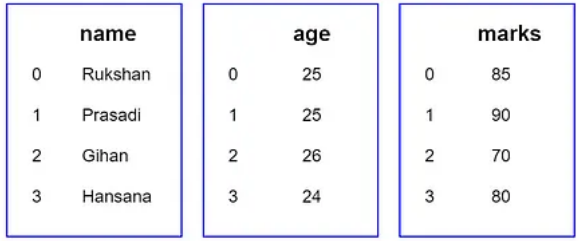
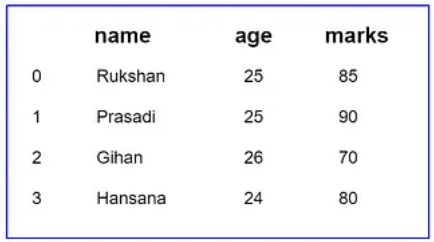
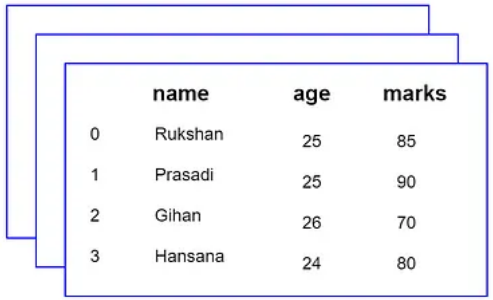
Sr. No.	Question	Answers
1	Define the Pandas/Python pandas and its renowned features	Pandas is an open-source Python library used for data manipulation and analysis. It provides data structures for efficiently storing and manipulating data, as well as tools for cleaning, merging, filtering, and transforming data. Some of the renowned features of Pandas are: Data Structures: Pandas provides two primary data structures: Series (1-dimensional) and DataFrame (2-dimensional). These data structures are designed to handle heterogeneous data types and missing values. Data Manipulation: Pandas allows you to perform operations on the data such as filtering, selecting, transforming, merging, and cleaning data. Handling Missing Data: Pandas provides tools to handle missing data in the form of NaN (Not a Number) or None values. Time Series Data: Pandas has built-in support for working with time series data, including tools for resampling, time zone handling, and date range generation. Data Visualization: Pandas integrates with other data visualization libraries such as Matplotlib and Seaborn to provide easy-to-use data visualization tools. Input/Output: Pandas can read and write data in a variety of formats, including CSV, Excel, SQL databases, and more. Efficient Data Processing: Pandas is built on top of NumPy and Cython, which allows for fast and efficient data processing. Flexible Data Joining: Pandas provides flexible tools for joining data from multiple sources, including different join types and the ability to merge on multiple columns.
2	List the different types of Data Structures in Pandas.	A data structure is a collection of data values and defines the relationship between the data, and the operations that can be performed on the data. There are three main data structures in pandas: Series — 1D

DataFrame — 2D
Panel — 3D

Data Structure	Dimensionality	Format	View
Series	1D	Column	
DataFrame	2D	Single Sheet	
Panel	3D	Multiple Sheets	

The most widely used pandas data structures are the Series and the DataFrame. Simply, a Series is similar to a single column of data while a DataFrame is similar to a sheet with rows and columns. Likewise, a Panel can have many DataFrames.

Dataframe.dtypes: (also called object data types)
Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels. It can be thought of as a dict-like container for Series objects. This is the primary data structure of the Pandas.

		Pandas dataframe.dtypes attribute return the dtypes in the DataFrame. It returns a Series with the data type of each column. **Series in Pandas**: What is a series? Technically, pandas Series is a one-dimensional labeled array capable of holding any data type. So, in terms of Pandas DataStructure, A Series represents a single column in memory, which is either independent or belongs to a Pandas DataFrame. **Panel:** A panel is a 3D container of data. The term Panel data is derived from econometrics and is partially responsible for the name pandas − pan(el)-da(ta)-s. items − axis 0, each item corresponds to a DataFrame contained inside. major_axis − axis 1, it is the index (rows) of each of the DataFrames.
3	Describe Series in Pandas.	The Pandas Series can be defined as a one-dimensional array that is capable of storing various data types. We can easily convert the list, tuple, and dictionary into series using "series' method. The row labels of series are called the index. A Series cannot contain multiple columns. It has the following parameter: • data: It can be any list, dictionary, or scalar value. • index: The value of the index should be unique and hashable. It must be of the same length as data. If we do not pass any index, default np.arrange(n) will be used. • dtype: It refers to the data type of series. • copy: It is used for copying the data. import pandas as pd import numpy as np data = np.array(['h','e','l','l','o']) ser = pd.Series(data) print(ser)
4	Describe DataFrame in Pandas.	Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels. It can be thought of as a dict-like container for Series objects. This is the primary data structure of the Pandas. Pandas dataframe.dtypes attribute return the dtypes in the DataFrame. It returns a Series with the data type of each column. Alternate constructors:

		Dataframe.from_dict Dataframe.from_record <u>Example:</u> import pandas as pd lst = ['Geeks', 'For', 'Geeks', 'is', 'portal', 'for', 'Geeks'] df = pd.DataFrame(lst) print(df)

5	Exemplify the different ways a DataFrame can be created in pandas.	Pandas is a data analysis and manipulation library for Python. It provides numerous functions and methods for efficient data analysis. The core Pandas object for storing data is called dataframe which consists of labelled rows and columns.
		
		Example 1: Creating a DataFrame from a Dictionary of Lists
		import pandas as pd
		
		# create a dictionary of lists
		data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
		'Age': [25, 32, 18, 47],
		'Country': ['USA', 'Canada', 'UK', 'Australia']}
		
		# create a Pandas DataFrame from the dictionary
		df = pd.DataFrame(data)
		
		# print the DataFrame
		print(df)
		
		Example 2: Creating a DataFrame from a List of Dictionaries
		
		import pandas as pd
		
		# create a list of dictionaries
		data = [{'Name': 'Alice', 'Age': 25, 'Country': 'USA'},

		{'Name': 'Bob', 'Age': 32, 'Country': 'Canada'},
		{'Name': 'Charlie', 'Age': 18, 'Country': 'UK'},
		{'Name': 'David', 'Age': 47, 'Country': 'Australia'}]
		
		# create a Pandas DataFrame from the list of dictionaries
		df = pd.DataFrame(data)
		
		# print the DataFrame
		print(df)
		
		Example 3: Creating a DataFrame from a CSV File
		import pandas as pd
		
		# read a CSV file into a Pandas DataFrame
		df = pd.read_csv('data.csv')
		
		# print the DataFrame
		print(df)

6	Exemplify Creating a series from dict in Pandas by writing suitable code snippets.	Pandas Series is a 1-dimensional array like object which can hold data of any type. You can create a pandas series from a dictionary by passing the dictionary to the command:
		pandas.Series()
		
		Create pandas series from dictionary using “Series” method of Pandas library.
		
		import pandas as pd
		
		# create a dictionary
		d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
		
		# create a Pandas Series from the dictionary
		s = pd.Series(d)
		
		# print the Series
		print(s)

7	Write a python program to create a copy of the series in Pandas.	import pandas as pd
		
		# create a Pandas Series
		s = pd.Series([1, 2, 3, 4])
		
		# create a copy of the Series
		s_copy = s.copy()
		
		# print the original Series
		print("Original Series:")
		print(s)
		
		# print the copied Series
		print("\nCopied Series:")
		print(s_copy)

8	Write a python program to create an empty DataFrame, add an Index, row and columns to a Pandas DataFrame. And manipulate using index.	import pandas as pd
		
		# create an empty DataFrame
		df = pd.DataFrame()
		
		# add columns to the DataFrame
		df['Name'] = ['Alice', 'Bob', 'Charlie', 'David']
		df['Age'] = [25, 30, 35, 40]
		df['Gender'] = ['Female', 'Male', 'Male', 'Male']
		df['Salary'] = [50000, 60000, 70000, 80000]
		
		# add an index to the DataFrame
		df.index = ['Row1', 'Row2', 'Row3', 'Row4']
		
		# print the original DataFrame
		print("Original DataFrame:")
		print(df)
		
		# select a row using index

		row2 = df.loc['Row2']
		print("\nRow with index 'Row2':")
		print(row2)
		
		# select a column using index
		age = df['Age']
		print("\n'Age' column:")
		print(age)
		
		# select a cell using index
		cell = df.at['Row3', 'Salary']
		print("\nValue of 'Salary' for 'Row3':")
		print(cell)
		
		# update a cell using index
		df.at['Row1', 'Salary'] = 55000
		print("\nDataFrame after updating 'Salary' value for 'Row1':")
		print(df)
		
		# delete a row using index
		df = df.drop('Row3')
		print("\nDataFrame after deleting 'Row3':")
		print(df)

9	Write a python program to Delete Indices, Rows or Columns From a Pandas Data Frame.	import pandas as pd
		
		# create a sample dataframe
		data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
		'Age': [25, 30, 35, 40],
		'Gender': ['Female', 'Male', 'Male', 'Male'],
		'Salary': [50000, 60000, 70000, 80000]}
		df = pd.DataFrame(data)
		
		# print the original dataframe
		print("Original DataFrame:")

		print(df)
		
		# deleting an index
		df = df.drop(0)
		print("\nDataFrame after deleting index 0:")
		print(df)
		
		# deleting a row
		df = df[df.Name != 'Charlie']
		print("\nDataFrame after deleting row with name 'Charlie':")
		print(df)
		
		# deleting a column
		df = df.drop('Gender', axis=1)
		print("\nDataFrame after deleting 'Gender' column:")
		print(df)

10	Create a DataFrame from Dictionary of series and perform the following operations
	A. Transpose B. print the head and tail C. Summarizing data
	Trace the output along with the code.
		import pandas as pd
		
		# create dictionary of series
		data = {'Name': pd.Series(['Alice', 'Bob', 'Charlie', 'David']),
		'Age': pd.Series([25, 30, 35, 40]),
		'Gender': pd.Series(['Female', 'Male', 'Male', 'Male']),
		'Salary': pd.Series([50000, 60000, 70000, 80000])}
		
		# create DataFrame from dictionary
		df = pd.DataFrame(data)
		
		# A. Transpose the DataFrame
		df_transpose = df.T
		print("Transposed DataFrame:")
		print(df_transpose)
		
		# B. Print the head and tail of the DataFrame
		print("Head of DataFrame:")

		print(df.head())
		print("Tail of DataFrame:")
		print(df.tail())
		
		# C. Summarize the data in the DataFrame
		print("Summary of DataFrame:")
		print(df.describe())

11	Write a python program to convert a numpy array to a dataframe of given shape.	import pandas as pd
		import numpy as np
		
		# create a NumPy array
		arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
		
		# create a list of column names
		columns = ['A', 'B', 'C']
		
		# create a list of row names
		index = ['Row 1', 'Row 2', 'Row 3', 'Row 4']
		
		# create a Pandas DataFrame from the NumPy array
		df = pd.DataFrame(data=arr, index=index[:arr.shape[0]], columns=columns[:arr.shape[1]])
		
		# print the DataFrame
		print(df)

12	Define Panda Numpy Array and Write a python program to convert a Series to DataFrame and vice versa.	import pandas as pd
		import numpy as np
		
		# create a Pandas Series
		s = pd.Series([1, 2, 3, 4])
		
		# convert the Series to a DataFrame
		df = s.to_frame()
		
		# print the resulting DataFrame

		print(df)
		
		# create a Pandas DataFrame
		df2 = pd.DataFrame({'A': [1, 2, 3, 4], 'B': ['a', 'b', 'c', 'd']})
		
		# convert the DataFrame to a Series
		s2 = df2['A']
		
		# print the resulting Series
		print(s2)

13	Describe Time Series, Time Offset and Time periods in Pandas with suitable program.	Time Series is a data structure in Pandas that represents a sequence of data points indexed in time order. Time series data is commonly found in finance, economics, and other fields where data is collected over time.
		
		Time Offset is a way to represent a duration of time in Pandas. It can be used to represent time differences between two points in time or to create a sequence of time intervals. Time offsets are represented by the pd.DateOffset class in Pandas.
		
		Time Periods represent a fixed length of time in Pandas. They are represented by the pd.Period class and can be used to represent time intervals such as days, months, or years.
		
		import pandas as pd
		import numpy as np
		
		# create a DatetimeIndex for a Time Series
		dates = pd.date_range('2022-01-01', periods=5, freq='D')
		
		# create a Time Series with random data
		ts = pd.Series(np.random.randn(5), index=dates)
		
		# print the Time Series
		print(ts)
		
		# create a Time Offset

		offset = pd.DateOffset(days=1)
		
		# add the offset to a date
		new_date = pd.Timestamp('2022-01-01') + offset
		
		# print the new date
		print(new_date)
		
		# create a Time Period
		period = pd.Period('2022-01')
		
		# create a DataFrame with Time Periods as the index
		df = pd.DataFrame(np.random.randn(31, 2), index=pd.period_range('2022-01', freq='D', periods=31))
		
		# print the DataFrame
		print(df)

14	Explore the different ways to sort the DataFrame in Panda.	1. sort_values(): This method sorts the DataFrame by the values along a specific axis. The by parameter specifies the column or columns to sort by: import pandas as pd df = pd.DataFrame({'col1': [1, 2, 3], 'col2': [4, 3, 1]}) df_sorted = df.sort_values(by='col1') print(df_sorted) 2. sort_index(): This method sorts the DataFrame by the row index or column index. The axis parameter specifies the axis to sort along. import pandas as pd df = pd.DataFrame({'col1': [1, 2, 3], 'col2': [4, 3, 1]}, index=['row3', 'row2', 'row1']) df_sorted = df.sort_index() print(df_sorted) 3. nlargest(): This method returns the n largest elements from the DataFrame. The n parameter specifies the number of elements to return, and the columns parameter specifies the columns to use for sorting. import pandas as pd df = pd.DataFrame({'col1': [1, 2, 3], 'col2': [4, 3, 1]}) df_sorted = df.nlargest(2, columns=['col1']) print(df_sorted)

		4. nsmallest(): This method returns the n smallest elements from the DataFrame. The n parameter specifies the number of elements to return, and the columns parameter specifies the columns to use for sorting. import pandas as pd df = pd.DataFrame({'col1': [1, 2, 3], 'col2': [4, 3, 1]}) df_sorted = df.nsmallest(2, columns=['col1']) print(df_sorted) 5. sort_values() with ascending=False: This method sorts the DataFrame in descending order. import pandas as pd df = pd.DataFrame({'col1': [1, 2, 3], 'col2': [4, 3, 1]}) df_sorted = df.sort_values(by='col1', ascending=False) print(df_sorted)

15	Discuss about the following functions in Numpy with suitable code and output - numpy.char.multiply(), numpy.power() and numpy.arrange(start, stop, step, dtype).	numpy.char.multiply():
		The numpy.char.multiply() function returns the string with multiple copies concatenated. The function takes two arguments, the first argument is the string that needs to be repeated and the second argument is the number of times that the string needs to be repeated.
		import numpy as np
		# repeat the string 'Hello' 3 times
		result = np.char.multiply('Hello ', 3)
		print(result)
		
		numpy.power():
		The numpy.power() function returns the element-wise exponential power of an array. The function takes two arguments, the first argument is the base of the power and the second argument is the exponent.
		import numpy as np
		# calculate the exponential power of an array
		arr = np.array([2, 3, 4])
		result = np.power(arr, 3)
		print(result)
		
		numpy.arange(start, stop, step, dtype):

		The numpy.arange() function returns evenly spaced values within a given interval. The function takes four arguments, the first argument is the start of the interval, the second argument is the end of the interval, the third argument is the step size, and the fourth argument is the data type of the resulting array.
		import numpy as np
		# create an array of evenly spaced values between 1 and 10 with a step of 2
		arr = np.arange(1, 10, 2)
		print(arr)

16	Brief on Data Aggregation in Pandas.	I n Pandas, data aggregation refers to the process of transforming data from one or more data sources into a summary form that provides insights about the data. It involves grouping data based on one or more attributes and then computing summary statistics, such as mean, median, mode, variance, and standard deviation, for each group. The Pandas library provides several functions to perform data aggregation, including groupby(), agg(), apply(), transform(), and pivot_table(). The groupby() function is used to group the data based on one or more attributes and then perform operations on the grouped data. The agg() function is used to apply one or more aggregation functions to the grouped data, such as mean(), sum(), count(), and max(). The apply() function is used to apply a custom function to the grouped data. The transform() function is used to apply a function to each group separately and return a Series with the same shape as the original data. The pivot_table() function is used to create a spreadsheet-style pivot table based on the data. Data aggregation is a powerful technique that allows us to gain insights into our data and make informed decisions based on the insights. It is widely used in data analysis, data science, and business intelligence.
17	Describe on Pandas Indexing, Multiindexing and Reindexing.	Pandas provides various indexing methods to access and manipulate data in a DataFrame or Series. Indexing allows us to select subsets of data from the DataFrame or Series, perform operations on the selected data, and modify the data if required. Indexing: In Pandas, indexing refers to selecting a subset of rows or columns from a DataFrame or Series based on their labels or positions. We can use the loc[] and iloc[] attributes to perform label-based and position-based indexing, respectively. The loc[] attribute allows us to select rows and columns based on their labels, whereas the iloc[] attribute allows us to select rows and columns based on their positions.

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]}, index=['X', 'Y', 'Z'])
# label-based indexing
print(df.loc['X', 'A'])
print(df.loc[:, 'B'])
# position-based indexing
print(df.iloc[0, 1])
print(df.iloc[:, 2])
```

Multiindexing:

Multiindexing in Pandas refers to indexing a DataFrame or Series using multiple levels of hierarchical labels. We can create a multi-index DataFrame using the MultiIndex() function and then use the loc[] attribute to access data at different levels of the index.

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]}, index=[['X', 'X', 'Y'], [1, 2, 1]])
# access data at different levels of the index
print(df.loc['X', :])
print(df.loc[('X', 2), 'B'])
```

Reindexing:

Reindexing in Pandas refers to creating a new DataFrame or Series by changing the order of the existing index or creating a new index altogether. We can use the reindex() method to perform reindexing. We can pass a new index to the reindex() method to create a new DataFrame or Series with the same data but a different index.

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]}, index=['X', 'Y', 'Z'])

# reindexing the DataFrame
new_index = ['Z', 'Y', 'X']
df_reindexed = df.reindex(new_index)
print(df_reindexed)
```

18	Describe Data Operations in Pandas.	Data operations in Pandas are essential for working with tabular data, and Pandas provides many functions to handle various data operations. Some of the common data operations in Pandas are:
		
		Filtering Data:
		Filtering data in Pandas refers to selecting a subset of rows or columns based on certain conditions. We can use boolean indexing or the query() method to filter data in Pandas.
		
		import pandas as pd
		df = pd.read_csv('data.csv')
		
		# filtering data using boolean indexing
		df_filtered = df[df['age'] > 30]
		
		# filtering data using query() method
		df_filtered = df.query('age > 30')
		
		Aggregating Data:
		Aggregating data in Pandas refers to summarizing data based on some function or criterion. We can use the groupby() method to group data based on one or more columns and then apply aggregation functions like sum(), mean(), min(), max(), etc. to the grouped data.
		
		import pandas as pd
		df = pd.read_csv('data.csv')
		
		# grouping data and calculating mean of each group
		df_grouped = df.groupby('gender').mean()
		
		Merging Data:
		Merging data in Pandas refers to combining two or more DataFrames based on some common column(s). We can use the merge() method to merge DataFrames based on one or more common columns.
		
		import pandas as pd
		df1 = pd.DataFrame({'id': [1, 2, 3], 'name': ['Alice', 'Bob', 'Charlie']})

		df2 = pd.DataFrame({'id': [2, 3, 4], 'age': [25, 30, 35]})
		
		# merging DataFrames based on common column 'id'
		df_merged = pd.merge(df1, df2, on='id')
		
		Reshaping Data:
		Reshaping data in Pandas refers to transforming data from one shape to another. We can use the pivot(), stack(), unstack(), and melt() methods to reshape data in Pandas.
		
		import pandas as pd
		df = pd.DataFrame({'id': [1, 2, 3], 'gender': ['F', 'M', 'M'], 'income': [50000, 60000, 70000]})
		
		# reshaping data using pivot()
		df_pivoted = df.pivot(index='id', columns='gender', values='income')
		
		Handling Missing Data:
		Handling missing data in Pandas refers to filling or removing missing or null values in the DataFrame. We can use the fillna() method to fill missing values, and the dropna() method to remove rows or columns with missing values.
		
		import pandas as pd
		df = pd.DataFrame({'id': [1, 2, 3, 4], 'age': [25, None, 30, 35], 'income': [50000, 60000, None, 70000]})
		
		# filling missing values with mean of column
		df_filled = df.fillna(df.mean())
		
		# removing rows with missing values
		df_dropped = df.dropna()

19	Describe GroupBy in Pandas by writing a suitable program.	GroupBy in Pandas is a powerful feature that allows us to group a DataFrame based on one or more columns and then apply aggregation functions to the grouped data. Aggregation functions like sum(), mean(), min(), max(), etc. can be applied to the grouped data, providing a convenient way to analyze and summarize large datasets.

		Let's consider a sample dataset of employee information, where each row represents an employee and the columns represent their name, department, salary, and age.
		
		import pandas as pd
		data = {'name': ['Alice', 'Bob', 'Charlie', 'David', 'Emily', 'Frank'],
		'department': ['Sales', 'Marketing', 'Sales', 'Marketing', 'Sales', 'Marketing'],
		'salary': [50000, 60000, 55000, 65000, 60000, 70000],
		'age': [25, 30, 35, 40, 45, 50]}
		df = pd.DataFrame(data)
		print(df)
		
		Now, let's group the above DataFrame by department and calculate the mean salary for each department using the groupby() method.
		# grouping by department and calculating mean salary
		grouped_data = df.groupby('department')['salary'].mean()
		print(grouped_data)
		
		# grouping by department and age, and calculating total salary and max age
		grouped_data = df.groupby(['department', 'age']).agg({'salary': 'sum', 'age': 'max'})
		print(grouped_data)

20	Demonstrate ways to create new columns derived from existing columns in Pandas.	**Using arithmetic operators:** We can create a new column by performing arithmetic operations on existing columns. For example, if we have two columns A and B, we can create a new column C which is the sum of A and B as follows: import pandas as pd df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]}) df['C'] = df['A'] + df['B'] print(df) **Using functions**: We can create a new column by applying a function to one or more existing columns. For example, we can create a new column D which is the product of A and B as follows: def multiply_columns(row): return row['A'] * row['B']

		df['D'] = df.apply(multiply_columns, axis=1) print(df) **Using the assign method:** We can create a new column using the assign method. For example, we can create a new column E which is the difference between A and B as follows: df = df.assign(E=df['A'] - df['B']) print(df)

21	Develop a bar chart for the following data.
	movies = ["Annie Hall", "Ben-Hur", "Casablanca", "Gandhi", "West Side Story"]
	num_oscars = [5, 11, 3, 8, 10].
		import matplotlib.pyplot as plt
		
		movies = ["Annie Hall", "Ben-Hur", "Casablanca", "Gandhi", "West Side Story"]
		num_oscars = [5, 11, 3, 8, 10]
		
		# set the positions of the bars on the x-axis
		x_pos = [i for i, _ in enumerate(movies)]
		
		# create a bar chart
		plt.bar(x_pos, num_oscars)
		
		# add labels for x and y axes
		plt.xlabel("Movies")
		plt.ylabel("# of Oscars")
		
		# add a title
		plt.title("Movies and Oscars")
		
		# add labels for the bars
		plt.xticks(x_pos, movies)
		
		# display the bar chart
		plt.show()

22	Explain the concept of NumPy Broadcasting with suitable examples.	NumPy broadcasting is a powerful mechanism that allows numpy arrays of different shapes to be used in arithmetic operations. When performing operations on arrays of different shapes, numpy automatically broadcasts them to make them compatible, which can save a lot of memory and computational resources.
		
		The following are some examples that demonstrate the concept of numpy broadcasting:
		Scalar and Array:
		
		import numpy as np
		# create a 3x3 numpy array
		arr1 = np.array([[1, 2, 3],
		[4, 5, 6],
		[7, 8, 9]])
		# multiply the array by a scalar value
		arr2 = arr1 * 2
		print(arr2)
		
		Array and Array:
		import numpy as np
		
		# create a 3x3 numpy array
		arr1 = np.array([[1, 2, 3],
		[4, 5, 6],
		[7, 8, 9]])
		
		# create a 1x3 numpy array
		arr2 = np.array([2, 2, 2])
		
		# add the two arrays
		arr3 = arr1 + arr2
		
		print(arr3)
		
		Incompatible Shapes:
		import numpy as np

		# create a 3x3 numpy array
		arr1 = np.array([[1, 2, 3],
		[4, 5, 6],
		[7, 8, 9]])
		
		# create a 1x2 numpy array
		arr2 = np.array([2, 2])
		
		# add the two arrays
		arr3 = arr1 + arr2

23	Exemplify Creating a series from dictionaries in Pandas by writing suitable code snippets. Perform column addition and removal of your choice.	import pandas as pd
		
		# Create a dictionary
		fruits_dict = {'apples': 10, 'bananas': 15, 'oranges': 20}
		
		# Convert the dictionary to a Pandas Series
		fruits = pd.Series(fruits_dict)
		
		# Print the Series
		print(fruits)
		
		Now, let's perform some column addition and removal on this Series:
		# Add a new fruit to the Series
		fruits['pears'] = 12
		
		# Print the updated Series
		print(fruits)
		
		# Remove oranges from the Series
		fruits = fruits.drop('oranges')
		
		# Print the updated Series
		print(fruits)

24	Describe Supervised learning. Explain few of them.	Supervised learning is a type of machine learning algorithm where the model is trained on a labeled dataset. In supervised learning, the model learns from the input-output pairs in the training dataset to predict the output for new, unseen inputs. The goal of supervised learning is to create a mapping function from input variables to an output variable. There are several types of supervised learning algorithms, including: Regression: In regression, the goal is to predict a continuous output variable based on one or more input variables. Linear regression, polynomial regression, and decision tree regression are some examples of regression algorithms. Classification: In classification, the goal is to predict a discrete output variable based on one or more input variables. Logistic regression, decision trees, random forests, and support vector machines are some examples of classification algorithms. Naive Bayes: Naive Bayes is a classification algorithm that is based on the Bayes theorem. It assumes that the features are independent of each other and predicts the class based on the probability of the input features. K-Nearest Neighbors (KNN): KNN is a classification algorithm that is based on the idea of similarity. It predicts the class of a new data point based on the class of its k nearest neighbors in the training set. Neural Networks: Neural networks are a type of machine learning algorithm that are modeled after the human brain. They can be used for both regression and classification tasks.
25	Describe UnSupervised learning. Explain few of them.	Unsupervised learning is a type of machine learning algorithm where the model is trained on an unlabeled dataset. In unsupervised learning, the model learns patterns and relationships in the input data without being given explicit output labels. The goal of unsupervised learning is to explore and find hidden structures in the data. There are several types of unsupervised learning algorithms, including:

		Clustering: In clustering, the goal is to group similar data points together based on some similarity metric. K-means clustering, hierarchical clustering, and density-based clustering are some examples of clustering algorithms. Dimensionality Reduction: In dimensionality reduction, the goal is to reduce the number of input variables while preserving the important information. Principal Component Analysis (PCA) and t-SNE are some examples of dimensionality reduction algorithms. Anomaly Detection: In anomaly detection, the goal is to identify data points that are significantly different from the rest of the data. One-class SVM and Isolation Forest are some examples of anomaly detection algorithms. Association Rule Mining: In association rule mining, the goal is to find patterns in the data by identifying frequent itemsets and rules that describe the relationships between them. Apriori and FP-growth are some examples of association rule mining algorithms. Generative Models: Generative models are a type of unsupervised learning algorithm that are used to generate new data points that are similar to the input data. Variational Autoencoder (VAE) and Generative Adversarial Networks (GAN) are some examples of generative models.

26	Demonstrate various NumPy statistical functions such as min, max, range, sd, mean, median and mode by writing a suitable python program.	import numpy as np
		
		# creating an array of random numbers
		arr = np.random.randint(0, 100, size=(10))
		
		# printing the array
		print("Array:", arr)
		
		# finding the minimum value in the array
		print("Minimum value:", np.min(arr))
		
		# finding the maximum value in the array
		print("Maximum value:", np.max(arr))
		
		# finding the range of values in the array

		print("Range of values:", np.ptp(arr))
		
		# finding the standard deviation of the values in the array
		print("Standard deviation:", np.std(arr))
		
		# finding the mean of the values in the array
		print("Mean value:", np.mean(arr))
		
		# finding the median of the values in the array
		print("Median value:", np.median(arr))
		
		# finding the mode of the values in the array
		print("Mode value:", np.mode(arr))

27	What is meant Matplotlib? Give features of Matplotlib.	Matplotlib is a Python data visualization library used for creating static, animated, and interactive visualizations in Python. It is widely used for creating high-quality plots, charts, histograms, and other types of visualizations. Some of the features of Matplotlib are: Wide Range of Plots: Matplotlib offers a wide range of plots such as line, bar, scatter, histogram, heatmaps, and many others. Customization: It offers high-level customization to its users to customize every aspect of the plot such as axes, labels, colors, fonts, and many others. Compatibility: It is compatible with various operating systems, platforms, and data analysis libraries like NumPy and Pandas. Interactive Plots: Matplotlib offers interactive plots which can be embedded in web applications, desktop applications, and notebooks. Highly Performant: Matplotlib is highly performant and can handle large amounts of data without compromising on speed.

		Variety of Outputs: It offers a variety of outputs such as PNG, PDF, SVG, and many others.
28	Discuss the different advantages and disadvantages of the data visualization.	Data visualization has become an essential tool for data analysts and business professionals to communicate complex data insights to a non-technical audience. Here are some of the advantages and disadvantages of data visualization: Advantages: Simplifies Complex Data: Data visualization simplifies complex data into a form that is easy to understand and interpret. It helps in identifying patterns and trends that might not be visible in the raw data. Better Communication: Data visualization makes it easier to communicate data insights to non-technical stakeholders. It helps in presenting data in a way that is more meaningful and relevant to the target audience. Quick Decision Making: Data visualization helps in making quick and informed decisions. It provides a clear picture of the data, making it easier to identify areas of improvement or opportunities. Better Analysis: Data visualization enables analysts to better analyze data by providing a visual representation of the data. It helps in identifying outliers, anomalies, and other patterns that may not be visible in tabular data. Improved Efficiency: Data visualization helps in improving the efficiency of the decision-making process. It enables stakeholders to quickly identify and act upon insights, leading to faster decision-making. Disadvantages: Biases: Data visualization can introduce biases if the data is not presented in an objective manner. Visualizations can be manipulated to tell a specific story or to present a particular point of view. Misinterpretation: Data visualization can be misinterpreted if it is not presented in a clear and concise manner. It is essential to ensure that the visualization accurately represents the underlying data.

		Overreliance: Data visualization can lead to overreliance on the visuals, leading to a lack of critical thinking and analysis. It is important to ensure that the data is properly analyzed before presenting it in a visual format. Complexity: Data visualization can become too complex, making it difficult for the target audience to understand. It is important to ensure that the visualization is simple, clear, and easy to understand. Technical Skills: Data visualization requires technical skills and expertise. It is essential to have the necessary skills and knowledge to create effective visualizations.

29	Develop bar chart for the following data.
	years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
	gdp = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3].
		import matplotlib.pyplot as plt
		
		years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
		gdp = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]
		
		# Create a new figure and axis
		fig, ax = plt.subplots()
		
		# Plot the data as a bar chart
		ax.bar(years, gdp, align='center')
		
		# Set the axis labels and title
		ax.set_xlabel('Year')
		ax.set_ylabel('GDP (in billions of dollars)')
		ax.set_title('Nominal GDP')
		
		# Display the chart
		plt.show()

30	Analyze the concept of dependent and independent variable and also discuss about the simple linear regression	In statistics, a dependent variable is a variable that is being studied or measured and whose value depends on the value of one or more independent variables. An independent variable is a variable that is manipulated or controlled by the researcher in order to study its effect on the dependent variable.

	with its mathematical equation.	In simple linear regression, there is a single independent variable and a single dependent variable. The goal of simple linear regression is to find a linear relationship between the independent variable and the dependent variable. This relationship is represented by the following equation: y = mx + b where y is the dependent variable, x is the independent variable, m is the slope of the line, and b is the y-intercept. The slope of the line represents the rate of change of the dependent variable with respect to the independent variable. The y-intercept represents the value of the dependent variable when the independent variable is zero. The process of finding the best values of m and b to fit the data is called regression analysis. The most common method for finding the best values is least squares regression, which minimizes the sum of the squared differences between the actual y values and the predicted y values. To perform simple linear regression, you would typically start by collecting data on the independent and dependent variables. Then, you would plot the data on a scatter plot to visualize the relationship between the two variables. Finally, you would use regression analysis to find the best fit line for the data and to calculate the slope and y-intercept of the line.

31	Write a python program to import data from json file that contains the mark scores of five students for 5 subjects. Retrieve the student detail having maximum marks.	Assuming the JSON file marks.json contains the following data:
		{
		"Alice": [80, 75, 85, 90, 95],
		"Bob": [70, 65, 80, 75, 85],
		"Charlie": [90, 95, 85, 80, 75],
		"David": [85, 80, 90, 95, 70],
		"Eve": [95, 90, 80, 85, 75]
		}
		
		Code:
		import json
		
		# Load data from JSON file
		with open('marks.json') as f:
		data = json.load(f)

		# Find student with maximum marks
		max_marks = 0
		max_student = None
		
		for student, marks in data.items():
		  total_marks = sum(marks)
		  if total_marks > max_marks:
		  max_marks = total_marks
		  max_student = student
		
		# Print student with maximum marks
		print("Student with maximum marks:")
		print("Name:", max_student)
		print("Marks:", data[max_ student])

32	Write a python program to import data from html file that includes a table having sample menu list of a restaurant. Retrieve the cheaper food item.	import pandas as pd
		
		# read the HTML file and extract the table
		url = 'http://www.abc.com/menu-item-prices/12345/sample-restaurant'
		table = pd.read_html(url)[0]
		
		# find the cheapest food item
		cheapest_item = table.loc[table['Price'].idxmin()]
		
		# print the cheapest item details
		print(f"The cheapest food item is {cheapest_item['Item']} priced at {cheapest_item['Price']}.")

33	Analyse Regression technique in ML and its types in detail.	Regression is a supervised machine learning technique that is used to model the relationship between a dependent variable and one or more independent variables. The primary objective of regression analysis is to predict the value of the dependent variable based on the values of the independent variables. There are several types of regression techniques in machine learning. Let's discuss some of them in detail. Linear Regression:

		Linear regression is one of the most widely used regression techniques in machine learning. It models the relationship between a dependent variable and one or more independent variables in the form of a linear equation. The linear equation can be represented as Y = a + bX, where Y is the dependent variable, X is the independent variable, a is the intercept, and b is the slope of the line. Linear regression can be further classified into two types: simple linear regression and multiple linear regression. Polynomial Regression: Polynomial regression is used when the relationship between the dependent variable and independent variables is non-linear. It models the relationship between the dependent variable and independent variables in the form of a polynomial equation of degree n. Polynomial regression can be represented as Y = a + b1X + b2X^2 + ... + bnX^n. Logistic Regression: Logistic regression is used when the dependent variable is categorical. It models the probability of an event occurring based on the values of the independent variables. Logistic regression can be binary or multinomial. Ridge Regression: Ridge regression is used when there is multicollinearity among the independent variables. It adds a penalty term to the cost function to prevent overfitting. The penalty term is proportional to the square of the coefficients. Lasso Regression: Lasso regression is similar to ridge regression. It adds a penalty term to the cost function to prevent overfitting. The penalty term is proportional to the absolute value of the coefficients. Lasso regression can be used to perform feature selection. Elastic Net Regression: Elastic net regression is a combination of ridge regression and lasso regression. It adds both L1 and L2 regularization terms to the cost function to prevent overfitting.
34	Explain the Multiple Regression with suitable example.	Multiple Regression is a type of regression analysis used to analyze the relationship between two or more independent variables and a dependent variable. It is used to predict the outcome of a dependent variable based on the values of two or more independent variables.

Consider a scenario where we want to predict the salary of a person based on their years of experience and education level. Here, we can consider years of experience and education level as independent variables and salary as a dependent variable.

Let's take an example of a dataset containing the following variables:

Salary (dependent variable)
Years of Experience (independent variable)
Education Level (independent variable)
We can perform multiple regression using the following steps:

Step 1: Import necessary libraries and load the dataset.

```
import pandas as pd
import numpy as np
import statsmodels.api as sm

data = pd.read_csv('salary.csv')
```

Step 2: Define the independent variables and dependent variable.

```
X = data[['Experience', 'Education']]
y = data['Salary']
```

Step 3: Fit the multiple regression model and obtain the summary.

```
X = sm.add_constant(X) # adding a constant

model = sm.OLS(y, X).fit()
predictions = model.predict(X)

print_model = model.summary()
print(print_model)
```

Step 4: Interpret the model summary.

		The model summary contains information about the regression coefficients, standard errors, t-statistics, p-values, and the overall fit of the model. For example, the regression equation might look like this: Salary = 22.51 + (10.12 * Experience) + (15.05 * Education)
35	Distinguish Between Classification and Regression.	Classification and Regression are two important machine learning techniques used for predicting the outcome of a given input data. The main differences between the two are: Definition: Regression is a statistical method that helps in identifying the relationship between a dependent variable and one or more independent variables. Classification, on the other hand, is a method used to identify the category or class to which the data belongs. Output: The output of regression is a continuous value, which means that the predicted value can take on any numerical value. The output of classification is discrete, which means that it can only be one of a limited set of values or categories. Input Data: Regression is used when the input data is continuous, while classification is used when the input data is categorical. Examples: A classic example of regression is predicting a person's weight based on their height, age, and gender. An example of classification is identifying whether an email is spam or not based on the contents of the email. Types: Regression can be of two types, simple linear regression and multiple linear regression, whereas classification can be of several types such as binary classification, multi-class classification, and multi-label classification. In summary, regression is used to predict a continuous value, while classification is used to predict the category or class to which a data point belongs.