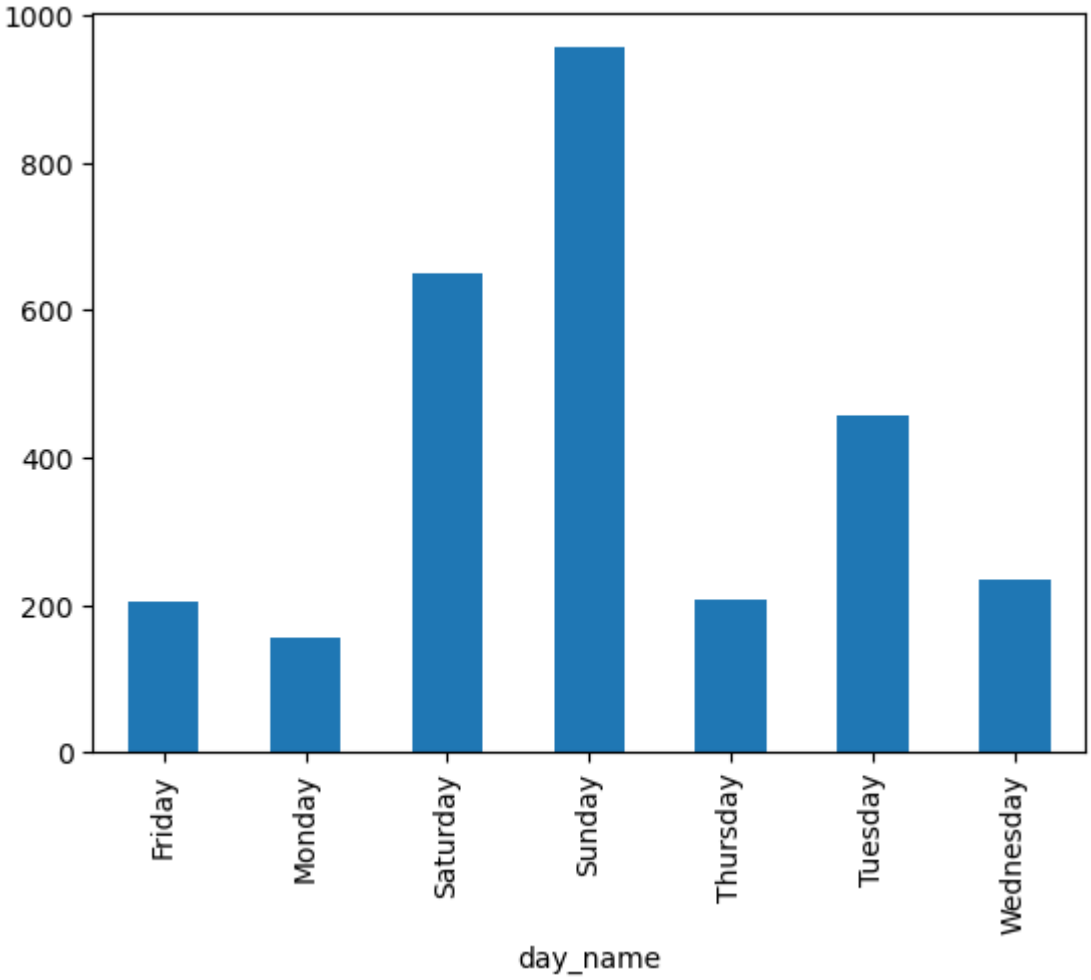
In [ ]:
```
df.head()
```

Out[ ]:

	Date	Account	Category	Subcategory	Note	INR	Income/Expense	Note.1	Amou
0	2022-03-02 10:11:00	CUB - online payment	Food	NaN	Brownie	50.0	Expense	NaN	50
1	2022-03-02 10:11:00	CUB - online payment	Other	NaN	To lended people	300.0	Expense	NaN	300
2	2022-03-01 19:50:00	CUB - online payment	Food	NaN	Dinner	78.0	Expense	NaN	78
3	2022-03-01 18:56:00	CUB - online payment	Transportation	NaN	Metro	30.0	Expense	NaN	30
4	2022-03-01 18:22:00	CUB - online payment	Food	NaN	Snacks	67.0	Expense	NaN	67

In [ ]:
```
df.groupby('day_name')['INR'].mean().plot(kind='bar')
```

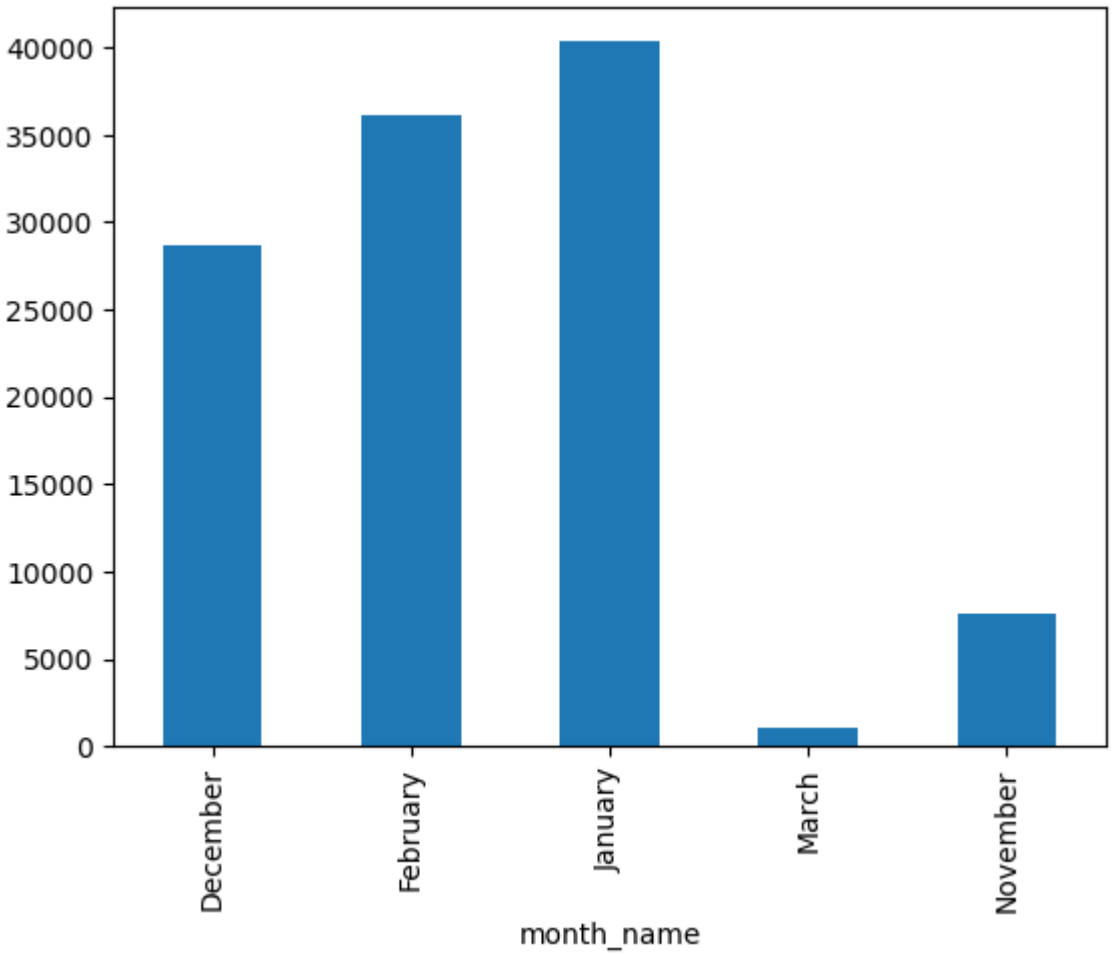
Out[ ]:
```
<Axes: xlabel='day_name'>
```



In [ ]:
```
df['month_name'] = df['Date'].dt.month_name()
```

In [ ]:
```
df.groupby('month_name')['INR'].sum().plot(kind='bar')
```

Out[ ]:
```
<Axes: xlabel='month_name'>
```



Pandas also provides powerful time series functionality, including the ability to resample, group, and perform various time-based operations on data. You can work with date and time data in Pandas to analyze and manipulate time series data effectively.

2D Line Plot in Matplotlib

Firstly,

What is Matplotlib?

Matplotlib is a popular data visualization library in Python. It provides a wide range of tools for creating various types of plots and charts, making it a valuable tool for data analysis, scientific research, and data presentation. Matplotlib allows you to create high-quality, customizable plots and figures for a variety of purposes, including line plots, bar charts, scatter plots, histograms, and more.

Matplotlib is highly customizable and can be used to control almost every aspect of your plots, from the colors and styles to labels and legends. It provides both a functional and an object-oriented interface for creating plots, making it suitable for a wide range of users, from beginners to advanced data scientists and researchers.

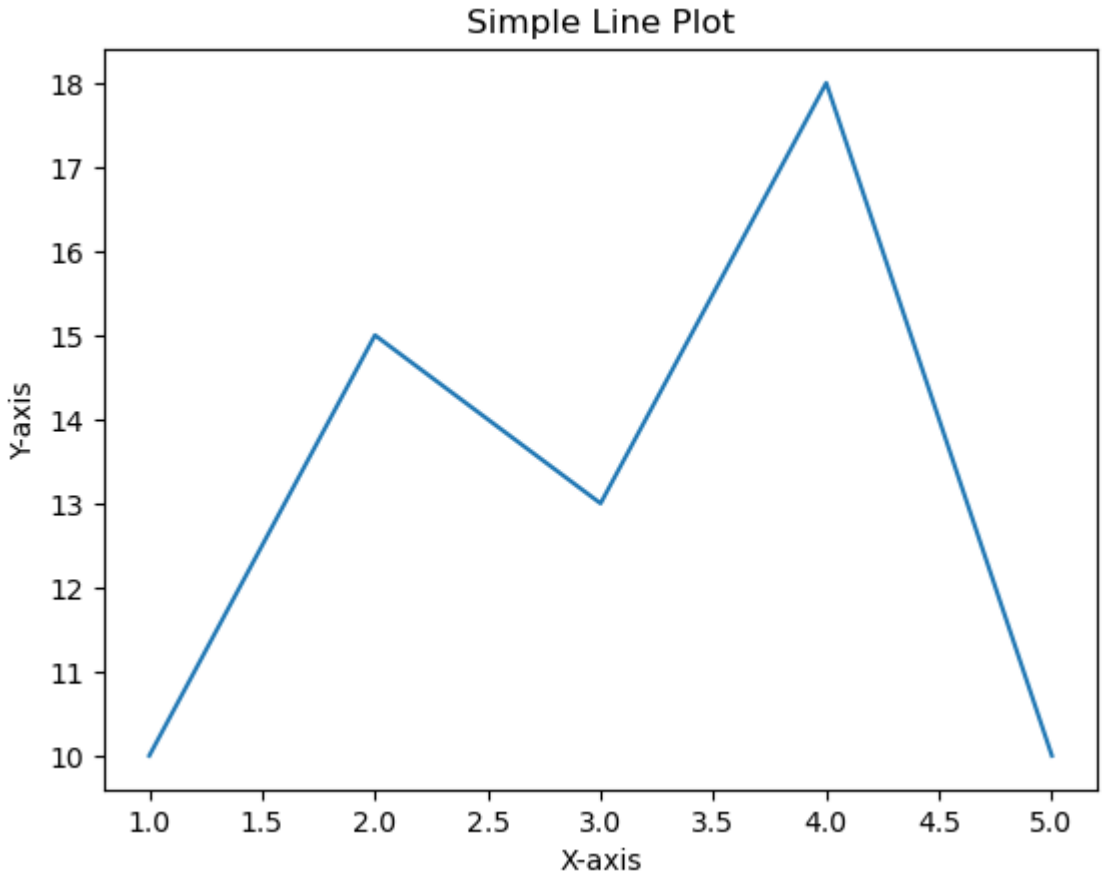
Matplotlib can be used in various contexts, including Jupyter notebooks, standalone Python scripts, and integration with web applications and GUI frameworks. It also works well with other Python libraries commonly used in data analysis and scientific computing, such as NumPy and Pandas.

To use Matplotlib, you typically need to import the library in your Python code, create the desired plot or chart, and then display or save it as needed. Here's a simple example of creating a basic line plot using Matplotlib:

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

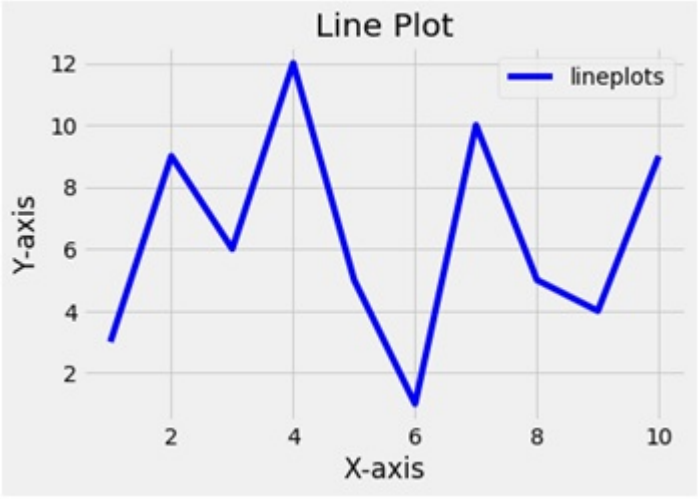
In [ ]:
```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [10, 15, 13, 18, 10]
```

```
# Create a line plot
plt.plot(x, y)

# Add labels and a title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')

# Show the plot
plt.show()
```



This is just a basic introduction to Matplotlib. The library is quite versatile, and you can explore its documentation and tutorials to learn more about its capabilities and how to create various types of visualizations for your data.

Line Plot

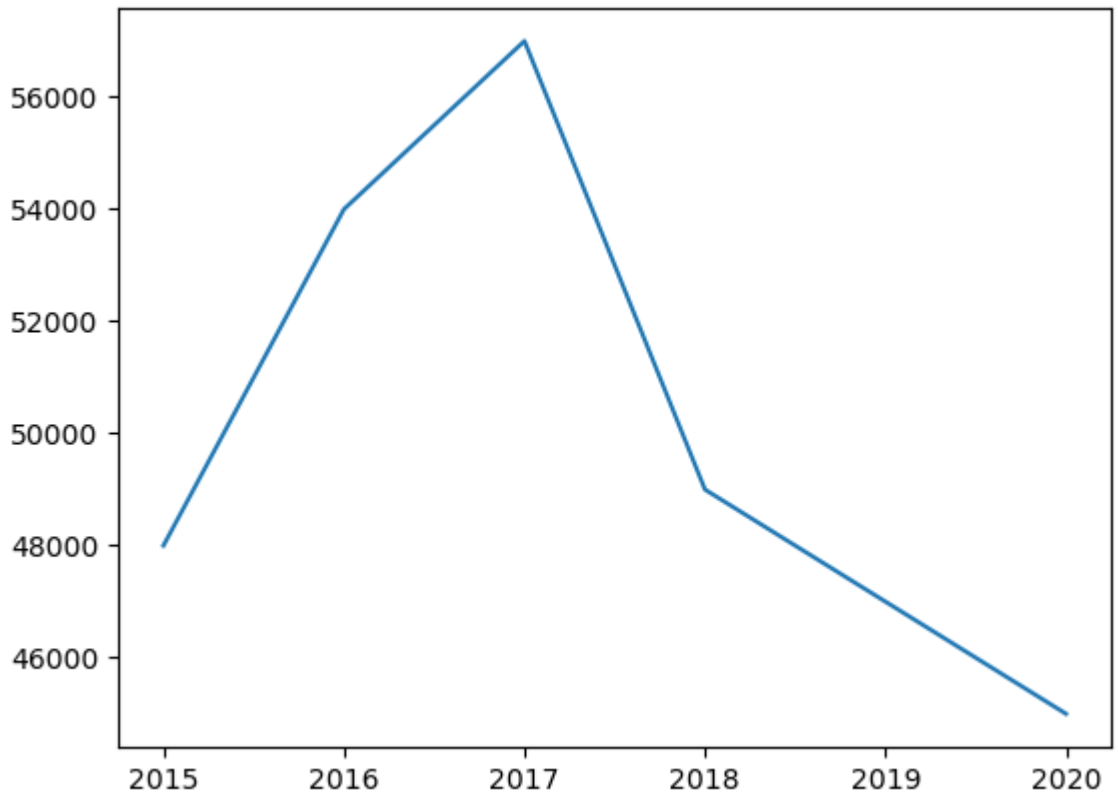


A 2D line plot is one of the most common types of plots in Matplotlib. It's used to visualize data with two continuous variables, typically representing one variable on the x-axis and another on the y-axis, and connecting the data points with lines. This type of plot is useful for showing trends, relationships, or patterns in data over a continuous range.

- Bivariate Analysis
- categorical -> numerical and numerical -> numerical
- Use case - Time series data

In [ ]:
```
# plotting simple graphs

price = [48000,54000,57000,49000,47000,45000]
year = [2015,2016,2017,2018,2019,2020]

plt.plot(year,price)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x28b7c3742e0>]
```



Real-world Dataset

In [ ]:
```
batsman = pd.read_csv('Data\Day45\sharma-kohli.csv')
```

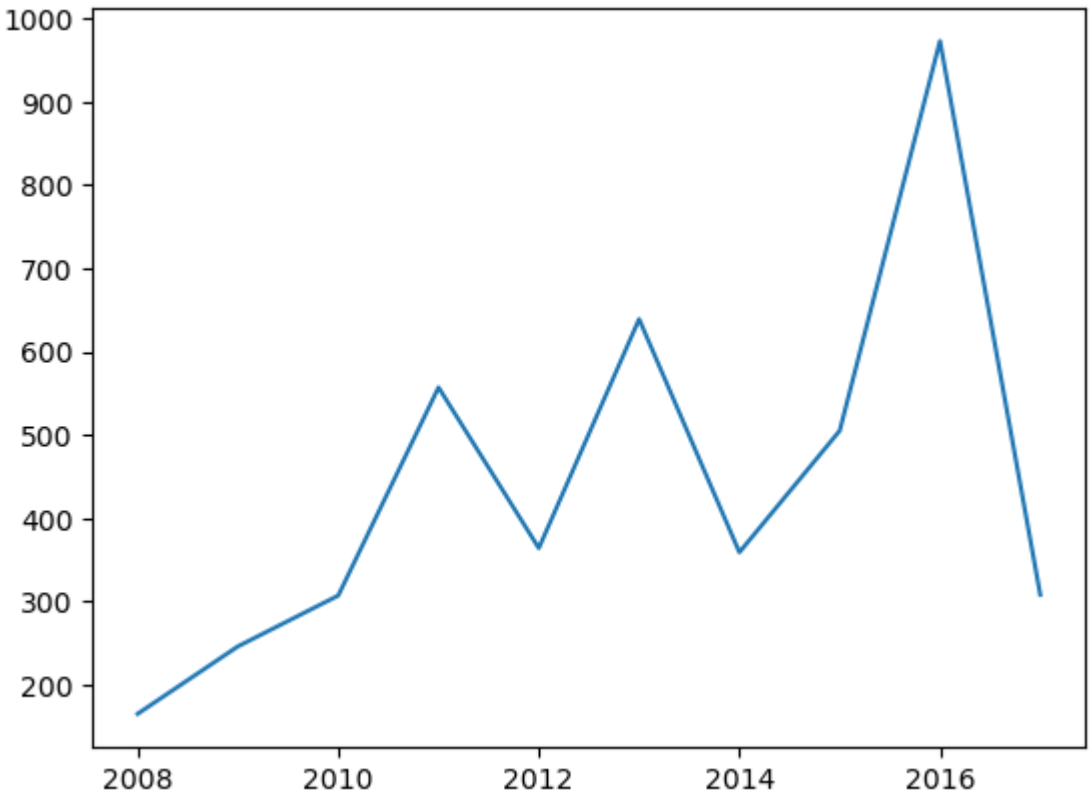
In [ ]:
```
batsman.head()
```

Out[ ]:

	index	RG Sharma	V Kohli
0	2008	404	165
1	2009	362	246
2	2010	404	307
3	2011	372	557
4	2012	433	364

In [ ]:
```
# plot the graph
plt.plot(batsman['index'],batsman['V Kohli'])
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x28b7c40ca60>]
```

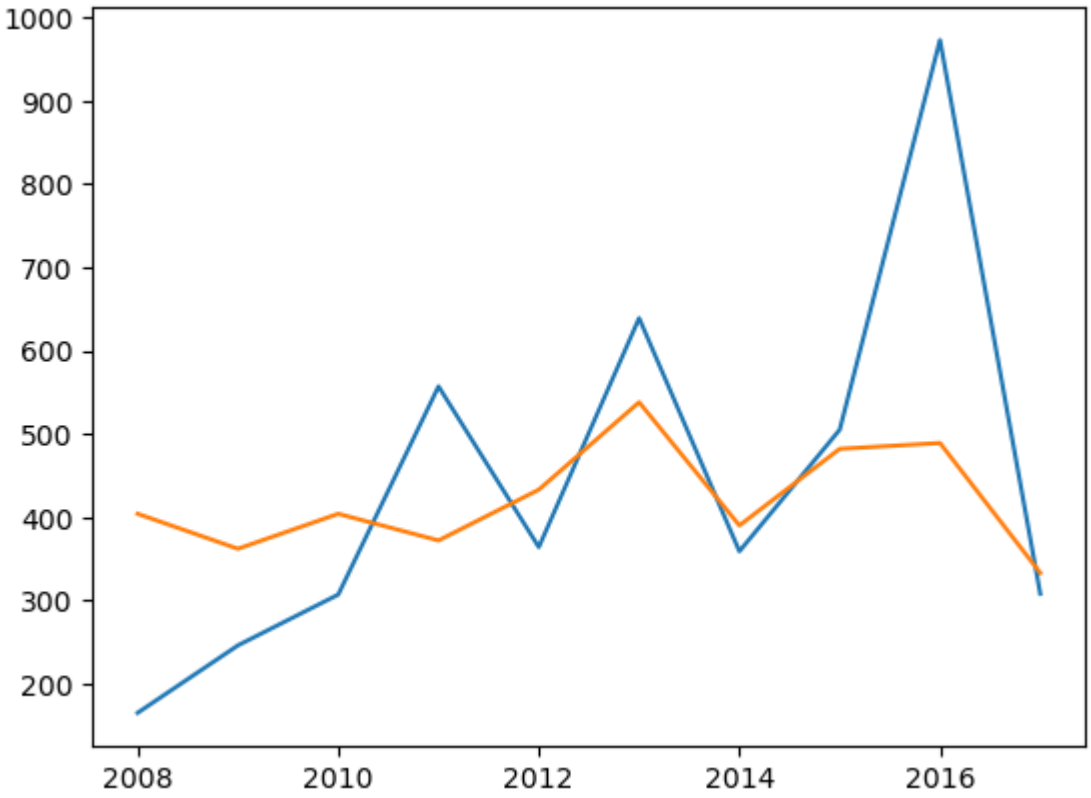


Multiple Plots:

It's possible to create multiple lines on a single plot, making it easy to compare multiple datasets or variables. In the example, both Rohit Sharma's and Virat Kohli's career runs are plotted on the same graph.

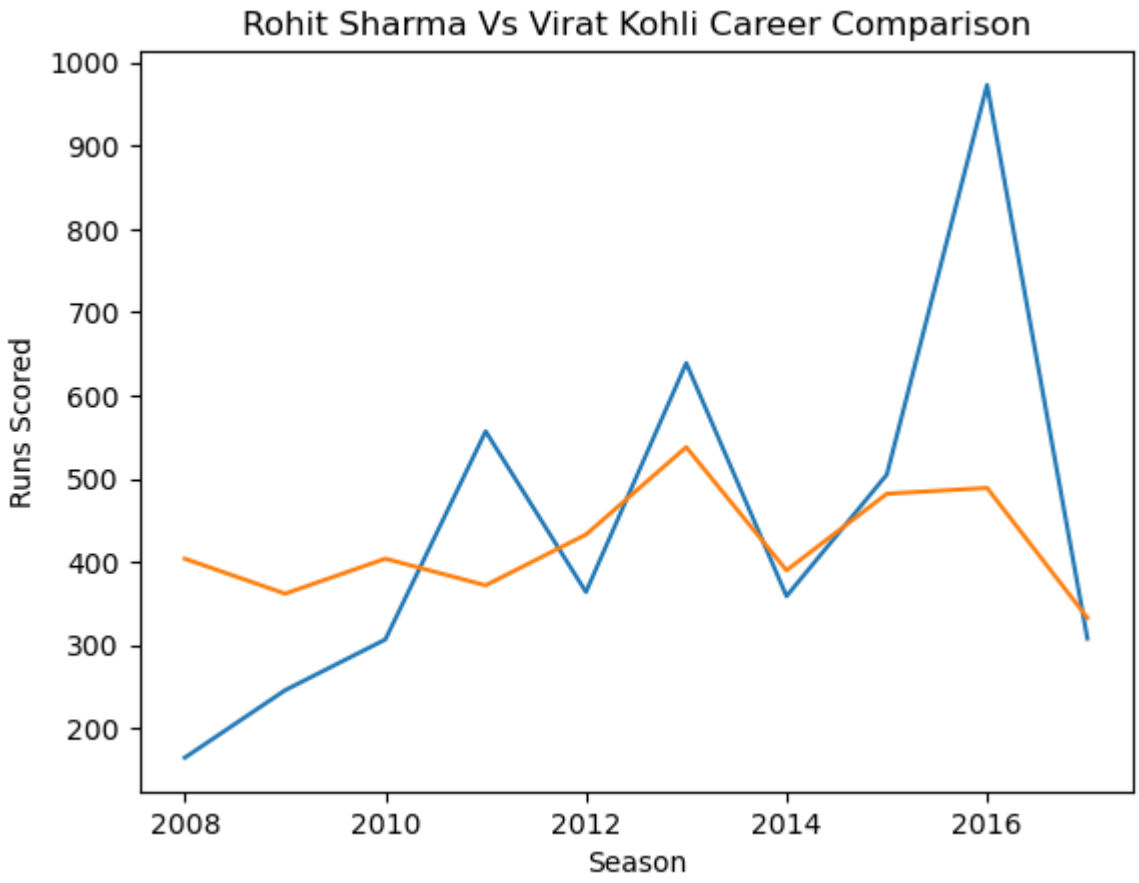
In [ ]:
```
# plotting multiple plots
plt.plot(batsman['index'],batsman['V Kohli'])
plt.plot(batsman['index'],batsman['RG Sharma'])
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x28b7d4e2610>]
```



In [ ]:
```
# Labels title
plt.plot(batsman['index'],batsman['V Kohli'])
plt.plot(batsman['index'],batsman['RG Sharma'])

plt.title('Rohit Sharma Vs Virat Kohli Career Comparison')
plt.xlabel('Season')
plt.ylabel('Runs Scored')
```

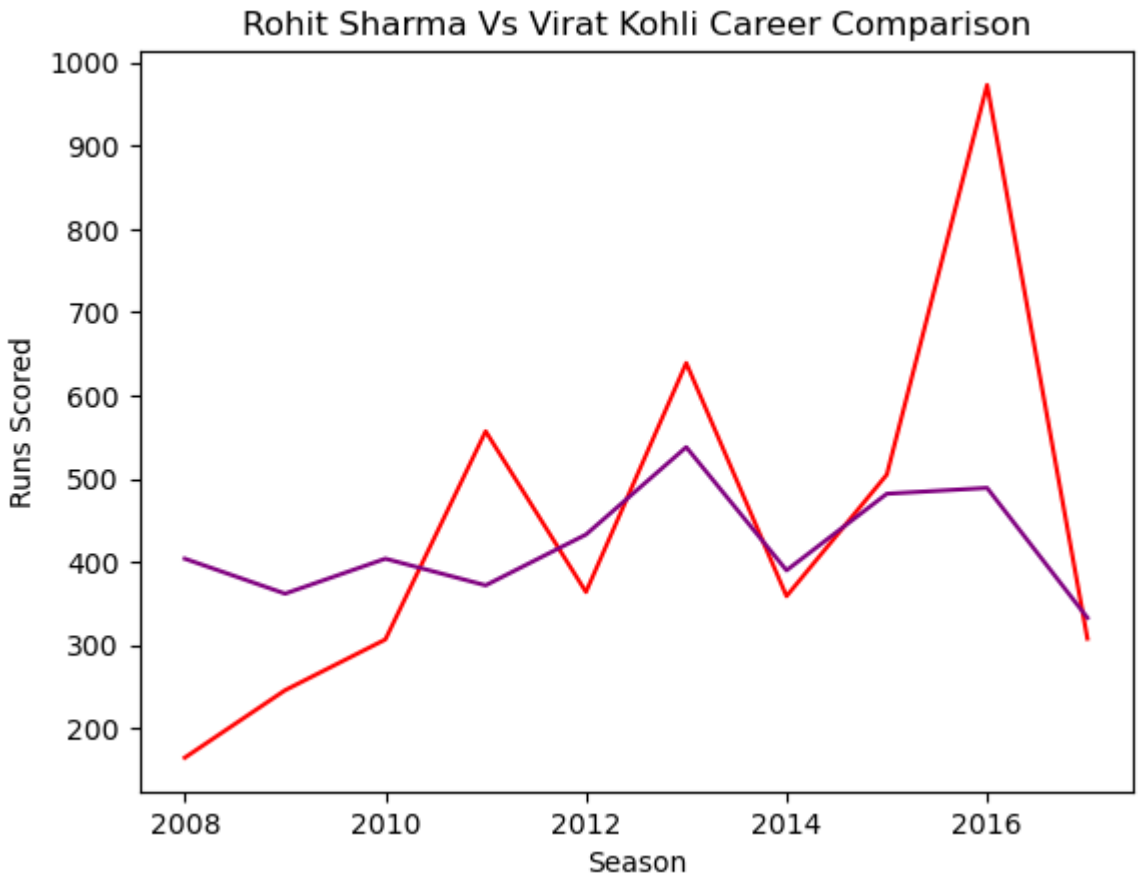
Out[ ]:
```
Text(0, 0.5, 'Runs Scored')
```



colors(hex) and line(width and style) and marker(size)

In [ ]:
```
#colors
plt.plot(batsman['index'],batsman['V Kohli'],color='Red')
plt.plot(batsman['index'],batsman['RG Sharma'],color='Purple')

plt.title('Rohit Sharma Vs Virat Kohli Career Comparison')
plt.xlabel('Season')
plt.ylabel('Runs Scored')
```

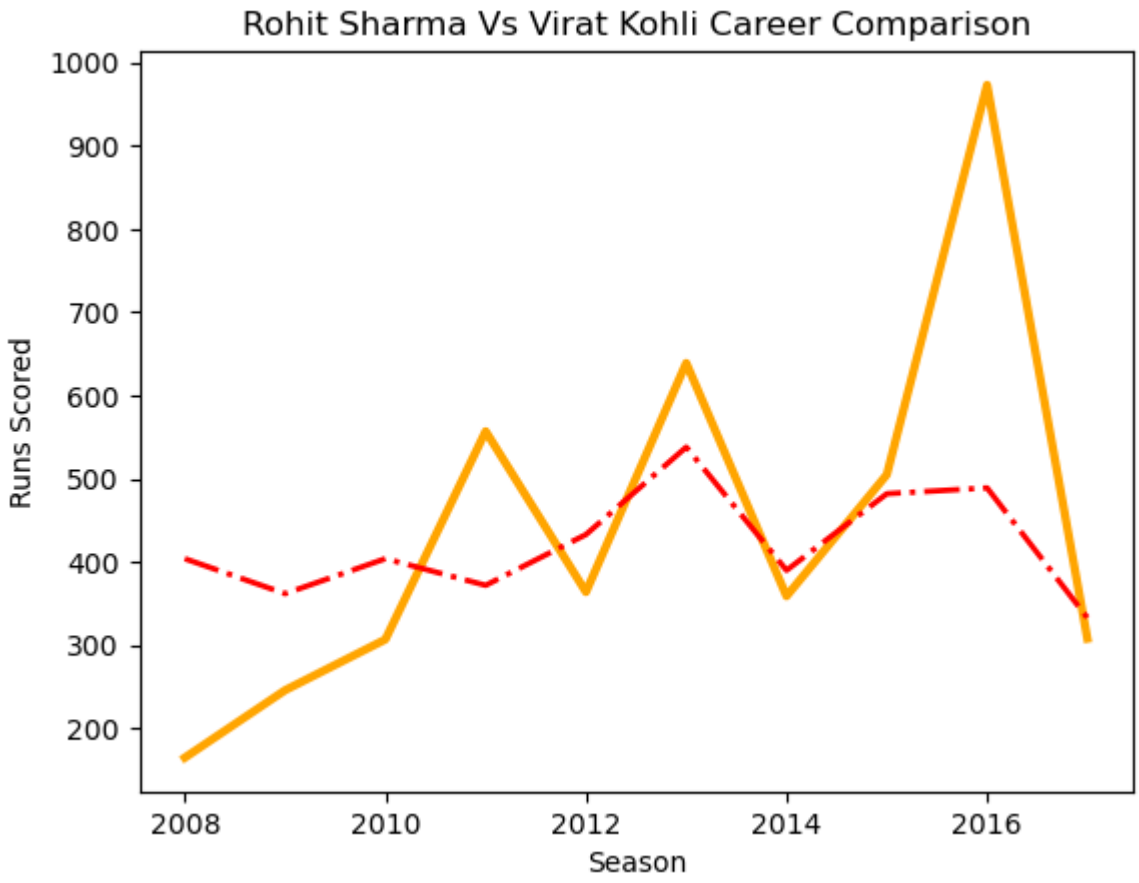
Out[ ]:
```
Text(0, 0.5, 'Runs Scored')
```



You can specify different colors for each line in the plot. In the example, colors like 'Red' and 'Purple' are used to differentiate the lines.

In [ ]:
```
# linestyle and linewidth
plt.plot(batsman['index'],batsman['V Kohli'],color='Orange',linestyle='solid',linew
plt.plot(batsman['index'],batsman['RG Sharma'],color='Red',linestyle='dashdot',line

plt.title('Rohit Sharma Vs Virat Kohli Career Comparison')
plt.xlabel('Season')
plt.ylabel('Runs Scored')
```

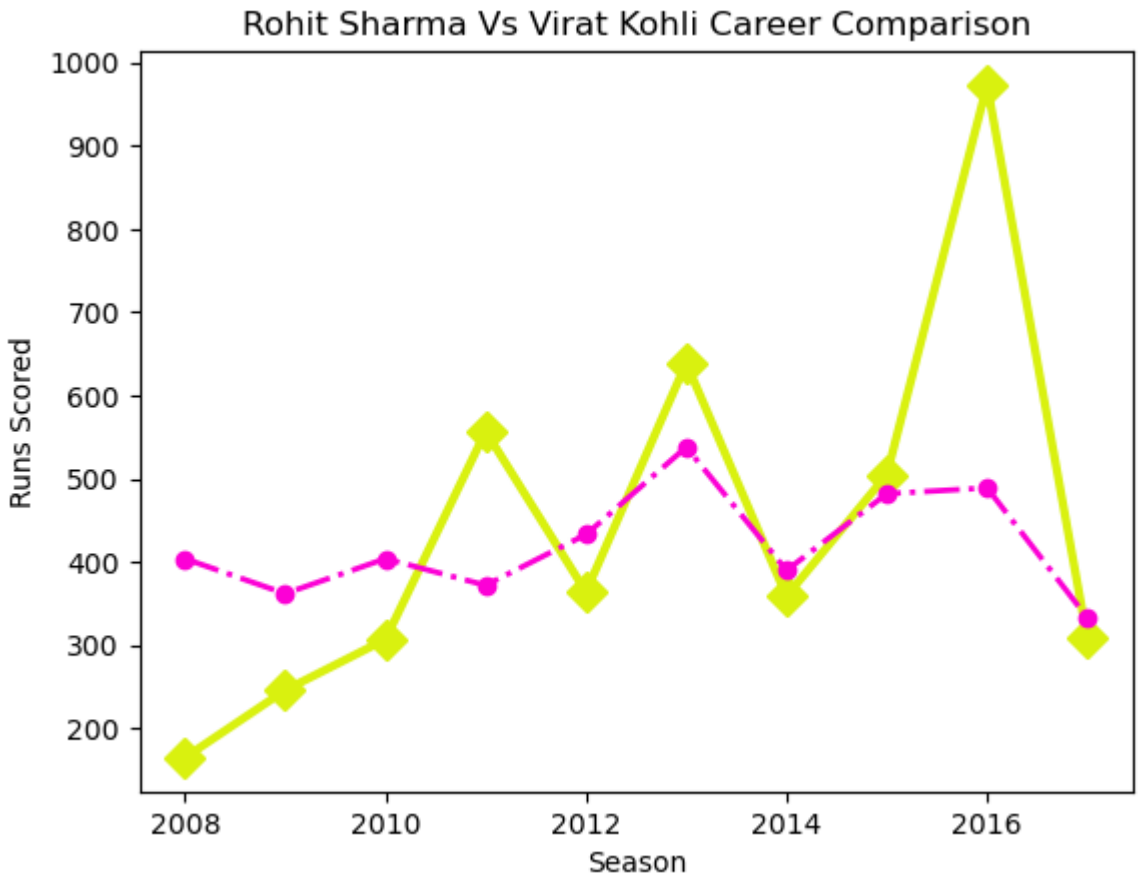
Out[ ]:
```
Text(0, 0.5, 'Runs Scored')
```



You can change the style and width of the lines. Common line styles include 'solid,' 'dotted,' 'dashed,' etc. In the example, 'solid' and 'dashdot' line styles are used.

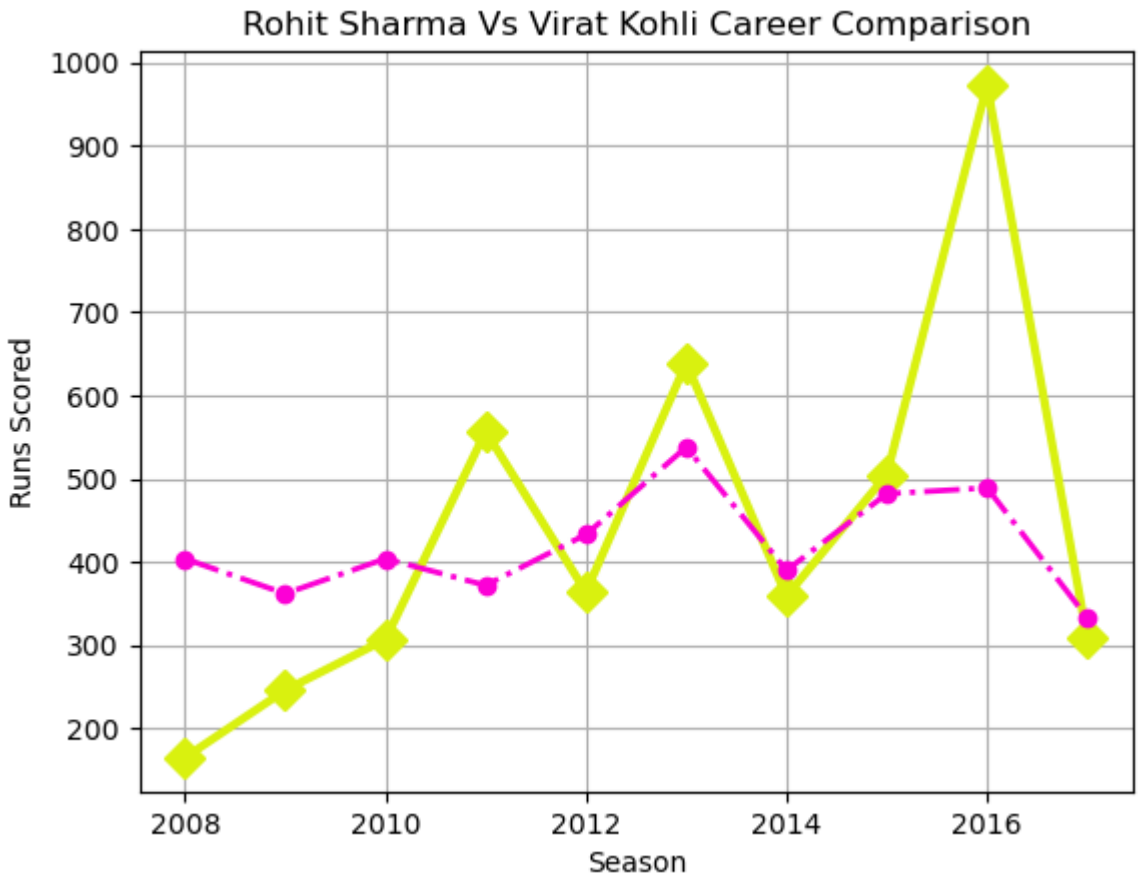
In [ ]:
```
# Marker
plt.plot(batsman['index'],batsman['V Kohli'],color='#D9F10F',linestyle='solid',line
plt.plot(batsman['index'],batsman['RG Sharma'],color='#FC00D6',linestyle='dashdot',

plt.title('Rohit Sharma Vs Virat Kohli Career Comparison')
plt.xlabel('Season')
plt.ylabel('Runs Scored')
```

Out[ ]:
```
Text(0, 0.5, 'Runs Scored')
```



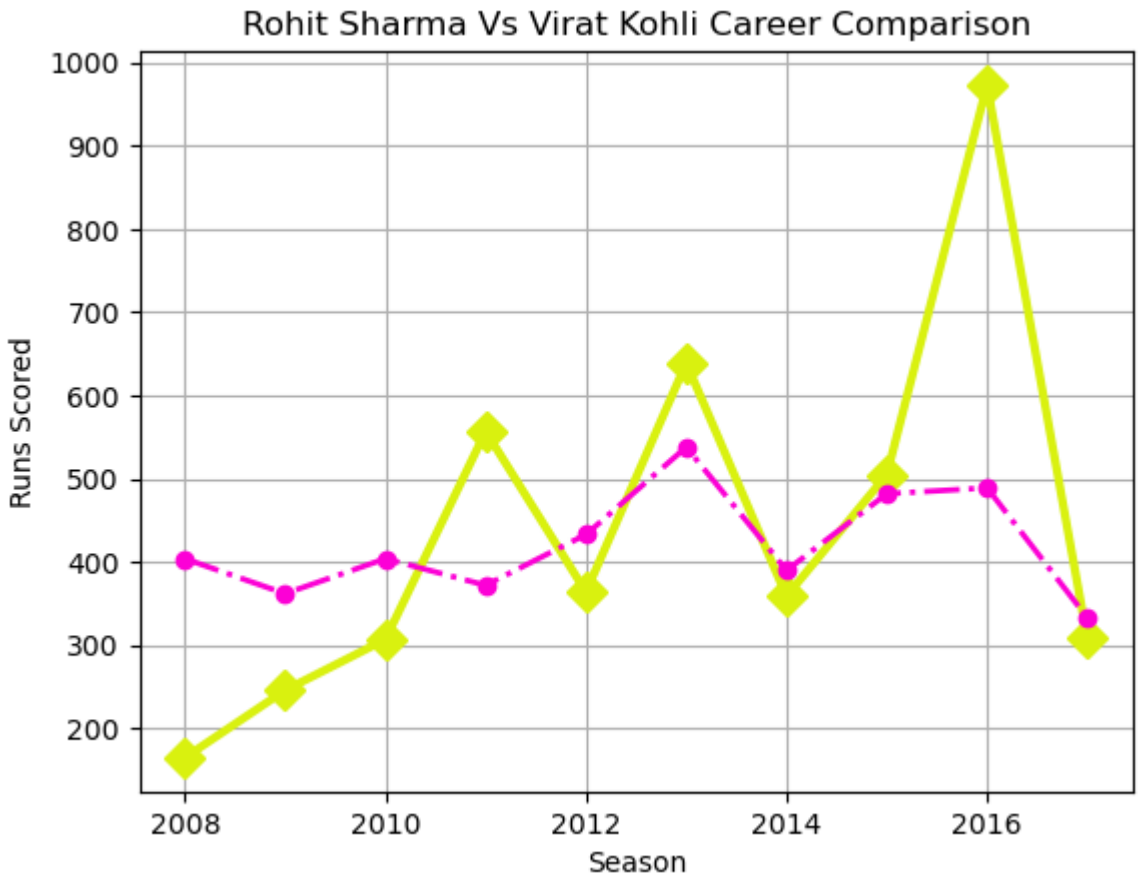
Markers are used to highlight data points on the line plot. You can customize markers' style and size. In the example, markers like 'D' and 'o' are used with different colors.

In [ ]:
```
# grid
plt.plot(batsman['index'],batsman['V Kohli'],color='#D9F10F',linestyle='solid',line
plt.plot(batsman['index'],batsman['RG Sharma'],color='#FC00D6',linestyle='dashdot',

plt.title('Rohit Sharma Vs Virat Kohli Career Comparison')
plt.xlabel('Season')
plt.ylabel('Runs Scored')

plt.grid()
```



Adding a grid to the plot can make it easier to read and interpret the data. The grid helps in aligning the data points with the tick marks on the axes.

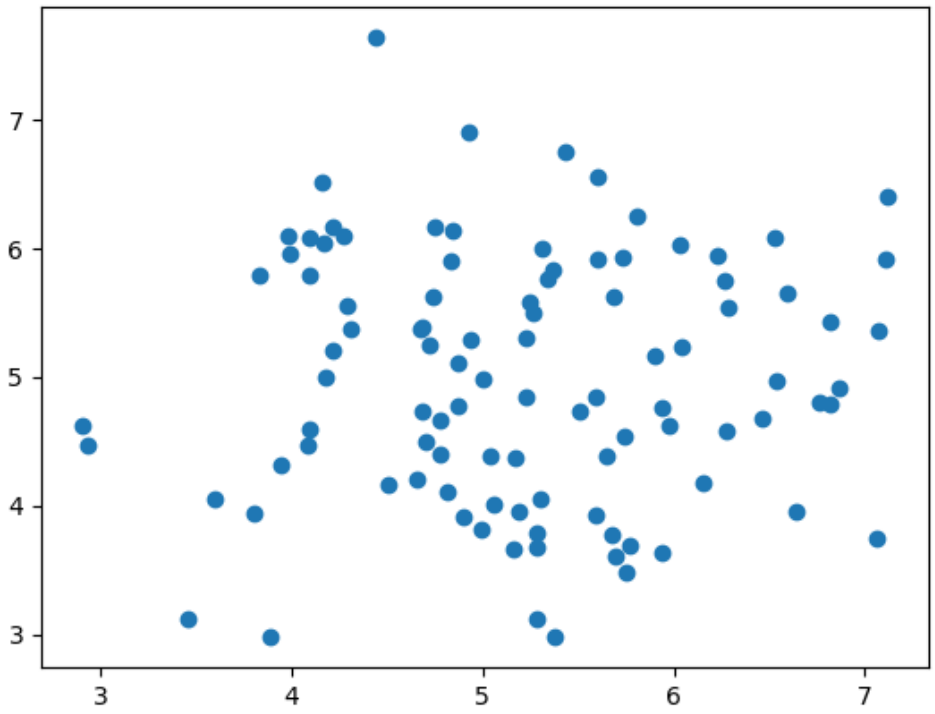
In [ ]:
```
# show
plt.plot(batsman['index'],batsman['V Kohli'],color='#D9F10F',linestyle='solid',line
plt.plot(batsman['index'],batsman['RG Sharma'],color='#FC00D6',linestyle='dashdot',

plt.title('Rohit Sharma Vs Virat Kohli Career Comparison')
plt.xlabel('Season')
plt.ylabel('Runs Scored')

plt.grid()

plt.show()
```



After customizing your plot, you can use plt.show() to display it. This command is often used in Jupyter notebooks or standalone Python scripts.

2D line plots are valuable for visualizing time series data, comparing trends in multiple datasets, and exploring the relationship between two continuous variables. Customization options in Matplotlib allow you to create visually appealing and informative plots for data analysis and presentation.

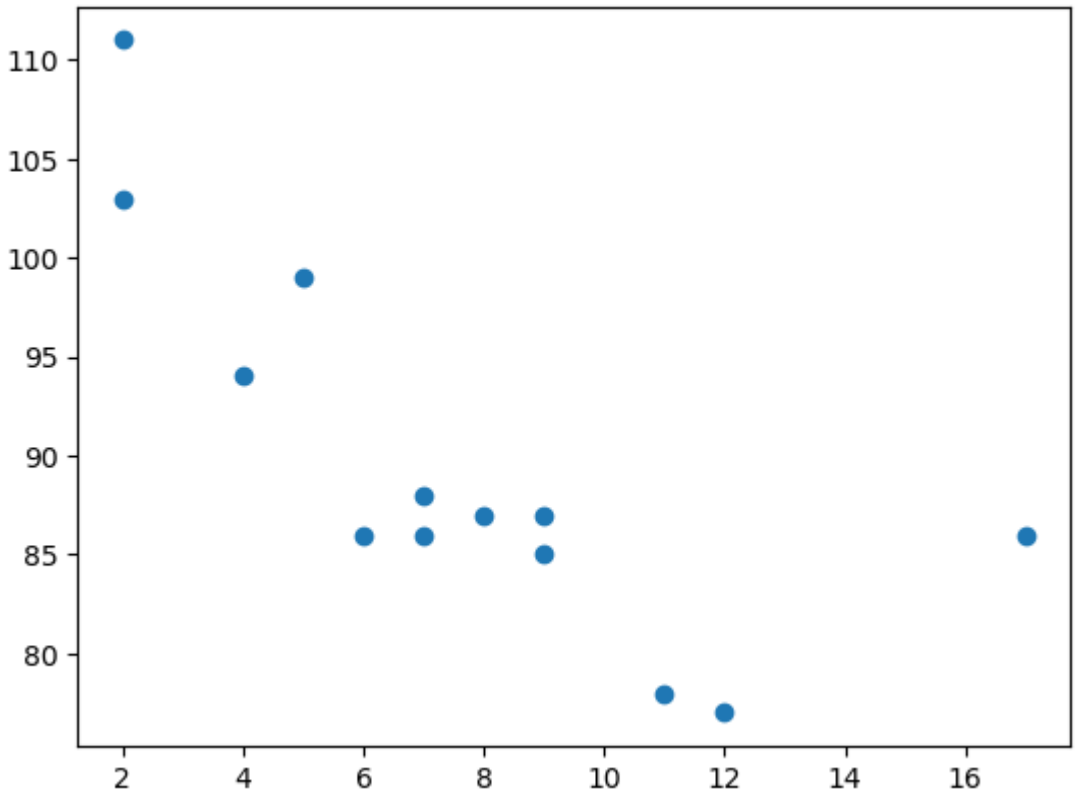
ScatterPlot and Bar Chart in Matplotlib



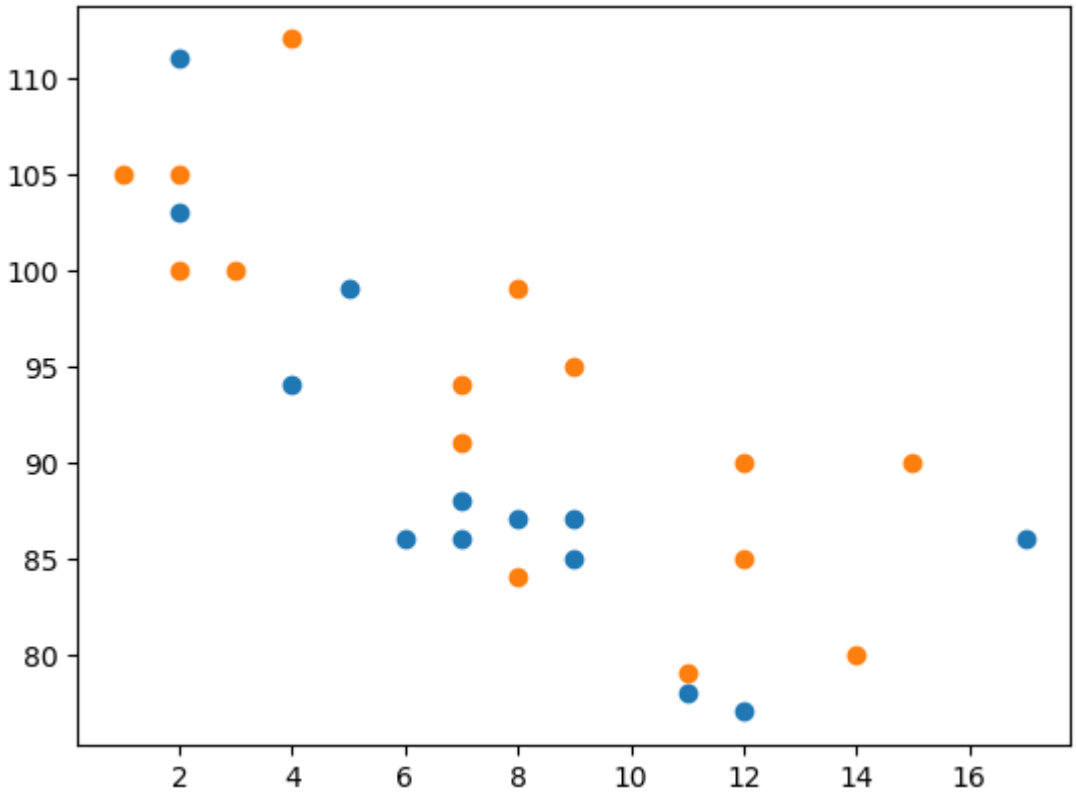
A scatter plot, also known as a scatterplot or scatter chart, is a type of data visualization used in statistics and data analysis. It's used to display the relationship between two variables by representing individual data points as points on a two-dimensional graph. Each point on the plot corresponds to a single data entry with values for both variables, making it a useful tool for identifying patterns, trends, clusters, or outliers in data.

- Bivariate Analysis
- numerical vs numerical
- Use case - Finding correlation

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```



In [ ]:
```
#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()
```

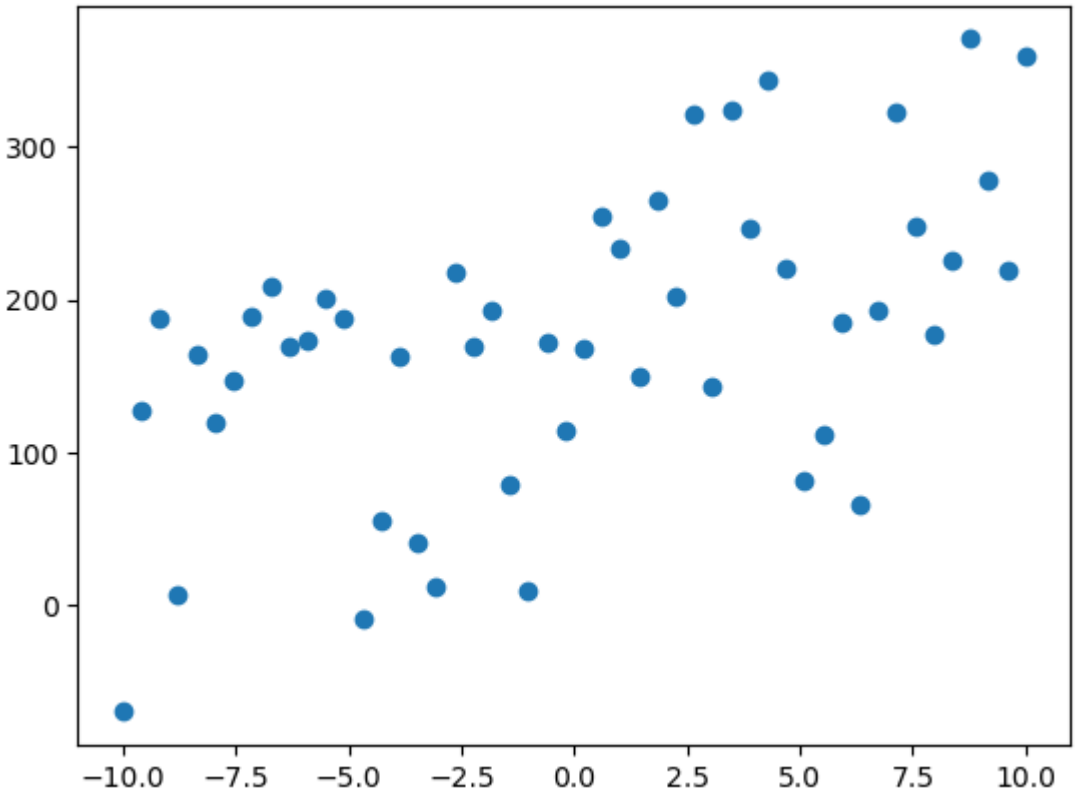


In [ ]:
```
# plt.scatter simple function
x = np.linspace(-10,10,50)

y = 10*x + 3 + np.random.randint(0,300,50)
y
```

Out[ ]:
```
array([-70.        , 127.08163265, 187.16326531,   6.24489796,
       164.32653061, 119.40816327, 146.48979592, 189.57142857,
       208.65306122, 169.73469388, 173.81632653, 200.89795918,
       187.97959184,  -8.93877551,  55.14285714, 162.2244898 ,
        40.30612245,  12.3877551 , 218.46938776, 169.55102041,
       193.63265306,  78.71428571,   9.79591837, 171.87755102,
       113.95918367, 168.04081633, 255.12244898, 233.20408163,
       150.28571429, 265.36734694, 202.44897959, 321.53061224,
       142.6122449 , 324.69387755, 246.7755102 , 343.85714286,
       220.93877551,  81.02040816, 111.10204082, 185.18367347,
        66.26530612, 193.34693878, 323.42857143, 248.51020408,
       177.59183673, 225.67346939, 370.75510204, 277.83673469,
       218.91836735, 360.        ])
```

In [ ]:
```
plt.scatter(x,y)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x264627ccc70>
```



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

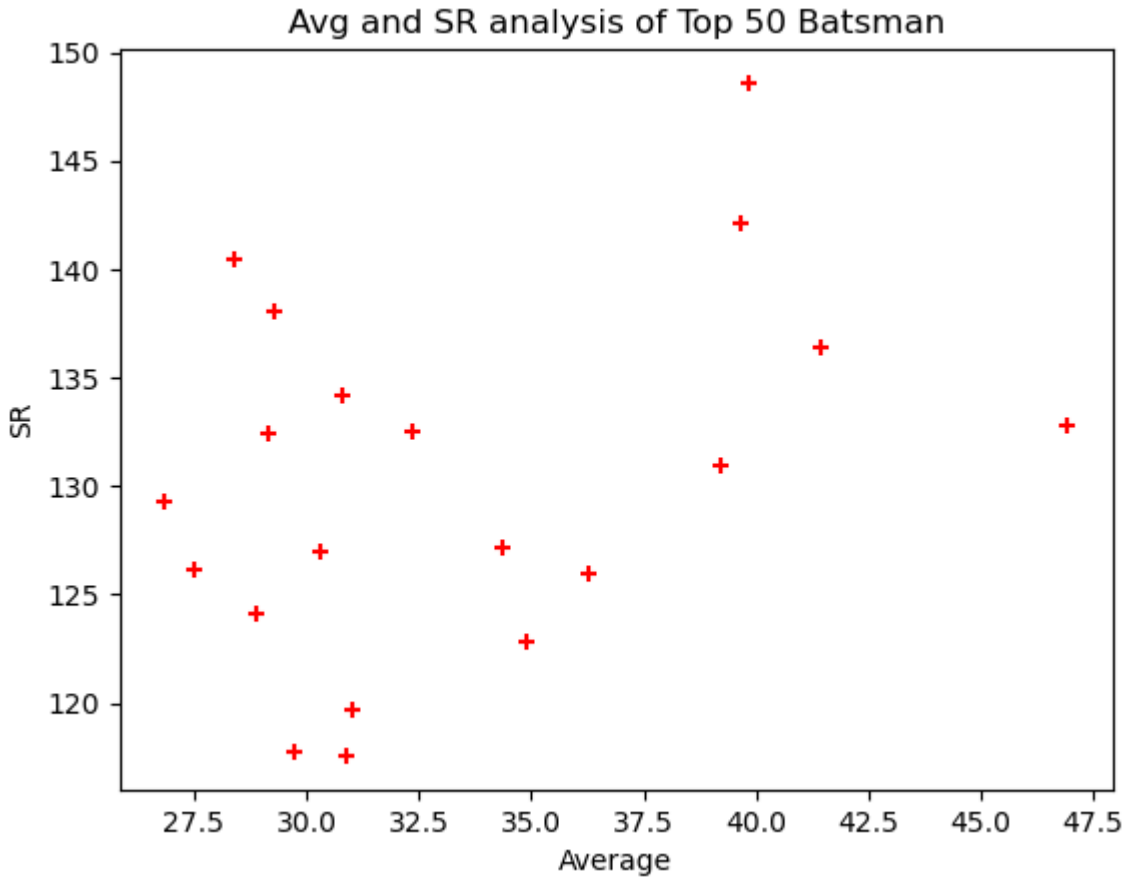
In [ ]:
```
# plt.scatter on pandas data
df = pd.read_csv('Data\Day47\Batter.csv')
df = df.head(20)
df
```

Out[ ]:

	batter	runs	avg	strike_rate
0	V Kohli	6634	36.251366	125.977972
1	S Dhawan	6244	34.882682	122.840842
2	DA Warner	5883	41.429577	136.401577
3	RG Sharma	5881	30.314433	126.964594
4	SK Raina	5536	32.374269	132.535312
5	AB de Villiers	5181	39.853846	148.580442
6	CH Gayle	4997	39.658730	142.121729
7	MS Dhoni	4978	39.196850	130.931089
8	RV Uthappa	4954	27.522222	126.152279
9	KD Karthik	4377	26.852761	129.267572
10	G Gambhir	4217	31.007353	119.665153
11	AT Rayudu	4190	28.896552	124.148148
12	AM Rahane	4074	30.863636	117.575758
13	KL Rahul	3895	46.927711	132.799182
14	SR Watson	3880	30.793651	134.163209
15	MK Pandey	3657	29.731707	117.739858
16	SV Samson	3526	29.140496	132.407060
17	KA Pollard	3437	28.404959	140.457703
18	F du Plessis	3403	34.373737	127.167414
19	YK Pathan	3222	29.290909	138.046272

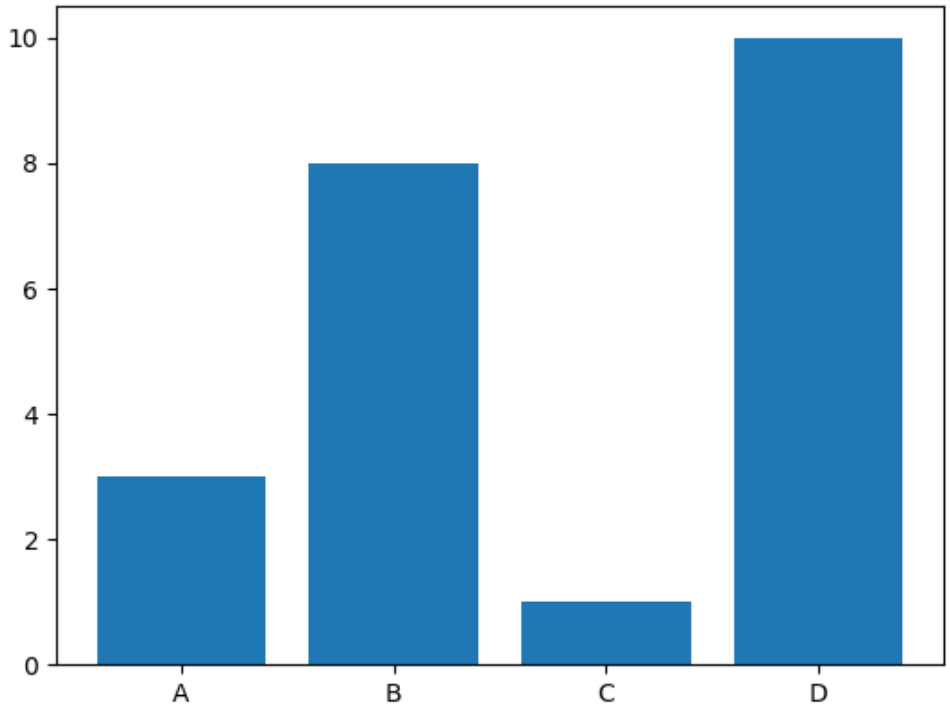
In [ ]:
```
# marker
plt.scatter(df['avg'],df['strike_rate'],color='red',marker='+')
plt.title('Avg and SR analysis of Top 50 Batsman')
plt.xlabel('Average')
plt.ylabel('SR')
```

Out[ ]:
```
Text(0, 0.5, 'SR')
```



Scatter plots are particularly useful for visualizing the distribution of data, identifying correlations or relationships between variables, and spotting outliers. You can adjust the appearance and characteristics of the scatter plot to suit your needs, including marker size, color, and transparency. This makes scatter plots a versatile tool for data exploration and analysis.

Bar plot

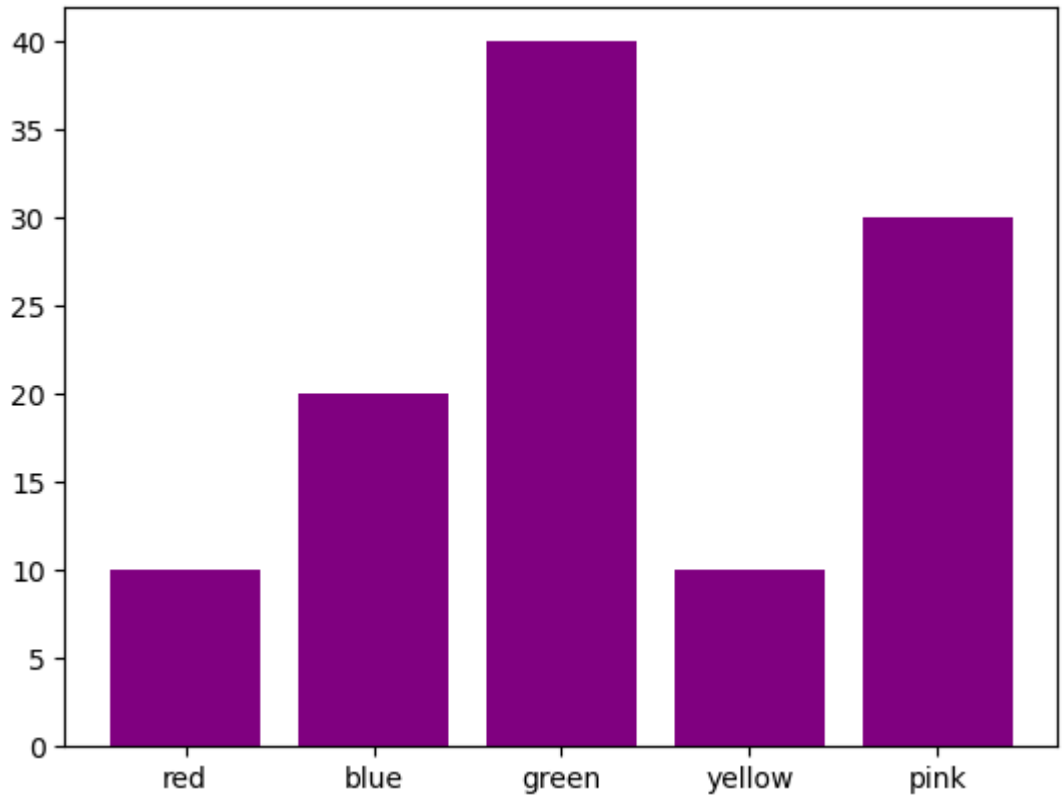


A bar plot, also known as a bar chart or bar graph, is a type of data visualization that is used to represent categorical data with rectangular bars. Each bar's length or height is proportional to the value it represents. Bar plots are typically used to compare and display the relative sizes or quantities of different categories or groups.

- Bivariate Analysis
- Numerical vs Categorical
- Use case - Aggregate analysis of groups

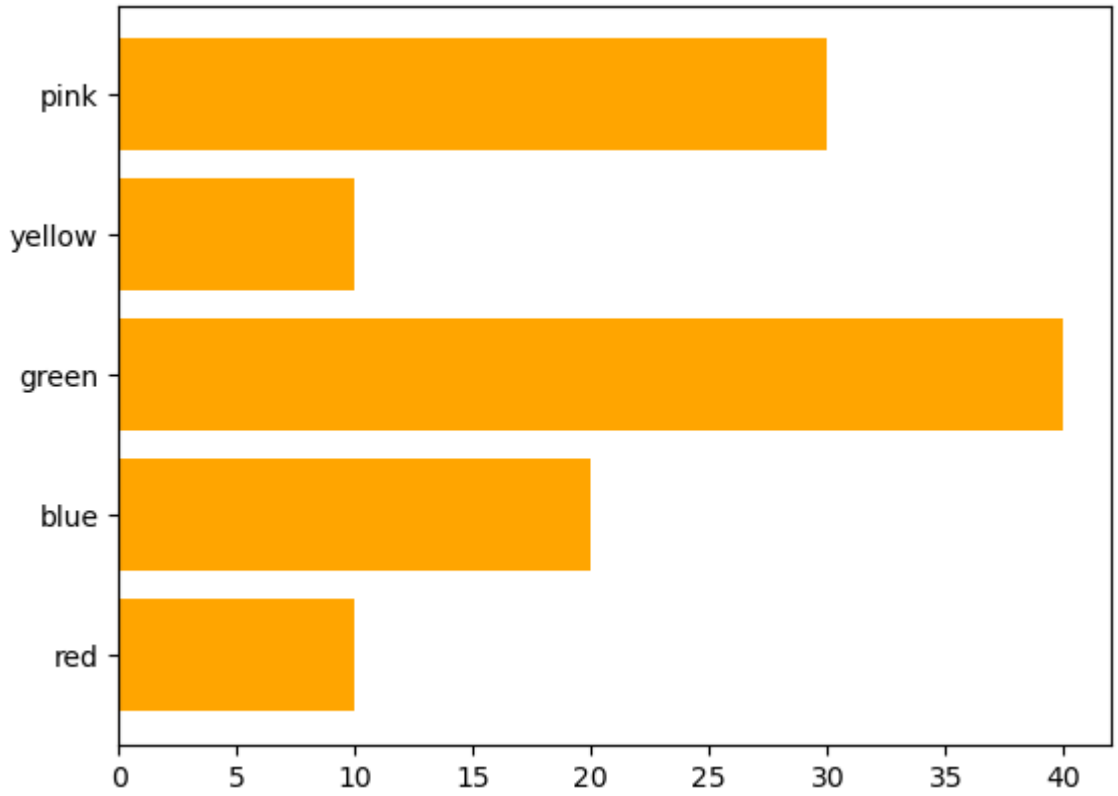
In [ ]:
```
# simple bar chart
children = [10,20,40,10,30]
colors = ['red','blue','green','yellow','pink']

plt.bar(colors,children,color='Purple')
```

Out[ ]:
```
<BarContainer object of 5 artists>
```



In [ ]:
```
# horizontal bar chart
plt.barh(colors,children,color='Orange')
```

Out[ ]:
```
<BarContainer object of 5 artists>
```

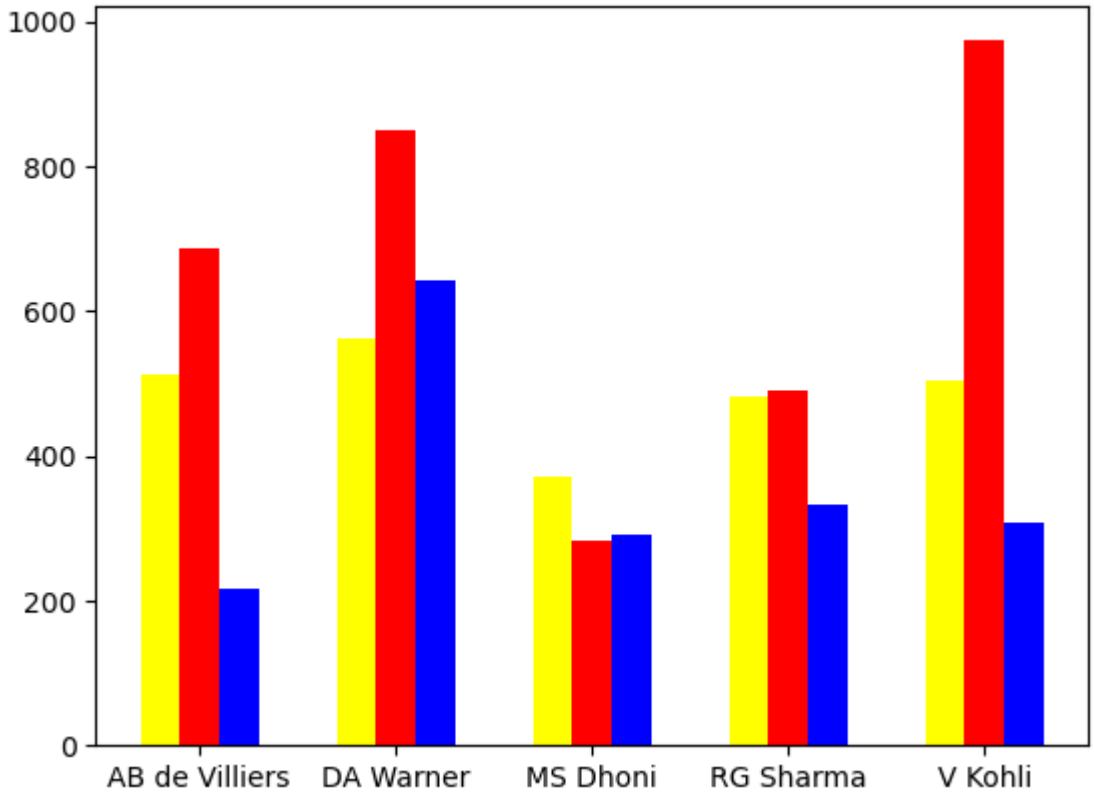


In [ ]:
```
# color and label
df = pd.read_csv('Data\Day47\Batsman_season.csv')
df
```

Out[ ]:

	batsman	2015	2016	2017
0	AB de Villiers	513	687	216
1	DA Warner	562	848	641
2	MS Dhoni	372	284	290
3	RG Sharma	482	489	333
4	V Kohli	505	973	308

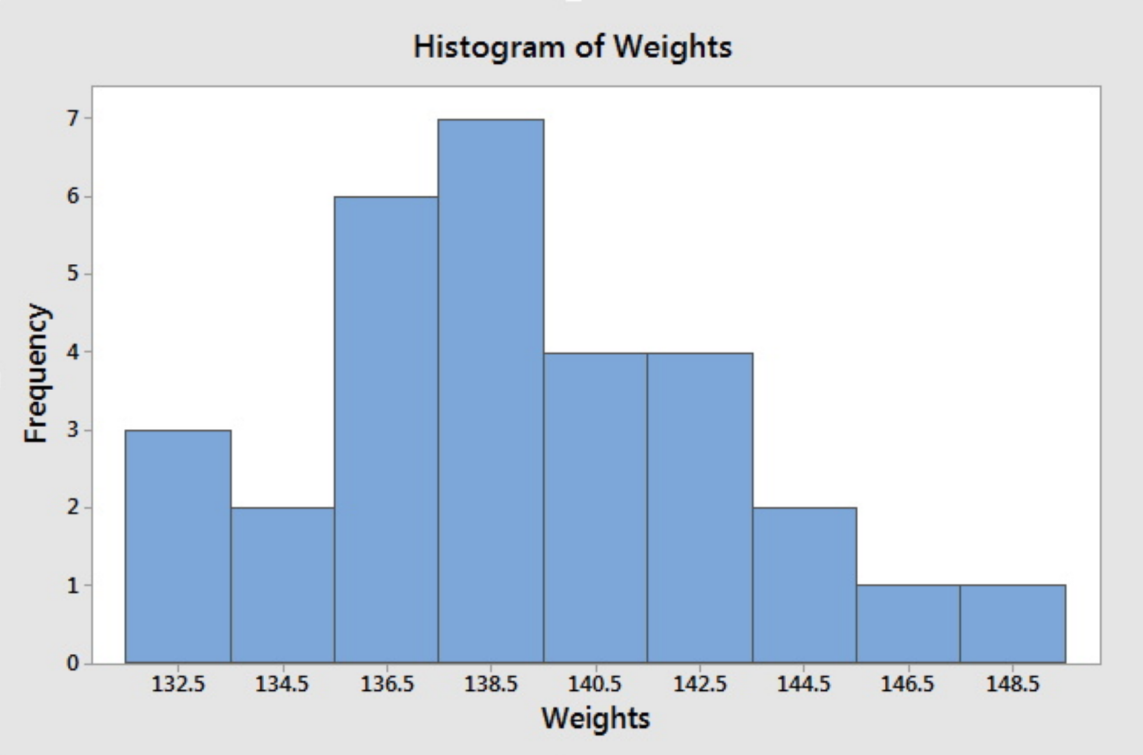
In [ ]:
```
plt.bar(np.arange(df.shape[0]) - 0.2,df['2015'],width=0.2,color='yellow')
plt.bar(np.arange(df.shape[0]),df['2016'],width=0.2,color='red')
plt.bar(np.arange(df.shape[0]) + 0.2,df['2017'],width=0.2,color='blue')

plt.xticks(np.arange(df.shape[0]), df['batsman'])

plt.show()
```



Bar plots are useful for comparing the values of different categories and for showing the distribution of data within each category. They are commonly used in various fields, including business, economics, and data analysis, to make comparisons and convey information about categorical data. You can customize bar plots to make them more visually appealing and informative.

Histogram and Pie chart in Matplotlib

Histogram



A histogram is a type of chart that shows the distribution of numerical data. It's a graphical representation of data where data is grouped into continuous number ranges and each range corresponds to a vertical bar. The horizontal axis displays the number range, and the vertical axis (frequency) represents the amount of data that is present in each range.

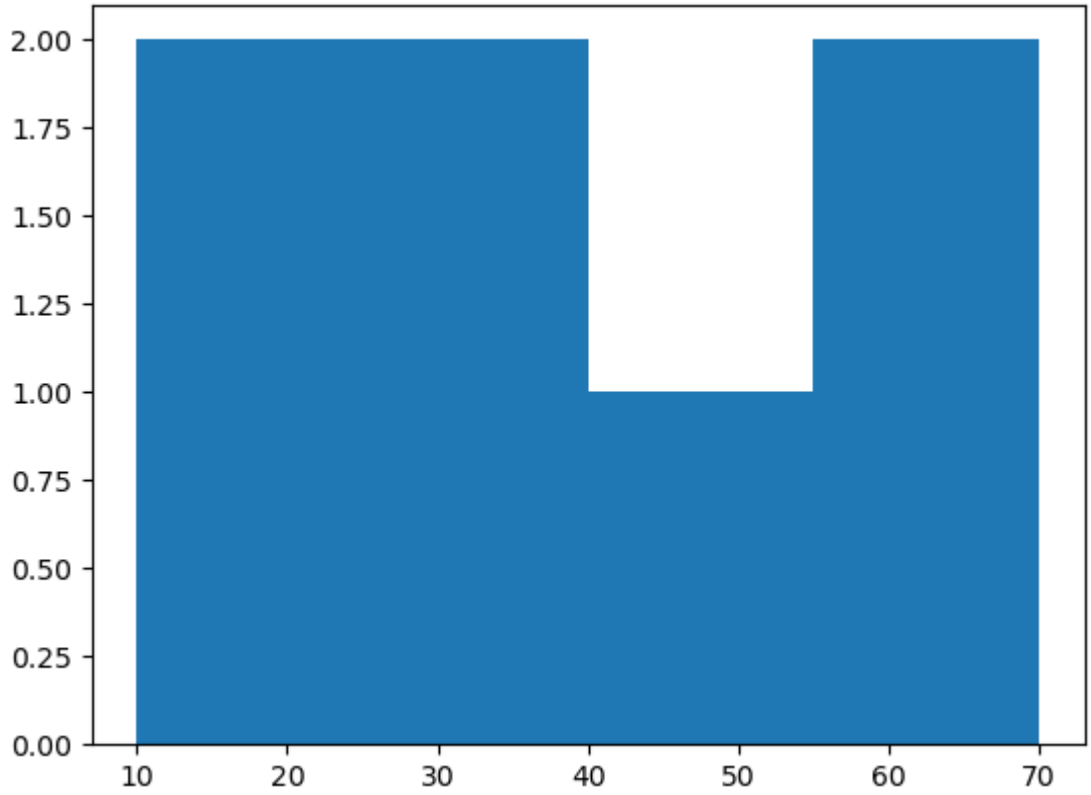
A histogram is a set of rectangles with bases along with the intervals between class boundaries and with areas proportional to frequencies in the corresponding classes. The x-axis of the graph represents the class interval, and the y-axis shows the various frequencies corresponding to different class intervals. A histogram is a type of data visualization used to represent the distribution of a dataset, especially when dealing with continuous or numeric data. It displays the frequency or count of data points falling into specific intervals or "bins" along a continuous range. Histograms provide insights into the shape, central tendency, and spread of a dataset.

- Univariate Analysis
- Numerical col
- Use case - Frequency Count

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
# simple data

data = [32,45,56,10,15,27,61]

plt.hist(data,bins=[10,25,40,55,70])
```

Out[ ]:
```
(array([2., 2., 1., 2.]),
 array([10., 25., 40., 55., 70.]),
 <BarContainer object of 4 artists>)
```



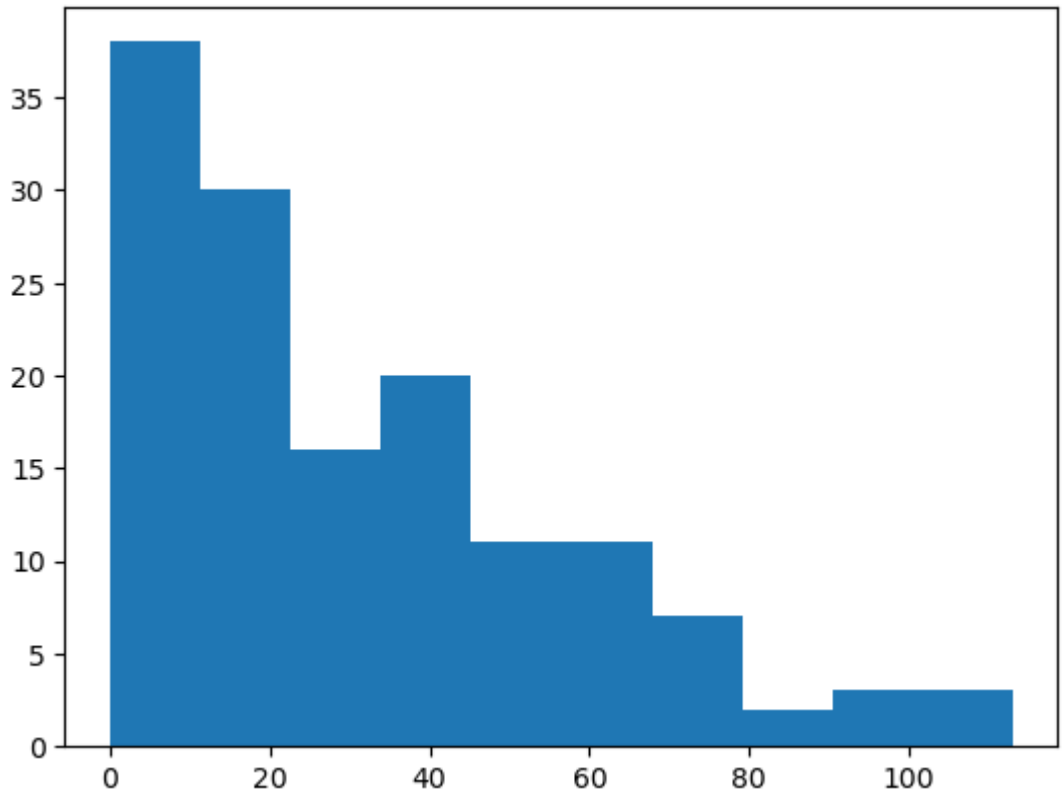
In [ ]:
```
# on some data
df = pd.read_csv('Data\Day48\Vk.csv')
df
```

Out[ ]:

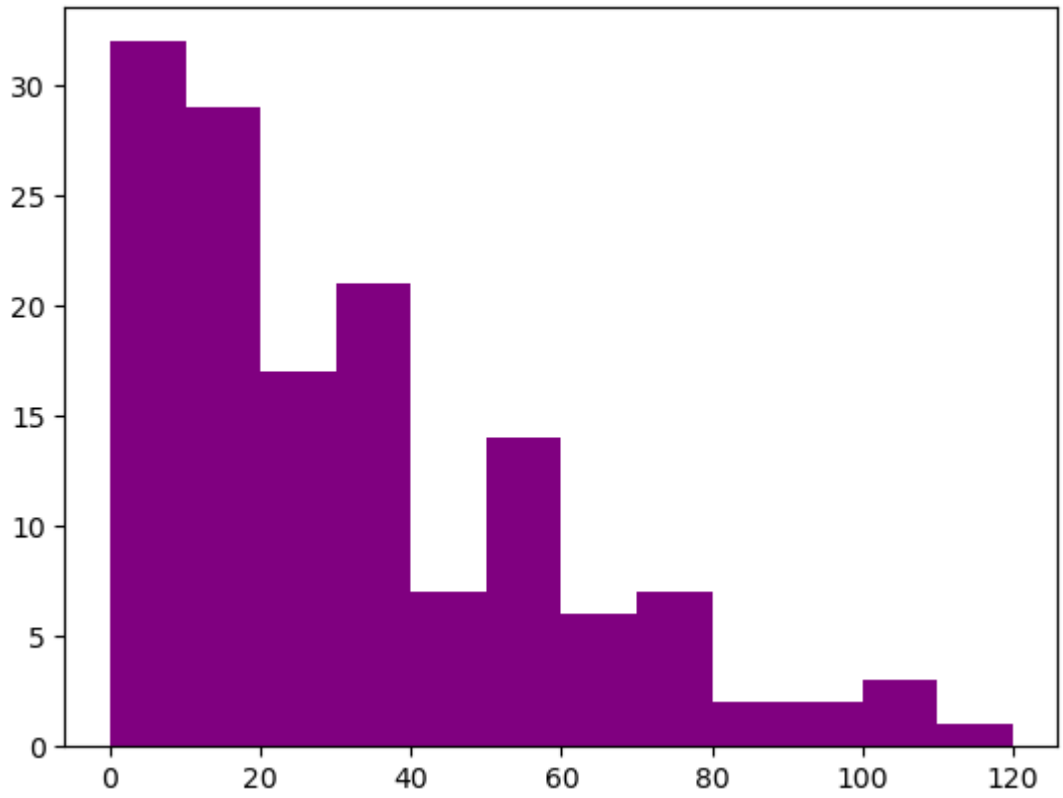
	match_id	batsman_runs
0	12	62
1	17	28
2	20	64
3	27	0
4	30	10
...	...	...
136	624	75
137	626	113
138	632	54
139	633	0
140	636	54

141 rows × 2 columns

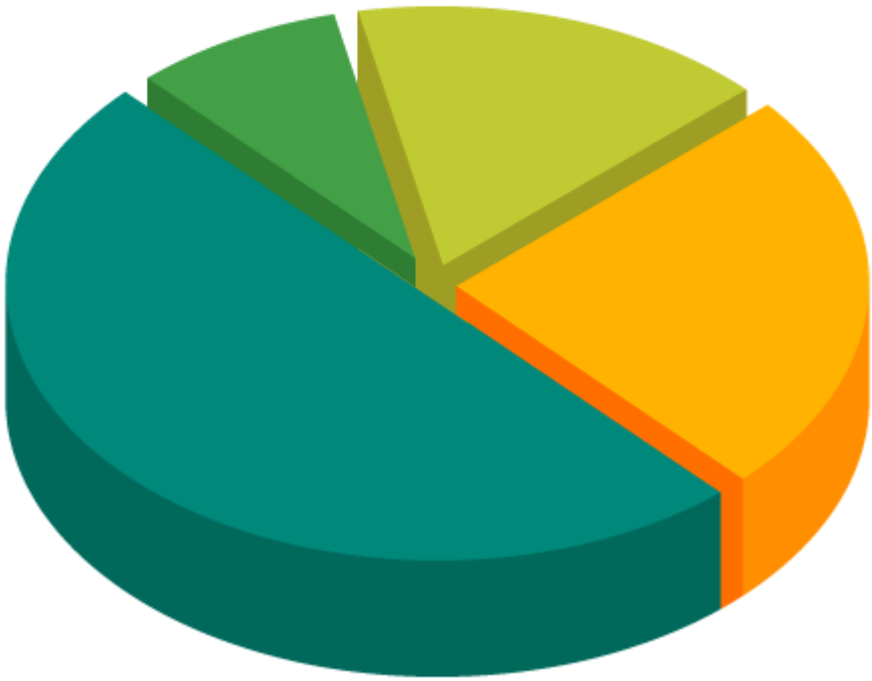
In [ ]:
```
plt.hist(df['batsman_runs'])
plt.show()
```



In [ ]:
```
# handling bins
plt.hist(df['batsman_runs'],bins=[0,10,20,30,40,50,60,70,80,90,100,110,120],color='
plt.show()
```



Pie Chart

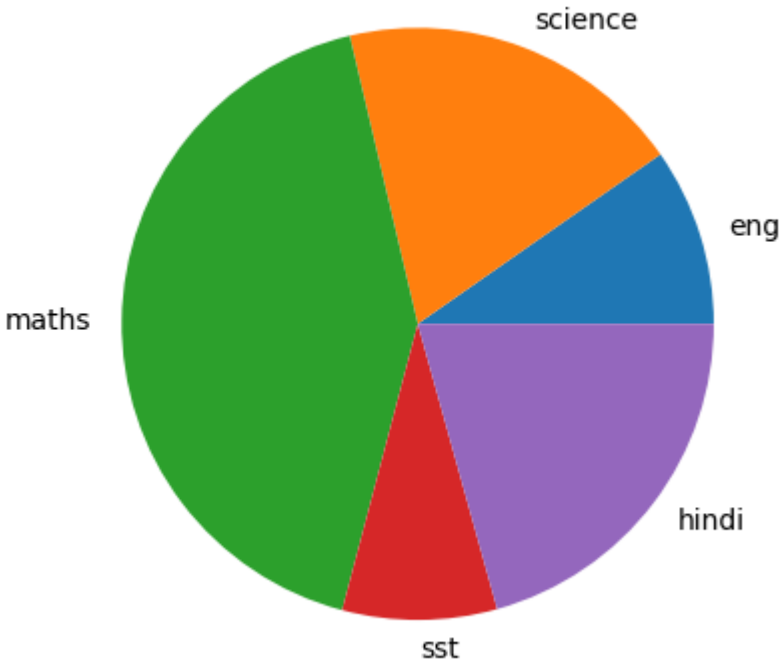


A pie chart is a circular graph that's divided into slices to illustrate numerical proportion. The slices of the pie show the relative size of the data. The arc length of each slice, and consequently its central angle and area, is proportional to the quantity it represents.

All slices of the pie add up to make the whole equaling 100 percent and 360 degrees. Pie charts are often used to represent sample data. Each of these categories is represented as a “slice of the pie”. The size of each slice is directly proportional to the number of data points that belong to a particular category.

- Univariate/Bivariate Analysis
- Categorical vs numerical
- Use case - To find contibution on a standard scale

In [ ]:
```
# simple data
data = [23,45,100,20,49]
subjects = ['eng','science','maths','sst','hindi']
plt.pie(data,labels=subjects)

plt.show()
```

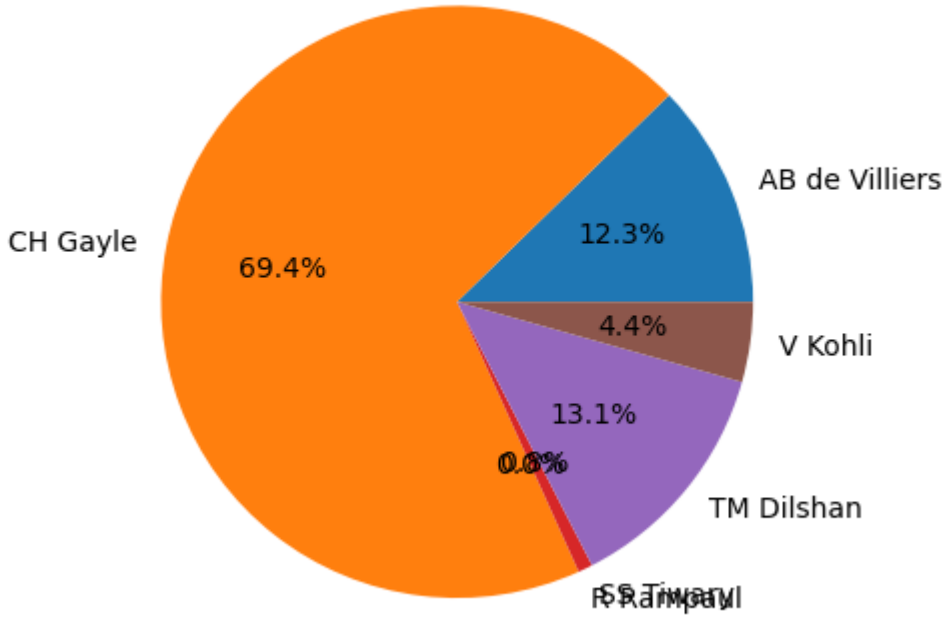


In [ ]:
```
# dataset
df = pd.read_csv('Data\Day48\Gayle-175.csv')
df
```

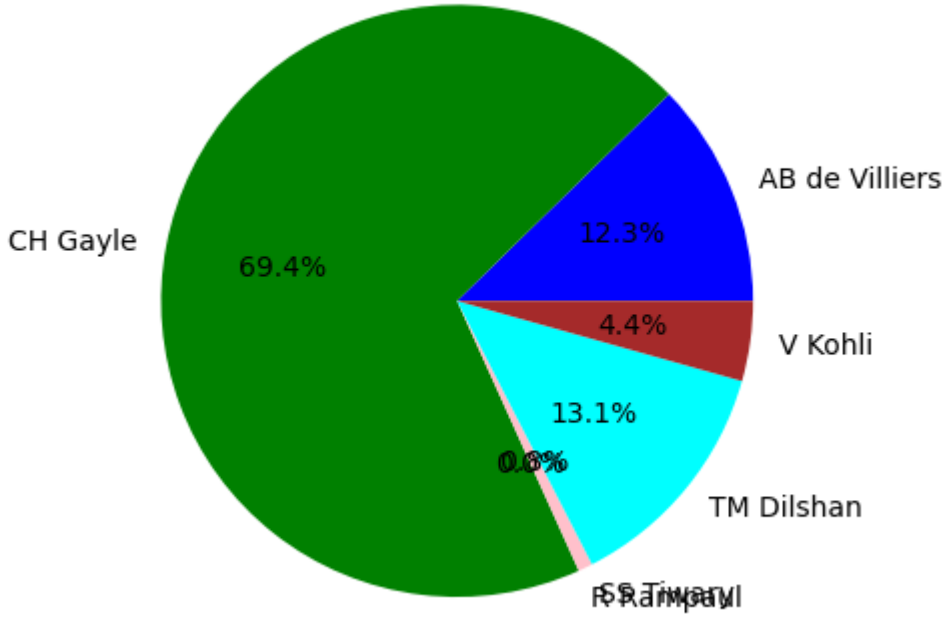
Out[ ]:

	batsman	batsman_runs
0	AB de Villiers	31
1	CH Gayle	175
2	R Rampaul	0
3	SS Tiwary	2
4	TM Dilshan	33
5	V Kohli	11

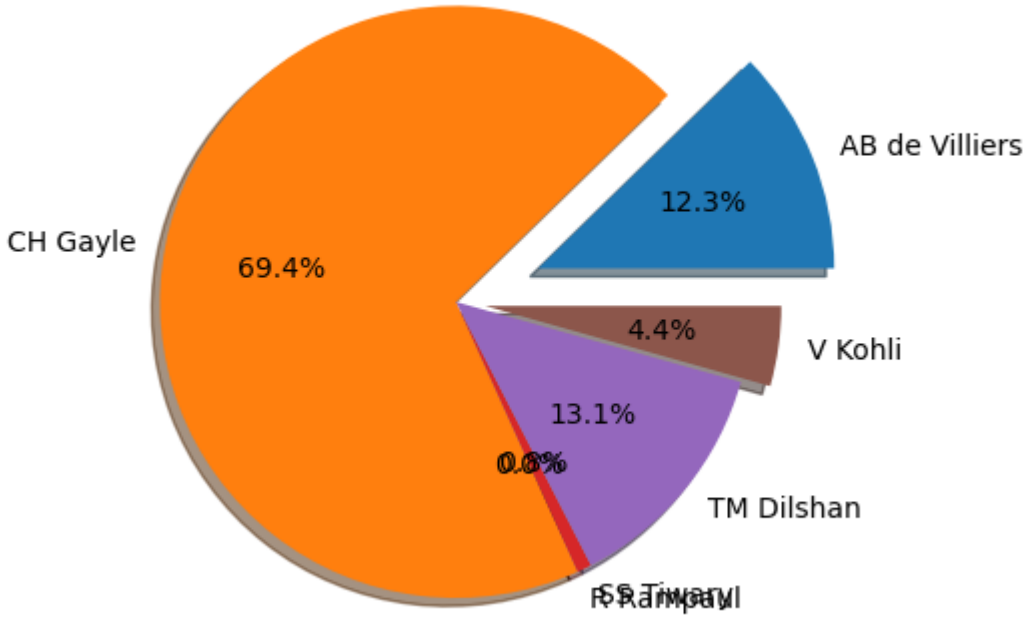
In [ ]:
```
plt.pie(df['batsman_runs'],labels=df['batsman'],autopct='%0.1f%%')
plt.show()
```



In [ ]:
```
# percentage and colors
plt.pie(df['batsman_runs'],labels=df['batsman'],autopct='%0.1f%%',colors=['blue','g
plt.show()
```



In [ ]:
```
# explode shadow
plt.pie(df['batsman_runs'],labels=df['batsman'],autopct='%0.1f%%',explode=[0.3,0,0,
plt.show()
```



Matplotlib Doc Website :

https://matplotlib.org/stable/

Advanced Matplotlib(part-1)

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Colored Scatterplots

In [ ]:
```
iris = pd.read_csv('Data\Day49\iris.csv')
iris.sample(5)
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
146	147	6.3	2.5	5.0	1.9	Iris-virginica
123	124	6.3	2.7	4.9	1.8	Iris-virginica
78	79	6.0	2.9	4.5	1.5	Iris-versicolor
127	128	6.1	3.0	4.9	1.8	Iris-virginica
143	144	6.8	3.2	5.9	2.3	Iris-virginica

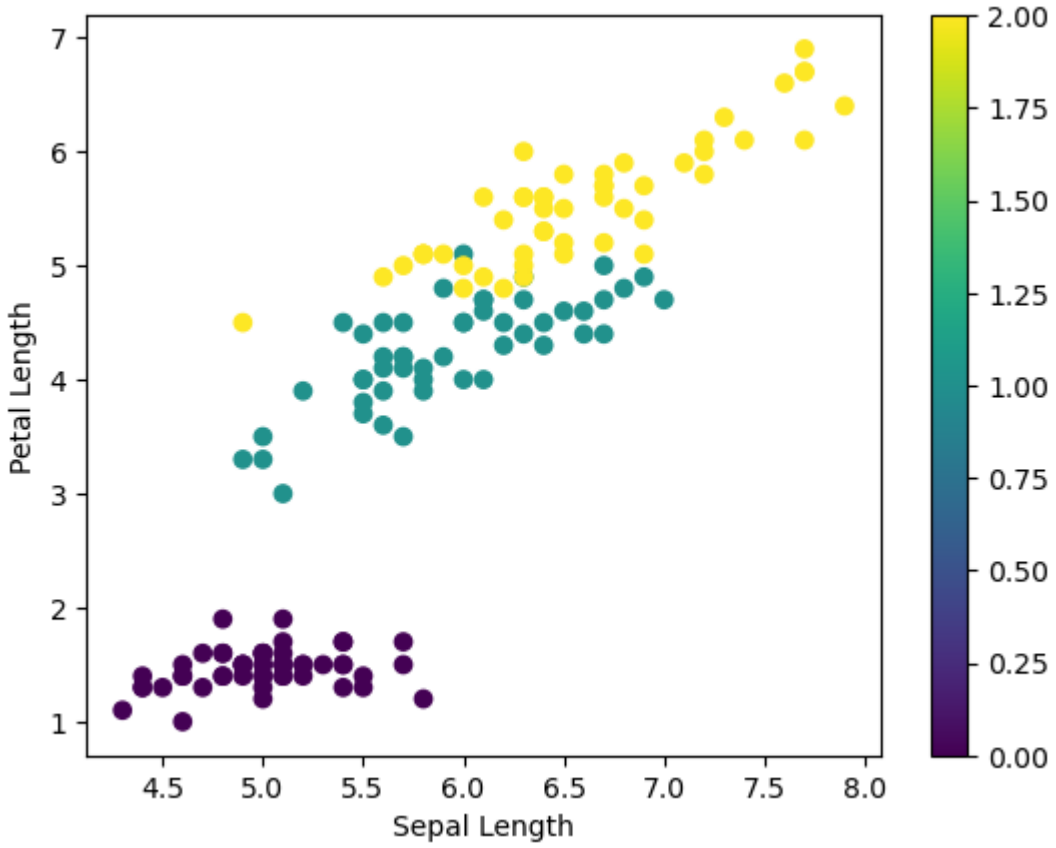
In [ ]:
```
iris['Species'] = iris['Species'].replace({'Iris-setosa':0,'Iris-versicolor':1,'Iri
iris.sample(5)
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
90	91	5.5	2.6	4.4	1.2	1
25	26	5.0	3.0	1.6	0.2	0
89	90	5.5	2.5	4.0	1.3	1
95	96	5.7	3.0	4.2	1.2	1
148	149	6.2	3.4	5.4	2.3	2

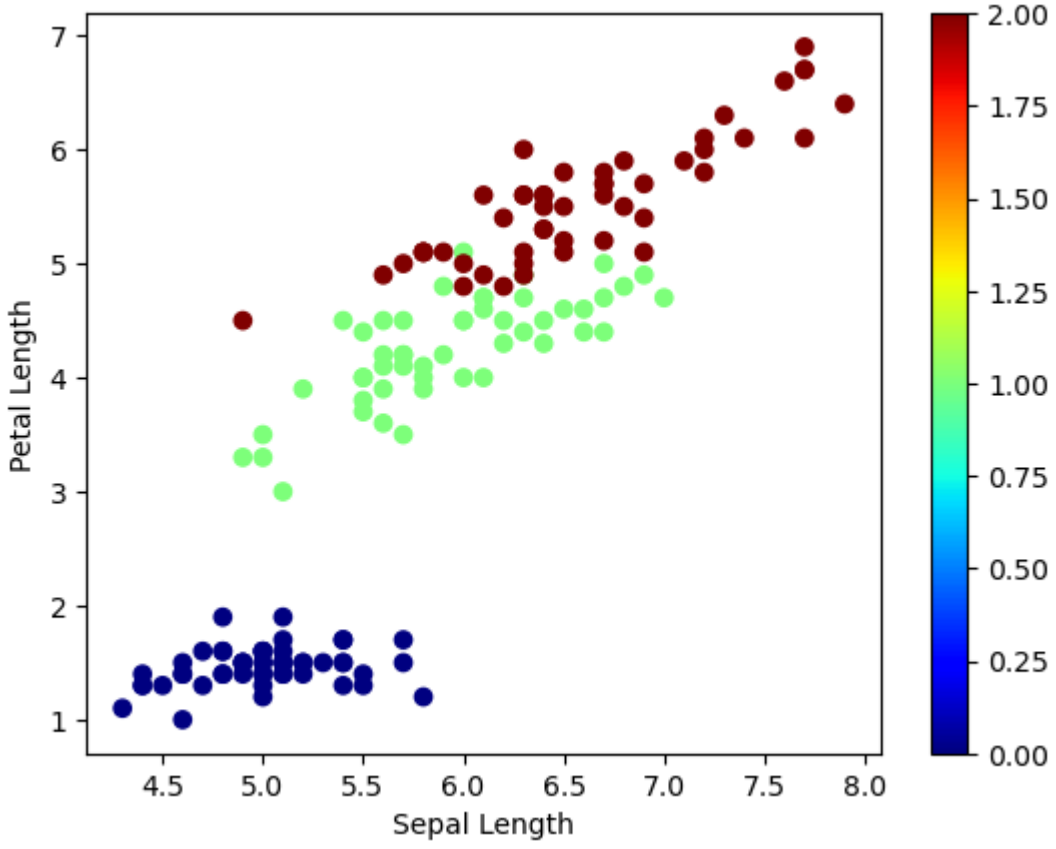
In [ ]:
```
plt.scatter(iris['SepalLengthCm'],iris['PetalLengthCm'],c=iris['Species'])
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1ec76de6880>
```



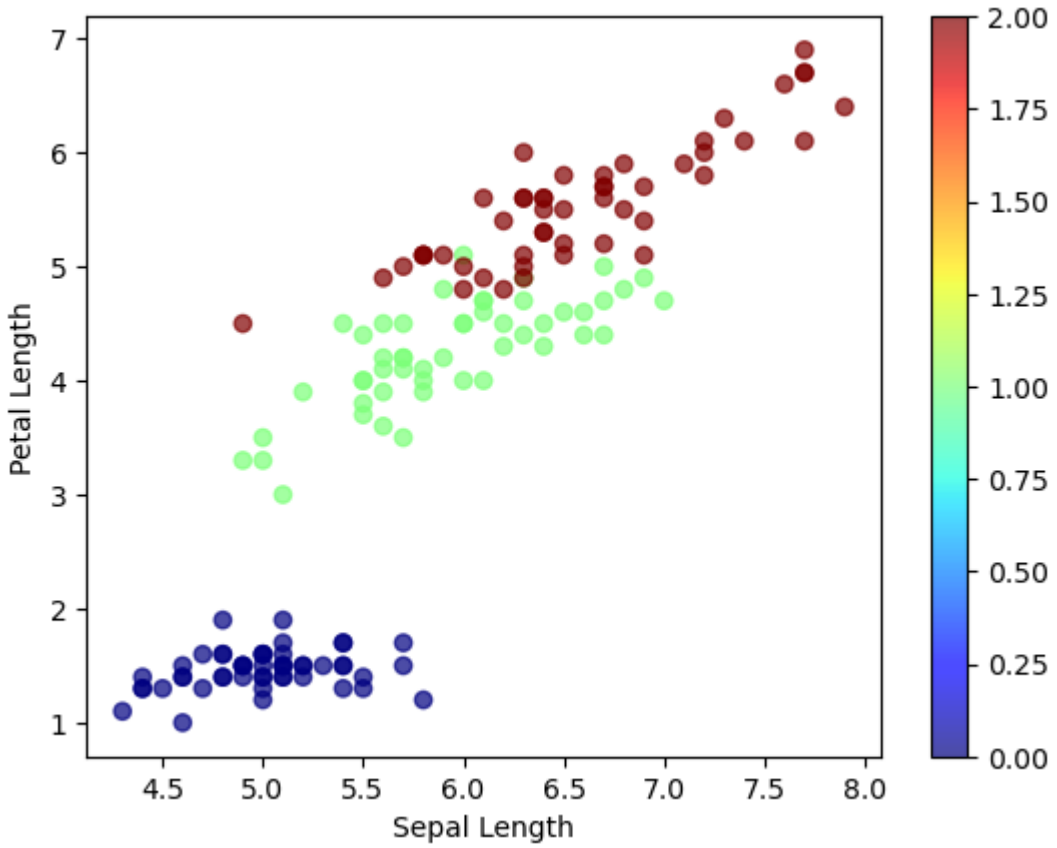
In [ ]:
```
# cmap
plt.scatter(iris['SepalLengthCm'],iris['PetalLengthCm'],c=iris['Species'],cmap='jet
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1ec75d12f10>
```



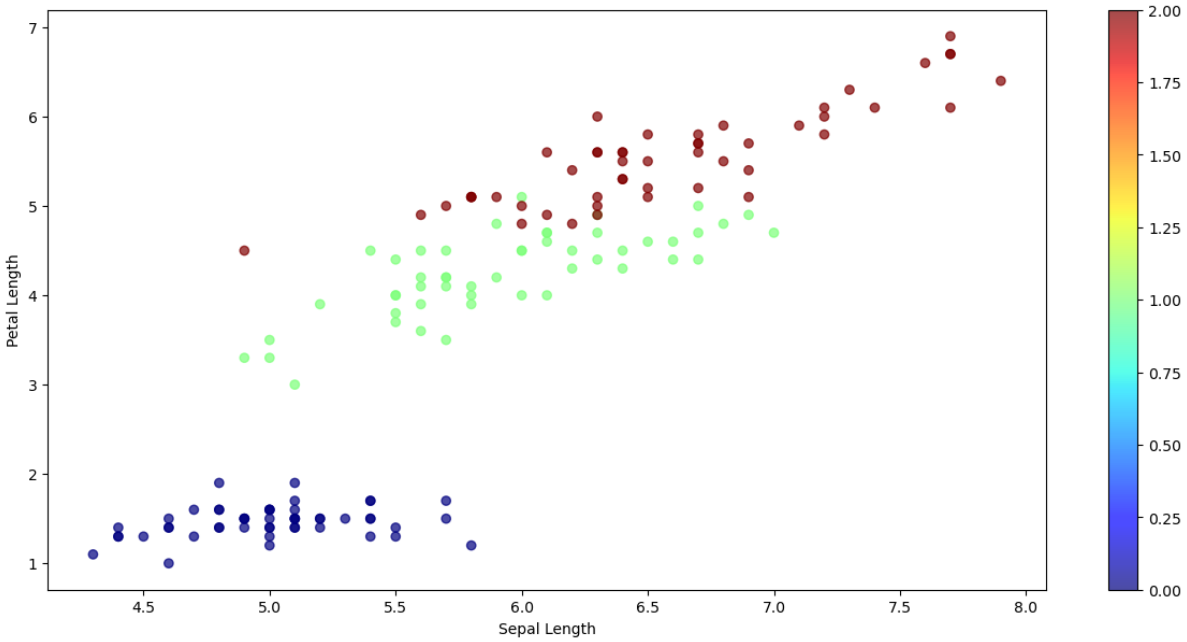
In [ ]:
```
# alpha
plt.scatter(iris['SepalLengthCm'],iris['PetalLengthCm'],c=iris['Species'],cmap='jet
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1ec76e54790>
```



In [ ]:
```
# plot size
plt.figure(figsize=(15,7))

plt.scatter(iris['SepalLengthCm'],iris['PetalLengthCm'],c=iris['Species'],cmap='jet
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1ec76f3bf40>
```



Annotations

In [ ]:
```
batters = pd.read_csv('Data\Day49\Batter.csv')
```

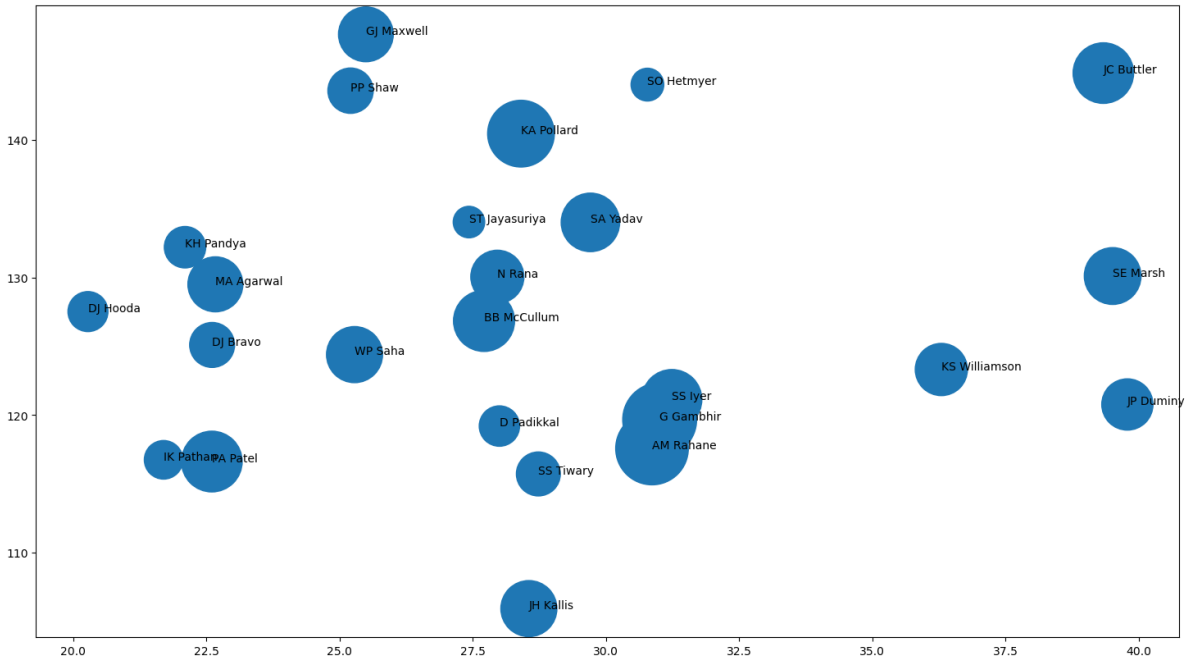
In [ ]:
```
sample_df = batters.head(100).sample(25,random_state=5)
```

In [ ]:
```
sample_df
```

Out[ ]:

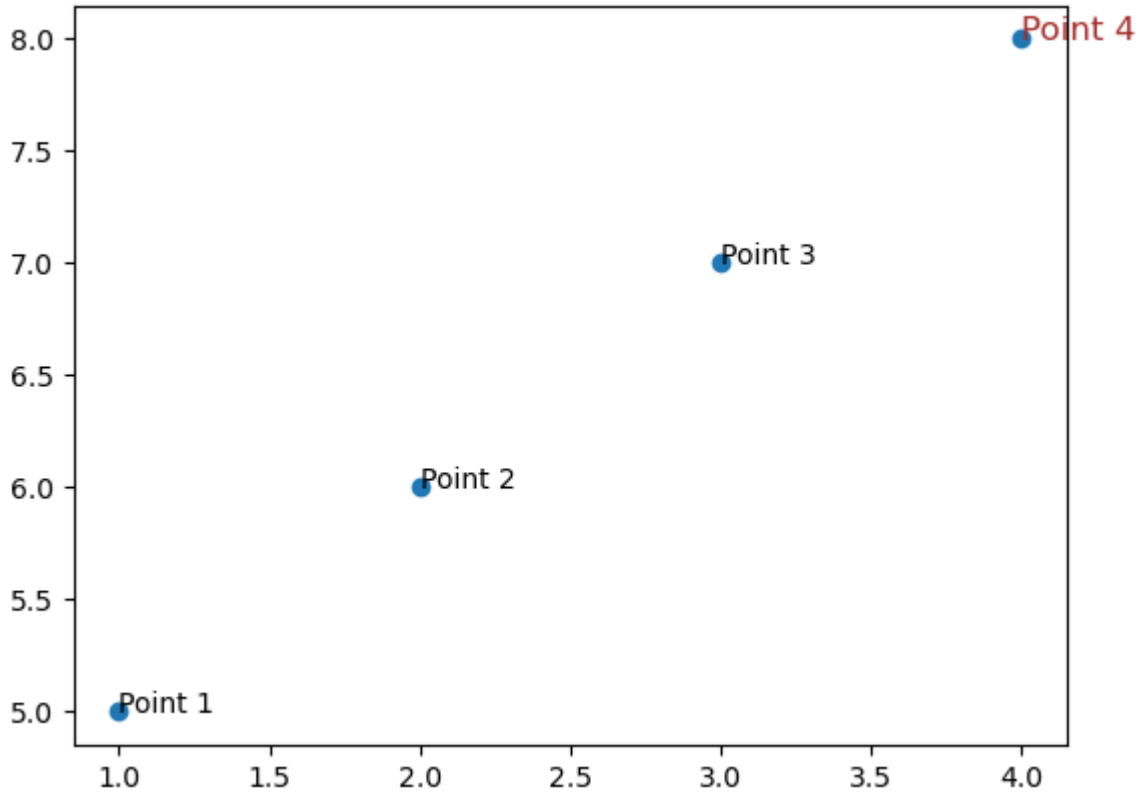
	batter	runs	avg	strike_rate
66	KH Pandya	1326	22.100000	132.203390
32	SE Marsh	2489	39.507937	130.109775
46	JP Duminy	2029	39.784314	120.773810
28	SA Yadav	2644	29.707865	134.009123
74	IK Pathan	1150	21.698113	116.751269
23	JC Buttler	2832	39.333333	144.859335
10	G Gambhir	4217	31.007353	119.665153
20	BB McCullum	2882	27.711538	126.848592
17	KA Pollard	3437	28.404959	140.457703
35	WP Saha	2427	25.281250	124.397745
97	ST Jayasuriya	768	27.428571	134.031414
37	MA Agarwal	2335	22.669903	129.506378
70	DJ Hooda	1237	20.278689	127.525773
40	N Rana	2181	27.961538	130.053667
60	SS Tiwary	1494	28.730769	115.724245
34	JH Kallis	2427	28.552941	105.936272
42	KS Williamson	2105	36.293103	123.315759
57	DJ Bravo	1560	22.608696	125.100241
12	AM Rahane	4074	30.863636	117.575758
69	D Padikkal	1260	28.000000	119.205298
94	SO Hetmyer	831	30.777778	144.020797
56	PP Shaw	1588	25.206349	143.580470
22	PA Patel	2848	22.603175	116.625717
39	GJ Maxwell	2320	25.494505	147.676639
24	SS Iyer	2780	31.235955	121.132898

In [ ]:
```
plt.figure(figsize=(18,10))
plt.scatter(sample_df['avg'],sample_df['strike_rate'],s=sample_df['runs'])

for i in range(sample_df.shape[0]):
  plt.text(sample_df['avg'].values[i],sample_df['strike_rate'].values[i],sample_df[
```



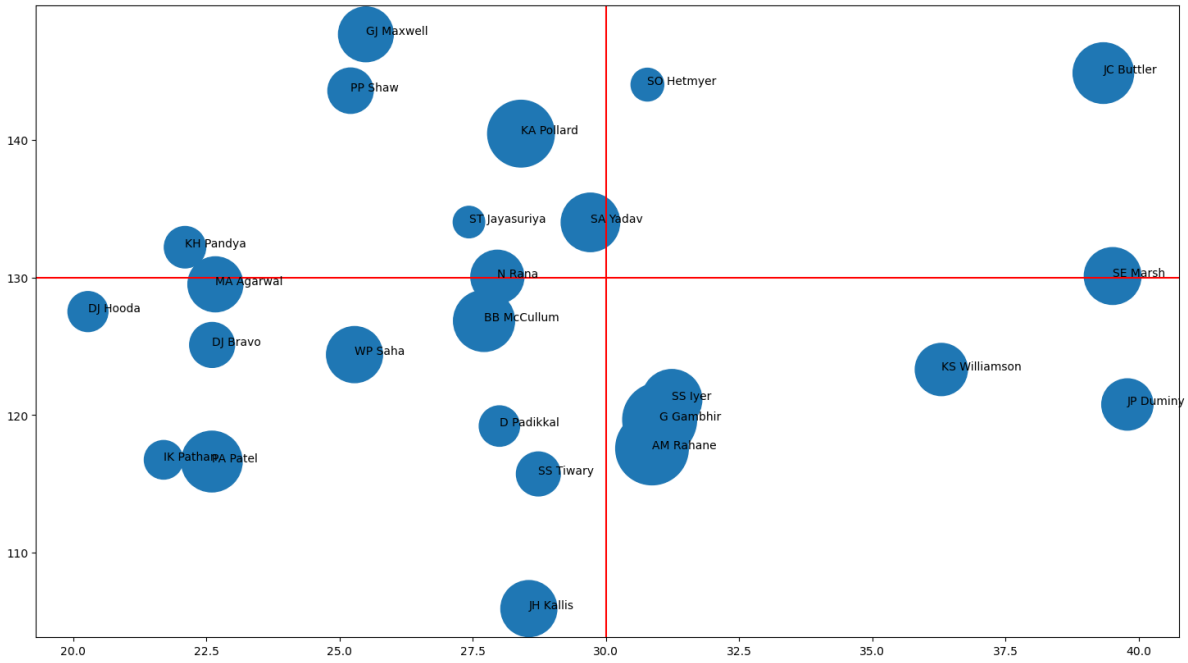
In [ ]:
```
x = [1,2,3,4]
y = [5,6,7,8]

plt.scatter(x,y)
plt.text(1,5,'Point 1')
plt.text(2,6,'Point 2')
plt.text(3,7,'Point 3')
plt.text(4,8,'Point 4',fontdict={'size':12,'color':'brown'})
```

Out[ ]:
```
Text(4, 8, 'Point 4')
```



In [ ]:
```
### Horizontal and Vertical lines

plt.figure(figsize=(18,10))
plt.scatter(sample_df['avg'],sample_df['strike_rate'],s=sample_df['runs'])
```

```
plt.axhline(130,color='red')

plt.axvline(30,color='red')

for i in range(sample_df.shape[0]):
  plt.text(sample_df['avg'].values[i],sample_df['strike_rate'].values[i],sample_df[
```



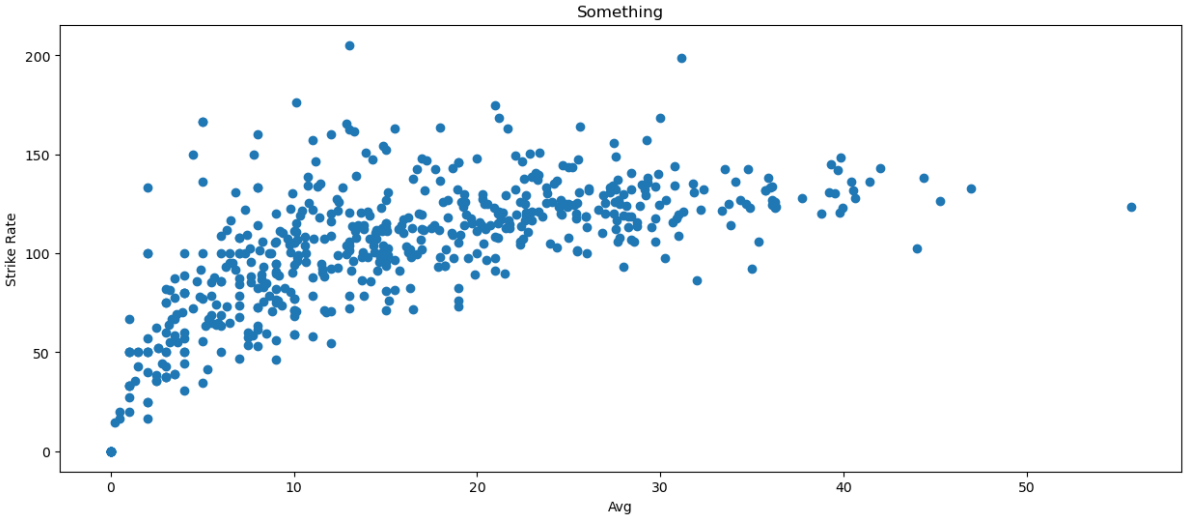
Subplots

In [ ]:
```
# A diff way to plot graphs
batters.head()
```

Out[ ]:

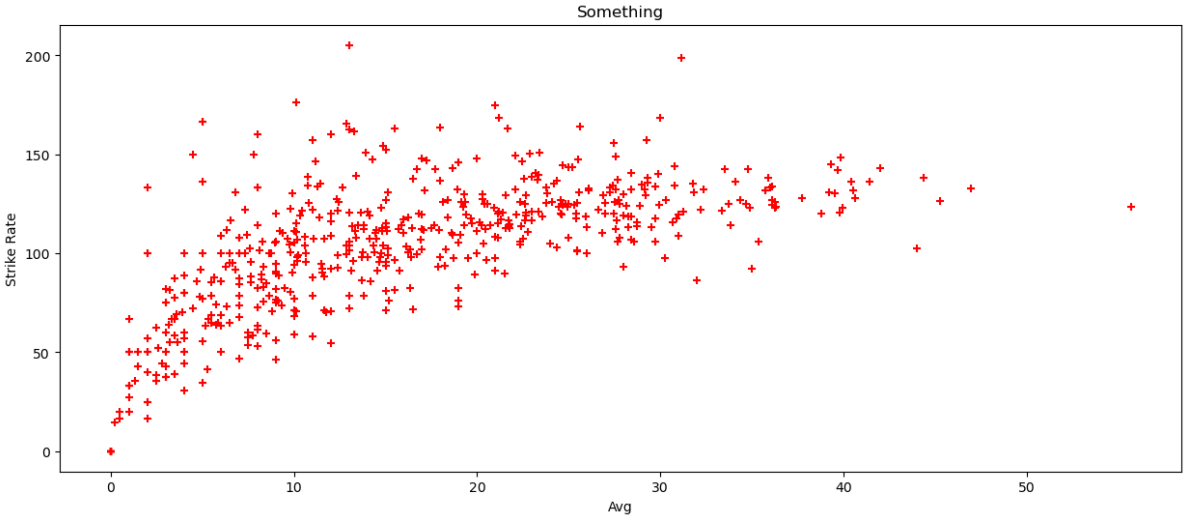
	batter	runs	avg	strike_rate
0	V Kohli	6634	36.251366	125.977972
1	S Dhawan	6244	34.882682	122.840842
2	DA Warner	5883	41.429577	136.401577
3	RG Sharma	5881	30.314433	126.964594
4	SK Raina	5536	32.374269	132.535312

In [ ]:
```
plt.figure(figsize=(15,6))
plt.scatter(batters['avg'],batters['strike_rate'])
plt.title('Something')
plt.xlabel('Avg')
plt.ylabel('Strike Rate')

plt.show()
```



In [ ]:
```
fig,ax = plt.subplots(figsize=(15,6))

ax.scatter(batters['avg'],batters['strike_rate'],color='red',marker='+')
ax.set_title('Something')
ax.set_xlabel('Avg')
ax.set_ylabel('Strike Rate')

fig.show()
```

```
C:\Users\disha\AppData\Local\Temp\ipykernel_4684\3179312453.py:8: UserWarning: Mat
plotlib is currently using module://matplotlib_inline.backend_inline, which is a n
on-GUI backend, so cannot show the figure.
  fig.show()
```

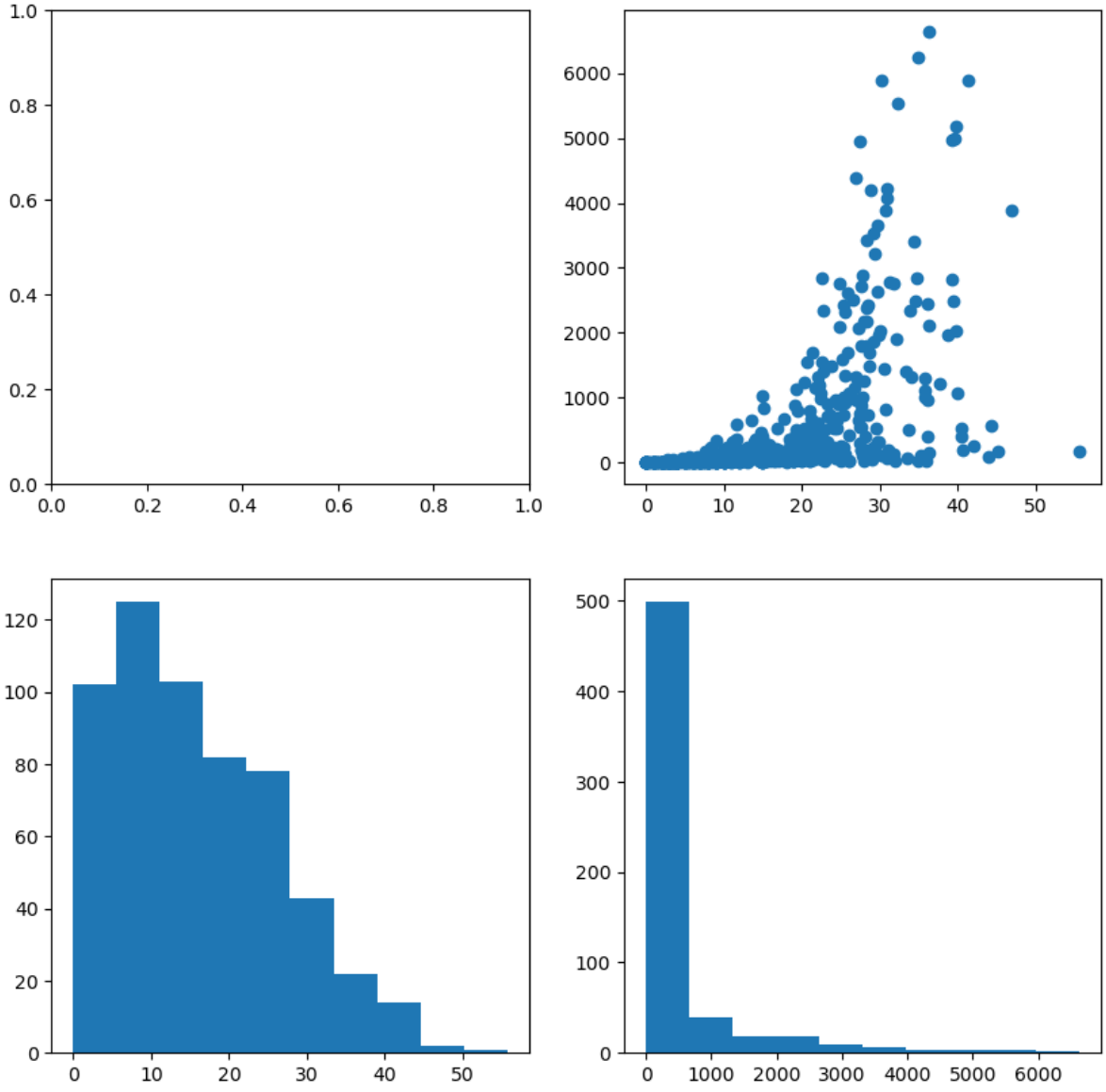


In [ ]:
```
ig, ax = plt.subplots(nrows=2,ncols=1,sharex=True,figsize=(10,6))

ax[0].scatter(batters['avg'],batters['strike_rate'],color='red')
ax[1].scatter(batters['avg'],batters['runs'])

ax[0].set_title('Avg Vs Strike Rate')
ax[0].set_ylabel('Strike Rate')


ax[1].set_title('Avg Vs Runs')
ax[1].set_ylabel('Runs')
ax[1].set_xlabel('Avg')
```

Out[ ]:
```
Text(0.5, 0, 'Avg')
```

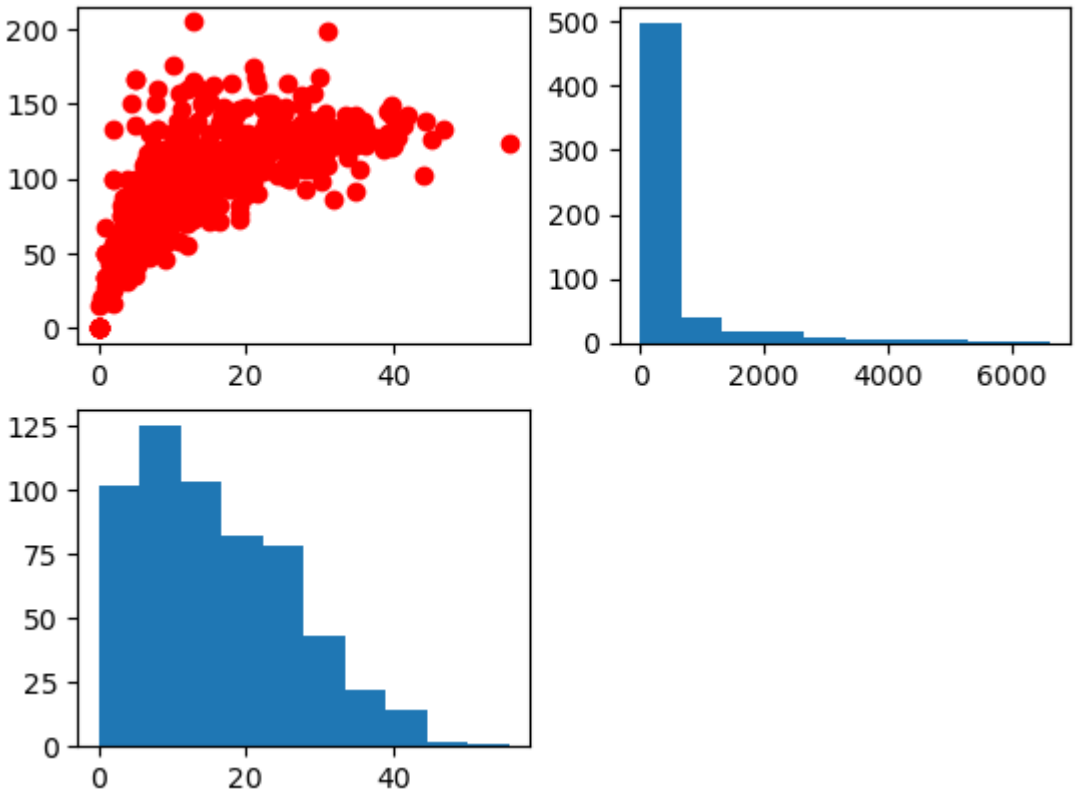
In [ ]:
```
fig, ax = plt.subplots(nrows=2,ncols=2,figsize=(10,10))

ax[0,0]
ax[0,1].scatter(batters['avg'],batters['runs'])
ax[1,0].hist(batters['avg'])
ax[1,1].hist(batters['runs'])
```

Out[ ]:
```
(array([499.,  40.,  19.,  19.,   9.,   6.,   4.,   4.,   3.,   2.]),
 array([   0. ,  663.4, 1326.8, 1990.2, 2653.6, 3317. , 3980.4, 4643.8,
        5307.2, 5970.6, 6634. ]),
 <BarContainer object of 10 artists>)
```



In [ ]:
```
fig = plt.figure()

ax1 = fig.add_subplot(2,2,1)
ax1.scatter(batters['avg'],batters['strike_rate'],color='red')

ax2 = fig.add_subplot(2,2,2)
ax2.hist(batters['runs'])

ax3 = fig.add_subplot(2,2,3)
ax3.hist(batters['avg'])
```

Out[ ]:
```
(array([102., 125., 103.,  82.,  78.,  43.,  22.,  14.,   2.,   1.]),
 array([ 0.        ,  5.56666667, 11.13333333, 16.7       , 22.26666667,
        27.83333333, 33.4       , 38.96666667, 44.53333333, 50.1       ,
        55.66666667]),
 <BarContainer object of 10 artists>)
```


200
150
100
50
0
0
20
40
500
400
300
200
100
0
0
2000
4000
6000
125
100
75
50
25
0
0
20
40

Advanced Matplotlib(part-2)

3D scatter Plot

- A 3D scatter plot is used to represent data points in a three-dimensional space.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

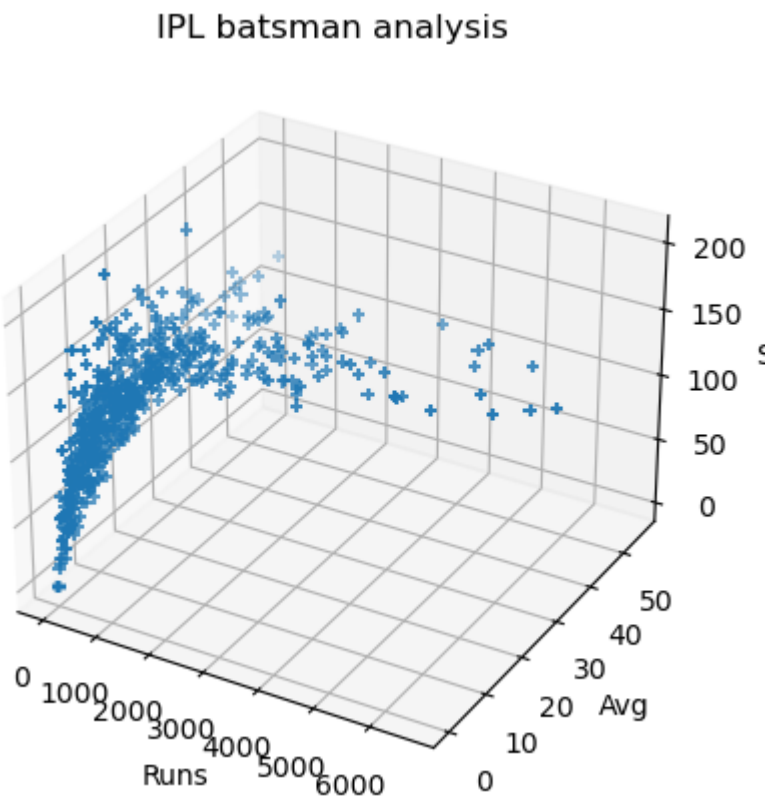
In [ ]:
```
batters = pd.read_csv('Data\Day49\Batter.csv')
batters.head()
```

Out[ ]:

	batter	runs	avg	strike_rate
0	V Kohli	6634	36.251366	125.977972
1	S Dhawan	6244	34.882682	122.840842
2	DA Warner	5883	41.429577	136.401577
3	RG Sharma	5881	30.314433	126.964594
4	SK Raina	5536	32.374269	132.535312

In [ ]:
```
fig = plt.figure()

ax = plt.subplot(projection='3d')

ax.scatter3D(batters['runs'],batters['avg'],batters['strike_rate'],marker='+')
ax.set_title('IPL batsman analysis')

ax.set_xlabel('Runs')
ax.set_ylabel('Avg')
ax.set_zlabel('SR')
```

Out[ ]:
```
Text(0.5, 0, 'SR')
```



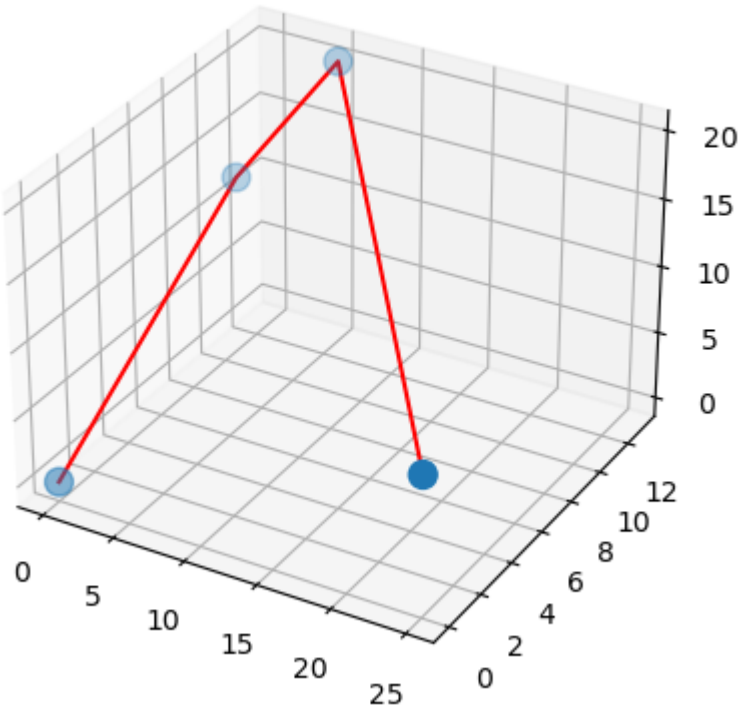
- In the example, you created a 3D scatter plot to analyze IPL batsmen based on runs, average (avg), and strike rate (SR).
- The ax.scatter3D function was used to create the plot, where the three variables were mapped to the x, y, and z axes.

3D Line Plot

- A 3D line plot represents data as a line in three-dimensional space.

In [ ]:
```
x = [0,1,5,25]
y = [0,10,13,0]
z = [0,13,20,9]

fig = plt.figure()

ax = plt.subplot(projection='3d')

ax.scatter3D(x,y,z,s=[100,100,100,100])
ax.plot3D(x,y,z,color='red')
```

Out[ ]:
```
[<mpl_toolkits.mplot3d.art3d.Line3D at 0x23ec4988340>]
```



- In the given example, you created a 3D line plot with three sets of data points represented by lists x, y, and z.
- The ax.plot3D function was used to create the line plot.

3D Surface Plots

- 3D surface plots are used to visualize functions of two variables as surfaces in three-dimensional space.

In [ ]:
```
x = np.linspace(-10,10,100)
y = np.linspace(-10,10,100)
```

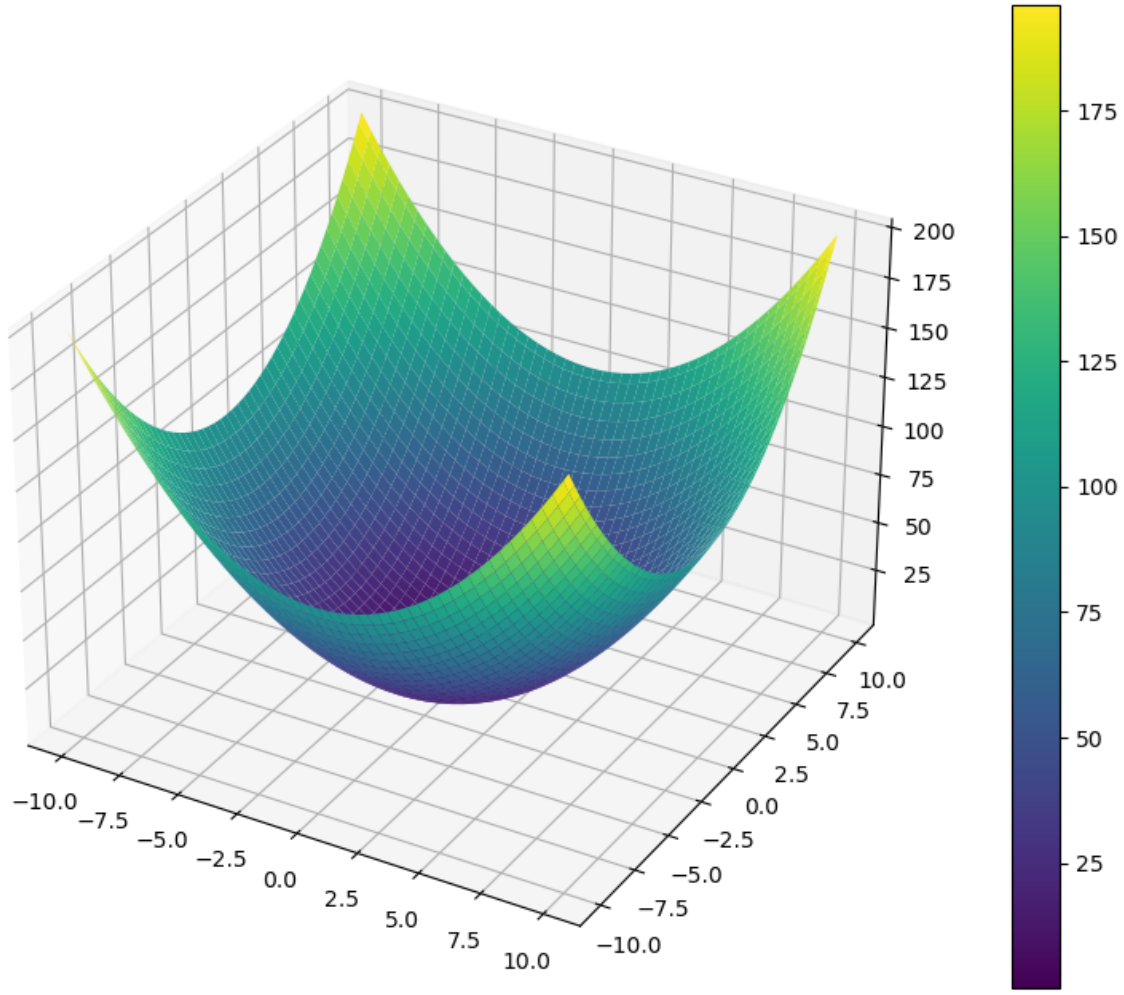
In [ ]:
```
xx, yy = np.meshgrid(x,y)
```

In [ ]:
```
z = xx**2 + yy**2
z.shape
```

Out[ ]:
```
(100, 100)
```

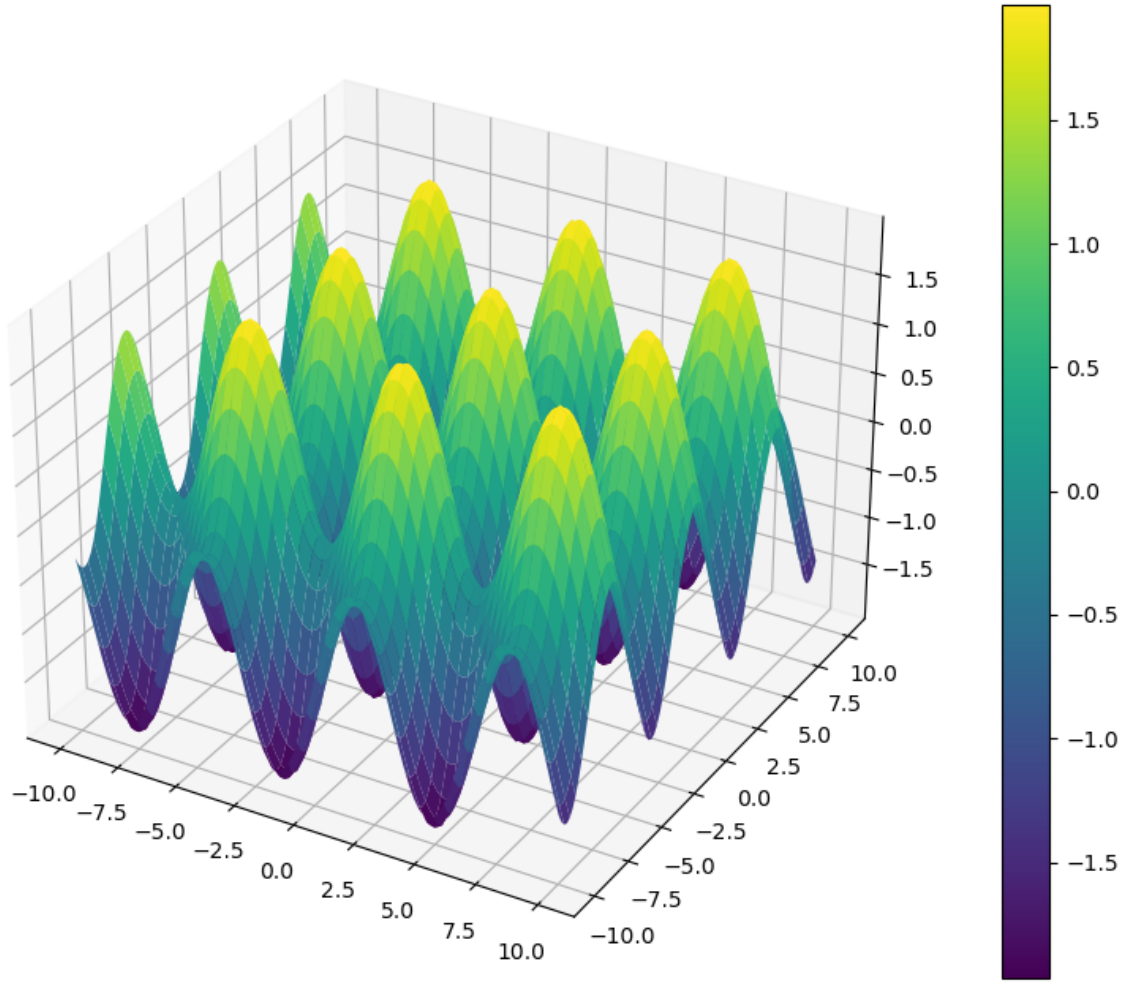
In [ ]:
```
fig = plt.figure(figsize=(12,8))

ax = plt.subplot(projection='3d')

p = ax.plot_surface(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec66ca9a0>
```



In [ ]:
```
z = np.sin(xx) + np.cos(yy)

fig = plt.figure(figsize=(12,8))

ax = plt.subplot(projection='3d')

p = ax.plot_surface(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec33aa520>
```



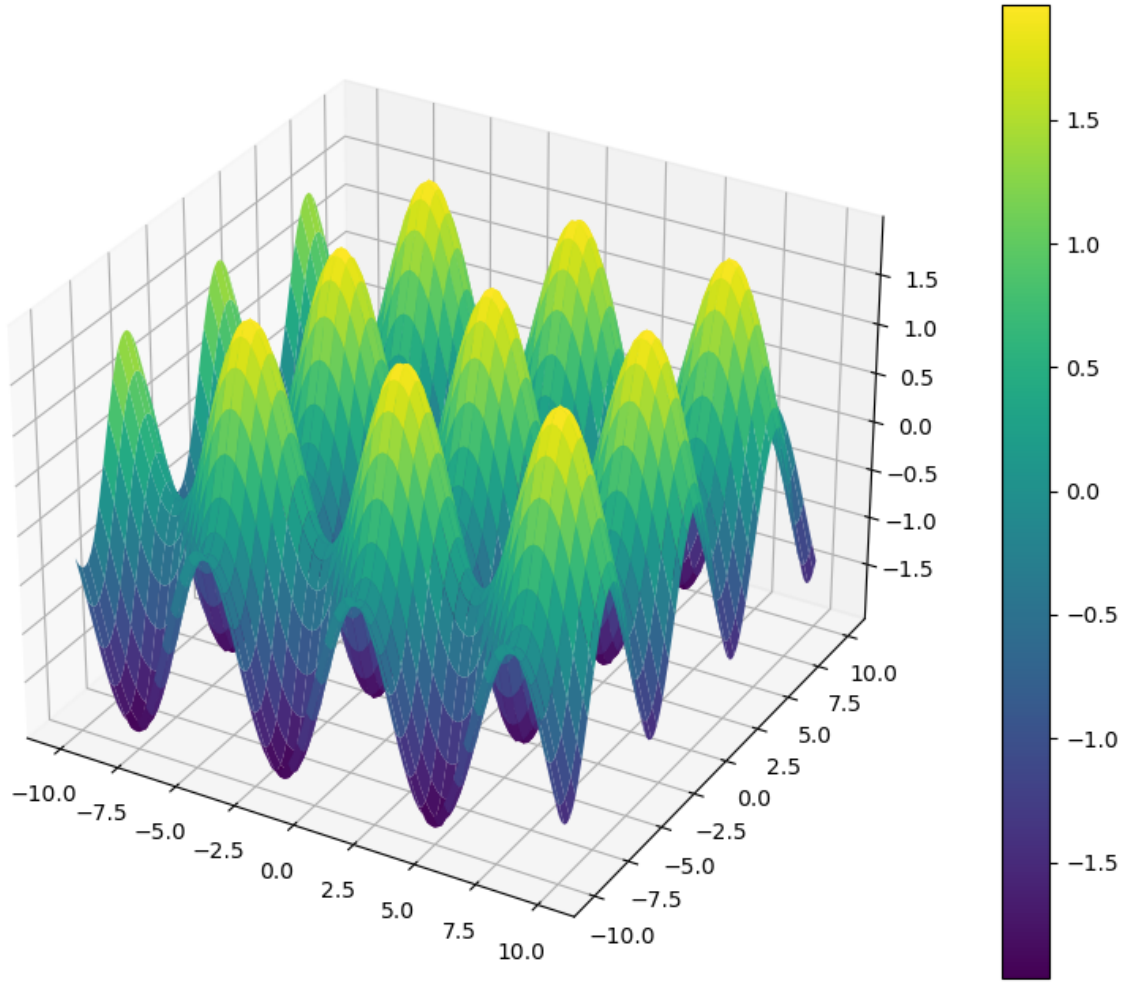
surface plot using the ax.plot_surface function. In First example, you plotted a parabolic surface, and in Seound, you plotted a surface with sine and cosine functions.

Contour Plots

- Contour plots are used to visualize 3D data in 2D, representing data as contours on a 2D plane.

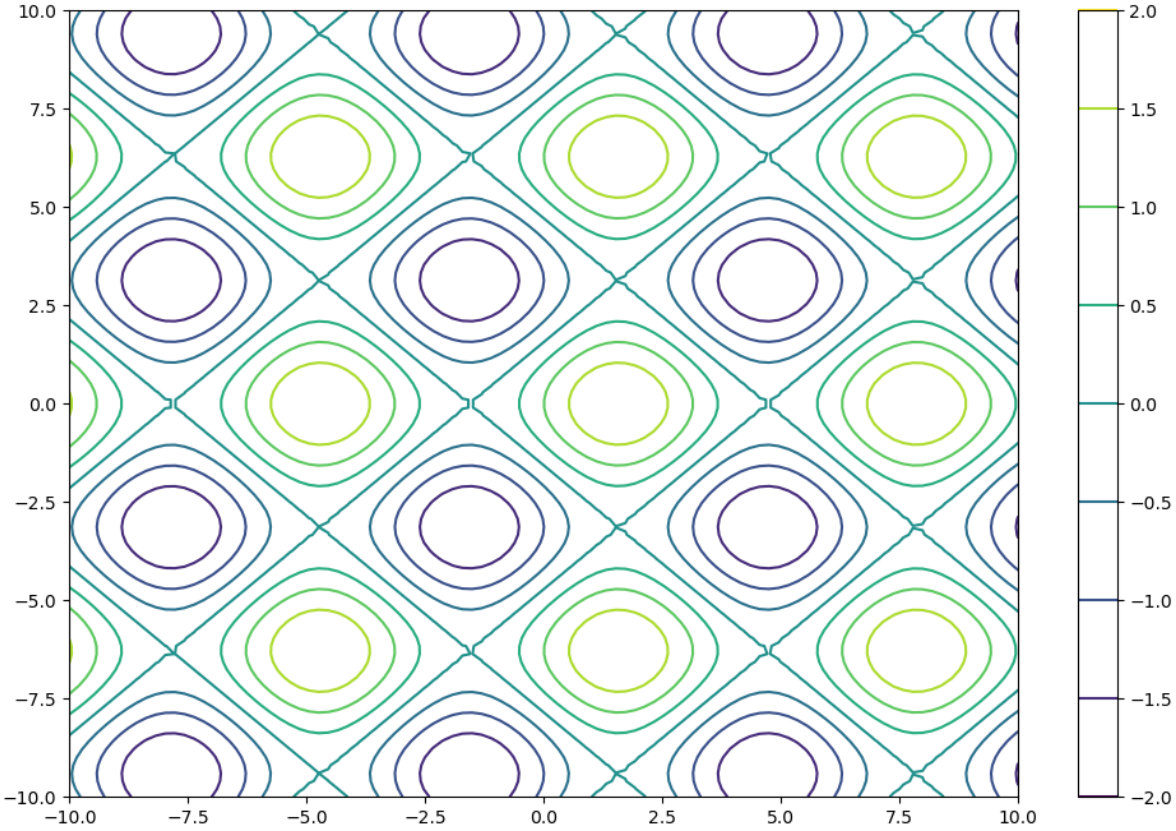
In [ ]:
```
fig = plt.figure(figsize=(12,8))

ax = plt.subplot(projection='3d')

p = ax.plot_surface(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec616e0d0>
```



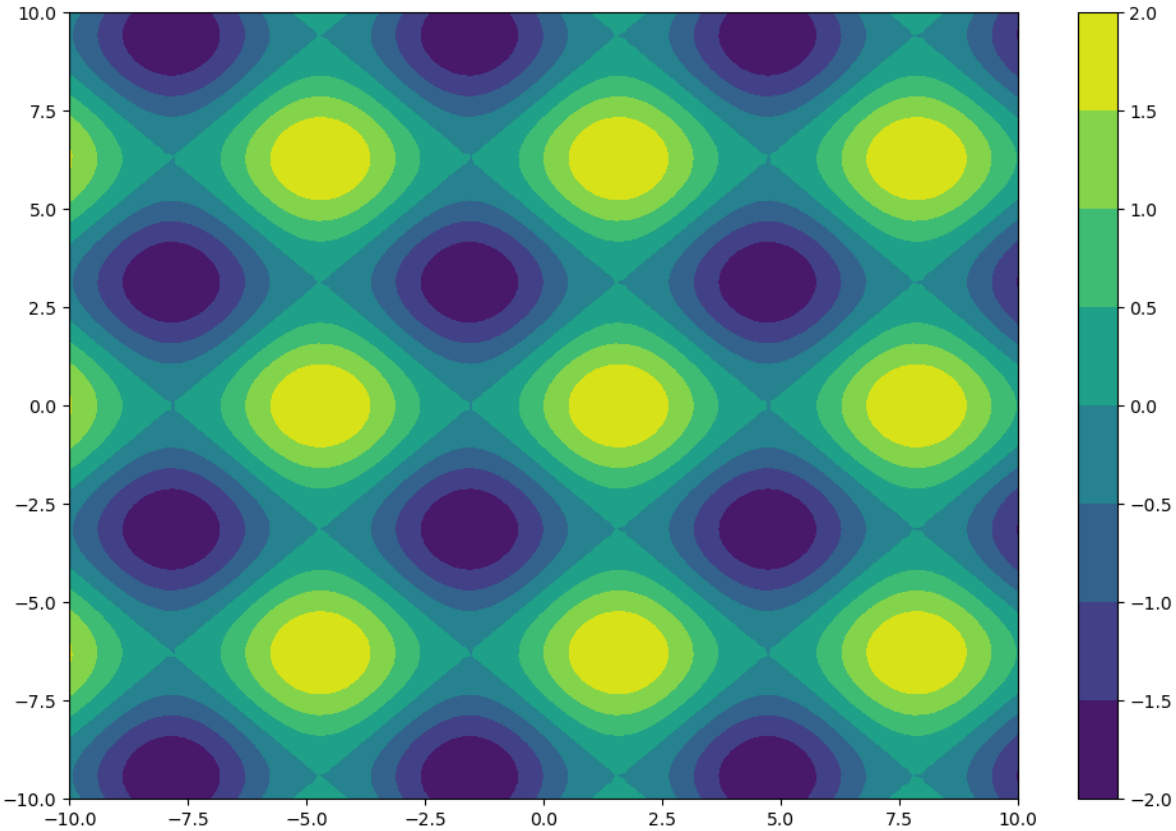
In [ ]:
```
fig = plt.figure(figsize=(12,8))

ax = plt.subplot()

p = ax.contour(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec56e4be0>
```



In [ ]:
```
z = np.sin(xx) + np.cos(yy)

fig = plt.figure(figsize=(12,8))

ax = plt.subplot()

p = ax.contourf(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

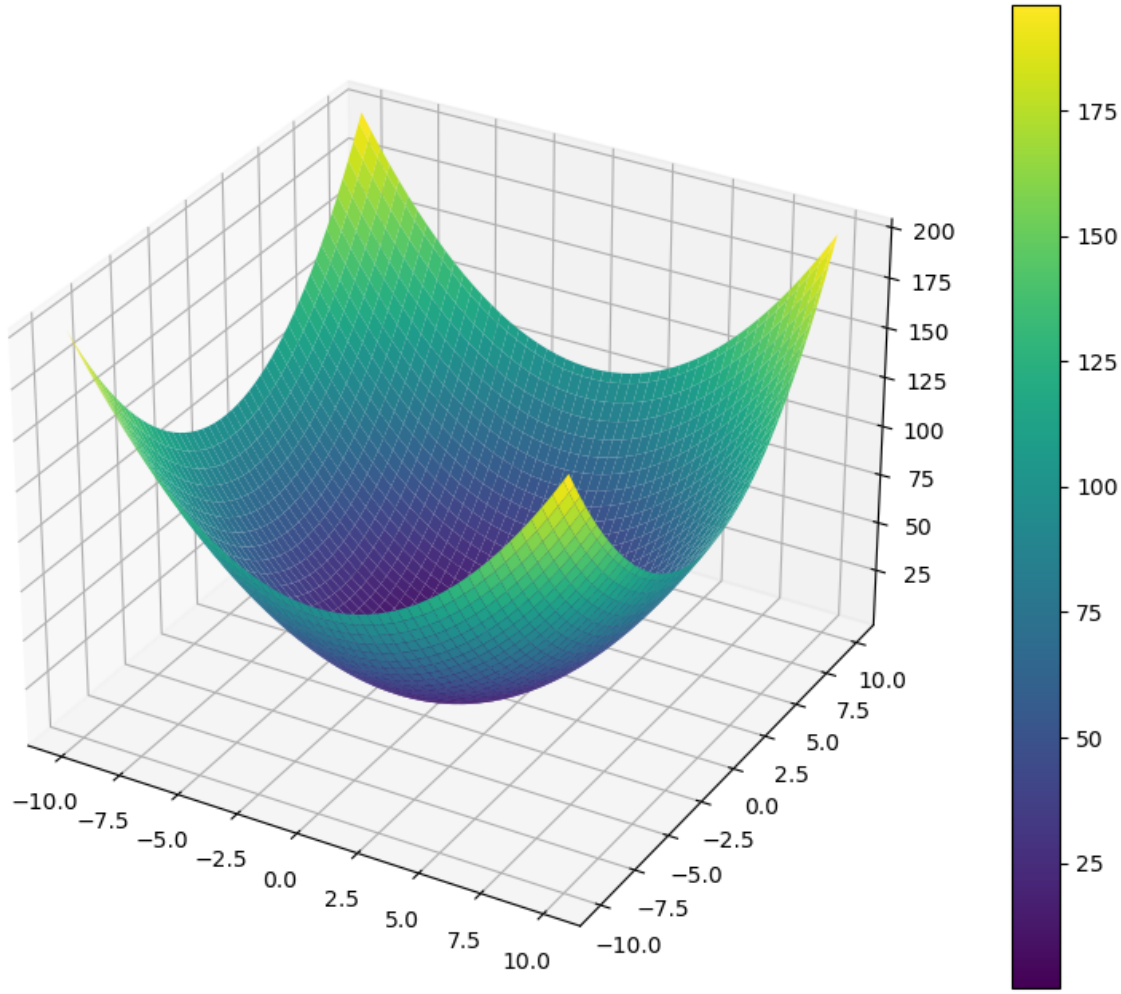
Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec8865f40>
```



You created both filled contour plots (ax.contourf) and contour line plots (ax.contour) in 2D space. These plots are useful for representing functions over a grid.

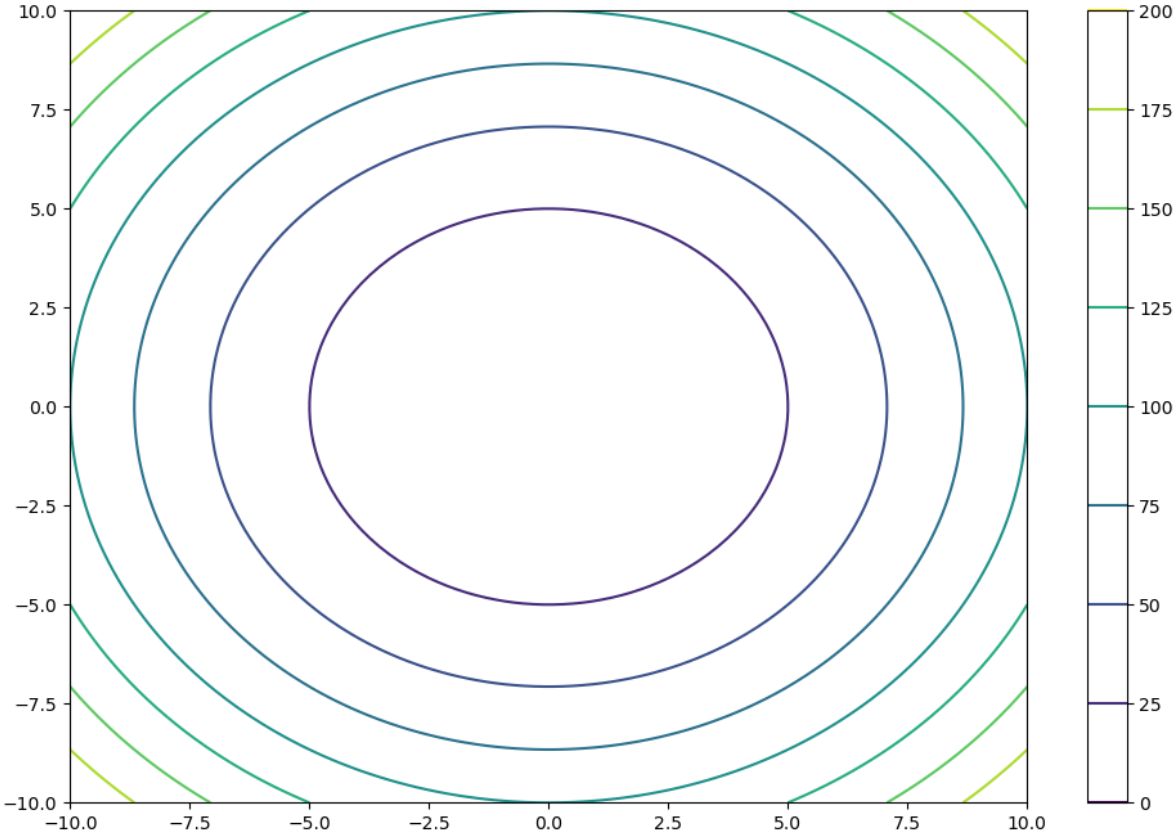
In [ ]:
```
fig = plt.figure(figsize=(12,8))

ax = plt.subplot(projection='3d')

p = ax.plot_surface(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec7b7ca00>
```



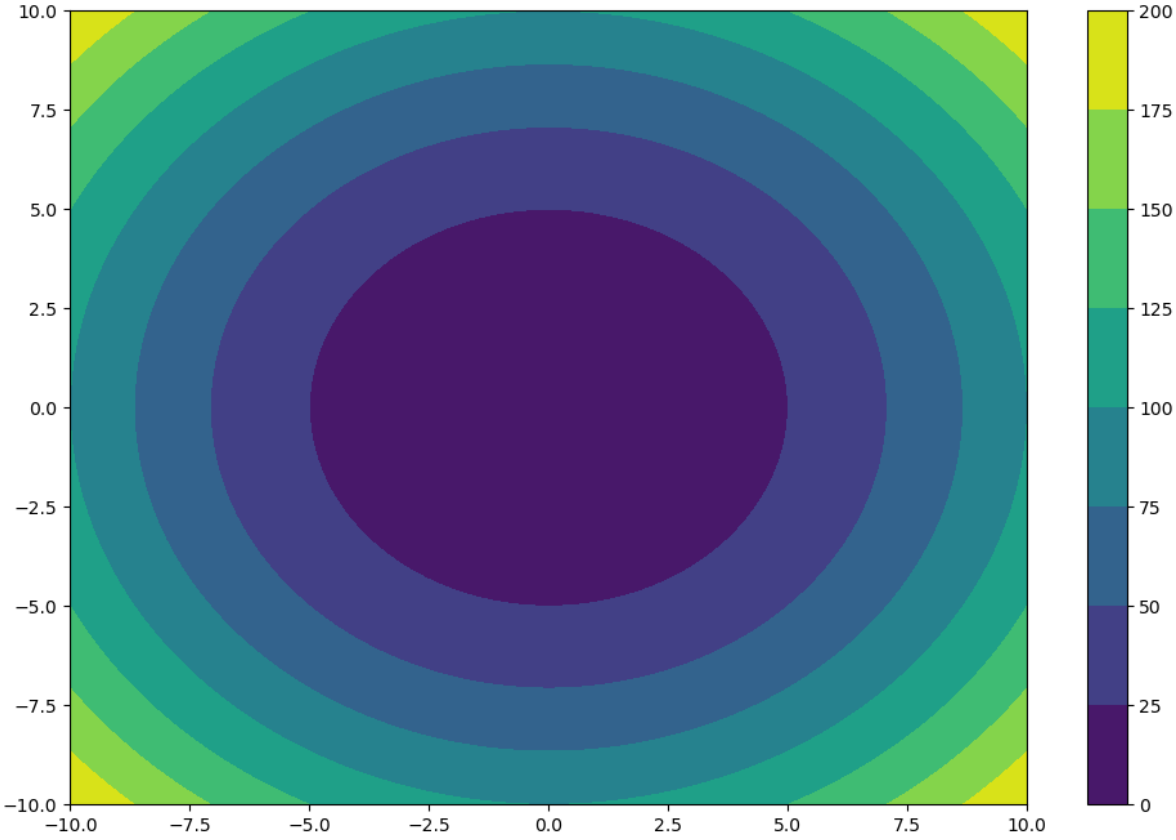
In [ ]:
```
fig = plt.figure(figsize=(12,8))

ax = plt.subplot()

p = ax.contour(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec7c698e0>
```



In [ ]:
```
fig = plt.figure(figsize=(12,8))

ax = plt.subplot()

p = ax.contourf(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec7ed9700>
```



Heatmap

A heatmap is a graphical representation of data in a 2D grid, where individual values are represented as colors.

In [ ]:
```
delivery = pd.read_csv('Data\Day50\IPL_Ball_by_Ball_2008_2022.csv')
delivery.head()
```

Out[ ]:

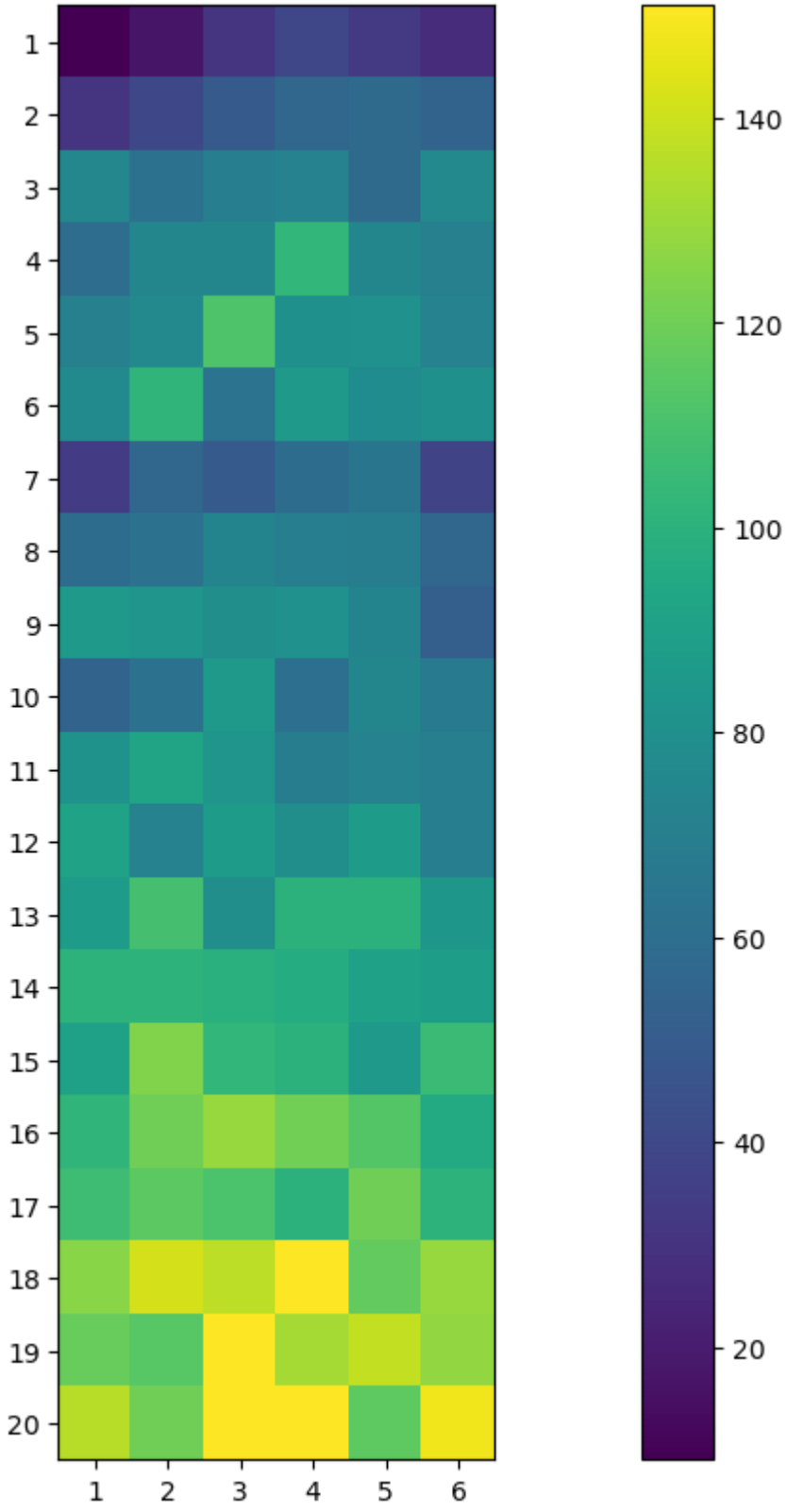
	ID	innings	overs	ballnumber	batter	bowler	non-striker	extra_type	batsman_run	ex
0	1312200	1	0	1	YBK Jaiswal	Mohammed Shami	JC Buttler	NaN	0	
1	1312200	1	0	2	YBK Jaiswal	Mohammed Shami	JC Buttler	legbyes	0	
2	1312200	1	0	3	JC Buttler	Mohammed Shami	YBK Jaiswal	NaN	1	
3	1312200	1	0	4	YBK Jaiswal	Mohammed Shami	JC Buttler	NaN	0	
4	1312200	1	0	5	YBK Jaiswal	Mohammed Shami	JC Buttler	NaN	0	

In [ ]:
```
temp_df = delivery[(delivery['ballnumber'].isin([1,2,3,4,5,6])) & (delivery['batsma
```

In [ ]:
```
grid = temp_df.pivot_table(index='overs',columns='ballnumber',values='batsman_run',
```

In [ ]:
```
plt.figure(figsize=(20,10))
plt.imshow(grid)
plt.yticks(delivery['overs'].unique(), list(range(1,21)))
plt.xticks(np.arange(0,6), list(range(1,7)))
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec9384820>
```



- In the given example, we used the `imshow` function to create a heatmap of IPL deliveries.
- The grid represented ball-by-ball data with the number of sixes (batsman_run=6) in each over and ball number.
- Heatmaps are effective for visualizing patterns and trends in large datasets.

These techniques provide powerful tools for visualizing complex data in three dimensions and for representing large datasets effectively. Each type of plot is suitable for different types of data and can help in gaining insights from the data.