Test Series 1

Q.1 Difference between Python version 2 and version 3 ?

Ans: (1). P2 : ==> Xrange() use for loop . , P3: ==> range() use for loop .

(2). P2 : ==> Work on ASCII , P3 : ==> Work on Unicode

(3). P2 : ==> print() not compulsory , P3 : ==> print() compulsory

How do you check current version of python ?

Ans: Go to command prompt => enter => python –version

Q.2 Difference between local variable vs global variable ?examples ?

Ans: **Local variable**

1. Local variables are defined within a specific function or block.
2. They are only accessible within that function or block.
3. Local variables have a limited scope and are destroyed once the function or block execution is completed.

Eg:

```
def my_function():
    x = 10  # Local variable
    print(x)
my_function()  # Output: 10
print(x)  # Error: x is not defined
```

Globall variable:

1. Global variables are defined outside any function or block.
2. They can be accessed from any part of the program, including functions or blocks.
3. Global variables have a global scope and can be used throughout the program.

Eg:

```
x = 10 # Global variable
def my_function():
    print(x)
my_function()  # Output: 10
print(x) # Output: 10
```

It's important to note that if a local variable has the same name as a global variable, the local variable will take precedence within its scope. To access the global variable in such cases, you can use the global keyword.

Q.3 What are the data types in python ?

Ans: following are the data type in python:

1. Integer - represents whole numbers, e.g., 1, 10, -5.
2. Float - represents floating-point numbers, e.g., 3.14, -2.5, 0.0.
3. String - represents a sequence of characters, enclosed in single quotes ('') or double quotes ("").
4. Boolean - represents either True or False.
5. List - represents an ordered collection of elements, enclosed in square brackets ([]).
6. Tuple - represents an ordered, immutable collection of elements, enclosed in parentheses ().
7. Dictionary - represents a collection of key-value pairs, enclosed in curly braces ({}) where each key is unique.

Q.4 What is difference between Mutable and Immutable Data types ? How many data types are mutable or not ?Please provide reason with examples?

Ans: In Python, mutable data types are the ones that can be modified after they are created, while immutable data types cannot be changed once they are assigned a value.

The mutable data types in Python include lists, sets, and dictionaries. Let's take a look at an example with a list:

```
my_list = [1, 2, 3]

print(my_list)  # Output: [1, 2, 3]

my_list.append(4)

print(my_list)  # Output: [1, 2, 3, 4]

my_list[0] = 5

print(my_list)  # Output: [5, 2, 3, 4]
```

As you can see, the list can be modified by adding elements using append() or by changing the value at a specific index.

On the other hand, immutable data types in Python include strings, tuples, and frozensets. Here's an example with strings:

```
my_string = "Hello"

print(my_string)  # Output: Hello

my_string += " World"

print(my_string)  # Output: Hello World

# Trying to change a character at a specific index will result in an error

my_string[0] = "J"  # Error: 'str' object does not support item assignment
```

In this case, the string cannot be modified directly, and attempting to change a character at a specific index will result in a TypeError.

To summarize, the key difference is that mutable data types allow modifications after creation, while immutable data types do not. Lists, sets, and dictionaries are mutable, while strings, tuples, and frozensets are immutable.

Q.5 Why we use loops ? Please provide 3 examples of for loop and 3 examples of while loop ?

Ans: Loops are used in programming to repeat a specific block of code multiple times. They are useful when we want to perform a certain action repeatedly until a certain condition is met. Here are three examples of for loops and three examples of while loops:

Examples of for loops:

1. Printing numbers from 1 to 10: for i in range(1, 11): print(i)
2. Iterating through a list and performing an action on each element: fruits = ["apple", "banana", "orange"] for fruit in fruits: print("I like", fruit)
3. Calculating the sum of numbers in a given range: sum = 0 for i in range(1, 6): sum += i print("Sum:", sum)

Examples of while loops:

1. Printing numbers from 1 to 5: i = 1 while i <= 5: print(i) i += 1
2. Asking for user input until a specific condition is met: correct_password = "password123" user_input = input("Enter password: ") while user_input != correct_password: print("Incorrect password. Try again.") user_input = input("Enter password: ") print("Access granted!")
3. Finding the first power of 2 that is greater than 100: power = 1 while power <= 100: power *= 2 print("First power of 2 greater than 100:", power)

Q.6 Why we use functions ? Difference between lambda , filter , reduce , map function with examples ?

Ans:

Functions are used in programming to group a set of instructions together and give them a name, allowing us to reuse the code and make our programs more modular and organized. They help in breaking down complex tasks into smaller, manageable parts.

Now, let's discuss the differences between lambda, filter, reduce, and map functions:

Lambda Functions:

1. Lambda functions, also known as anonymous functions, are functions without a name.
2. They are defined inline and are typically used for simple and one-time operations.
3. Lambda functions are defined using the keyword lambda.

Example: double = lambda x: x * 2 result = double(5) print(result) # Output: 10

Filter Function:

1. The filter() function is used to filter out elements from a sequence based on a specific condition.
2. It takes a function and an iterable as arguments and returns an iterator with the elements that satisfy the condition.

Example: numbers = [1, 2, 3, 4, 5, 6] even_numbers = list(filter(lambda x: x % 2 == 0, numbers)) print(even_numbers) # Output: [2, 4, 6]

Reduce Function:

1. The reduce() function is used to apply a function to an iterable and reduce it to a single cumulative value.
2. It takes a function and an iterable as arguments and returns a single value.

3. Note: In Python 3, the reduce() function is part of the functools module, so you need to import it before using.

Example: from functools import reduce numbers = [1, 2, 3, 4, 5] sum_of_numbers = reduce(lambda x, y: x + y, numbers) print(sum_of_numbers) # Output: 15

Map Function:

1. The map() function is used to apply a function to each element in an iterable and return a new iterator with the results.
2. It takes a function and an iterable as arguments and returns an iterator with the modified elements.

Example: numbers = [1, 2, 3, 4, 5] squared_numbers = list(map(lambda x: x ** 2, numbers)) print(squared_numbers) # Output: [1, 4, 9, 16, 25]

Q.7 Difference between list and tuple ?Please provide index() , insert() , append(), reverse() , pop() explain with examples ?

Ans:

A **list** in Python is a mutable data type, meaning you can modify its elements. It is represented by square brackets [] and can store elements of different data types. Here's an example:

my_list = [1, 2, 3, 4, 5]

On the other hand, a tuple is an immutable data type, meaning its elements cannot be modified once it is created. It is represented by parentheses () and can also store elements of different data types. Here's an example:

my_tuple = (1, 2, 3, 4, 5)

1. index(): This method returns the index of the first occurrence of a specified element within the list or tuple.

Example: my_list = [10, 20, 30, 40, 50, 20] print(my_list.index(20)) # Output: 1

my_tuple = (10, 20, 30, 40, 50, 20) print(my_tuple.index(20)) # Output: 1 ```

2. insert(): This method inserts an element at a given index within the list. For tuples, since they are immutable, this operation is not possible.

Example: my_list = [1, 2, 3] my_list.insert(1, 10) print(my_list) # Output: [1, 10, 2, 3] my_tuple = (1, 2,

3) # Tuples cannot be modified ```

3. append(): This method adds an element at the end of the list. Tuples, being immutable, do not have an append() method.

Example: my_list = [1, 2, 3] my_list.append(4) print(my_list) # Output: [1, 2, 3, 4]my_tuple =

(1, 2, 3) # Tuples cannot be modified ```

4. reverse(): This method reverses the order of elements in the list. Since tuples are immutable, they do not have a reverse() method.

Example: my_list = [1, 2, 3, 4, 5] my_list.reverse() print(my_list) # Output: [5, 4, 3, 2, 1]my_tuple = (1,

2, 3) # Tuples cannot be modified ```

5. pop(): This method removes and returns the element at the specified index within the list. For tuples, as they are immutable, this operation is not possible.

Example: my_list = [1, 2, 3] removed_element = my_list.pop(1) print(my_list) # Output: [1, 3] print(removed_element) # Output: 2

my_tuple = (1, 2, 3) # Tuples cannot be modified ```

Q.8 Difference between dictionary vs set ? Please provide keys() , values() ,items() example of dictionary and union() , intersection() in sets ? Is FrozenSet is mutable or not ?

Ans: A **dictionary** is a collection of key-value pairs where each key is unique, whereas a set is a collection of unique elements with no specific order.

For a dictionary, you can use the keys(), values(), and items() methods to access its keys, values, and key-value pairs, respectively.

Dictionary example

my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}

Accessing keys

print(my_dict.keys()) # Output: ['name', 'age', 'city']

Accessing values

print(my_dict.values()) # Output: ['John', 25, 'New York']

Accessing key-value pairs

print(my_dict.items()) # Output: [('name', 'John'), ('age', 25), ('city', 'New York')]

On the other hand, **sets** provide various operations such as union() and intersection().

Set examples

set1 = {1, 2, 3, 4}

set2 = {3, 4, 5, 6}

Union of two sets

union_set = set1.union(set2)

print(union_set) # Output: {1, 2, 3, 4, 5, 6}

Intersection of two sets

intersection_set = set1.intersection(set2)

print(intersection_set) # Output: {3, 4}

Regarding the **FrozenSet**, it is an immutable variant of the set. It means that once a FrozenSet is created, you cannot modify its elements. FrozenSets are created using the frozenset() function:

frozen_set = frozenset([1, 2, 3])

print(frozen_set) # Output: frozenset({1, 2, 3})

Q.9 What are decorators ? Please provide example ?

Ans: Decorators in Python are functions that modify the behavior of other functions or classes without changing their source code. They allow us to add additional functionality to existing functions or classes by wrapping them with another function.

Eg:

```
def decorator_function(original_function):
    def wrapper_function():
        print("Before the function execution")
        original_function()
        print("After the function execution")
    return wrapper_function
@decorator_function
def hello():
    print("Hello, world!")
hello()
```

In this example, we define a decorator function decorator_function that takes an original function as its parameter. It defines a wrapper function wrapper_function that adds some extra functionality before and after the original function is executed. The wrapper function is then returned.

The @decorator_function syntax is used to apply the decorator to the hello function. This is equivalent to writing hello = decorator_function(hello). When we call hello(), it will execute the decorated version of the hello function, which prints "Before the function execution", then calls the original hello function that prints "Hello, world!", and finally prints "After the function execution".

Decorators are useful for adding logging, timing, authentication, or other cross-cutting concerns to functions or classes without modifying their code directly. They provide a way to enhance the functionality of existing code in a clean and modular manner.

Q.10 What is list comprehension ?Please provide example ?

Ans: List comprehension is a concise way to create lists in Python. It allows you to create a new list by applying an expression to each item in an existing list, along with optional conditional statements. Here's an example:

Let's say we have a list of numbers [1, 2, 3, 4, 5], and we want to create a new list that contains the squares of these numbers. We can use list comprehension to achieve this:

```
numbers = [1, 2, 3, 4, 5]
squares = [x**2 for x in numbers]
```

In this example, [x**2 for x in numbers] is the list comprehension expression. It iterates over each element x in the numbers list and squares it (x**2), creating a new list called squares with the squared values.

The resulting squares list will be [1, 4, 9, 16, 25].

Q.11 How do you create an array from a user defined list?

Ans: To create an array from a user-defined list in Python, you can use the array module. Here's an example:

import array

User-defined list

user_list = [1, 2, 3, 4, 5]

Creating an array from the list

arr = array.array('i', user_list)

'i' represents the type code for signed integers

print(arr)

This code snippet will create an array (arr) from a user-defined list (user_list) using the array module in Python. The type code 'i' represents signed integers in the array.

Q.12 Difference between hstack() vs vstack() with example ?

Ans:

The **hstack()** and **vstack()** functions are used for stacking arrays horizontally and vertically, respectively.

**hstack(**) is used to horizontally stack arrays, meaning it concatenates arrays along the horizontal axis. This means that it combines arrays column-wise. Here's an example:

import numpy as np

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

result = np.hstack((a, b))

print(result)

Output: [1 2 3 4 5 6]

vstack() is used to vertically stack arrays, meaning it concatenates arrays along the vertical axis. This means that it combines arrays row-wise. Here's an example:

import numpy as np

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

result = np.vstack((a, b))

print(result)

Output: [[1 2 3] [4 5 6]]

In summary, hstack() combines arrays column-wise, while vstack() combines arrays row-wise.

Q.13 What is Broadcasting in NumPy ? Please provide an example ?

Ans:

Broadcasting in NumPy refers to the ability of NumPy arrays to perform arithmetic operations on arrays with different shapes. When performing element-wise operations, NumPy automatically adjusts the shapes of the arrays to make them compatible. This eliminates the need for explicit looping over the arrays.

import numpy as np

Creating two NumPy arrays

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

Performing element-wise addition using broadcasting

result = a + b

print(result)

Output: [5 7 9]

In the example above, the arrays a and b have different shapes, but NumPy automatically broadcasts them to perform the addition operation. The result is a new array [5, 7, 9], which is the element-wise sum of a and b.

Q.14 What is zeros() , ones() , eye() , diag() , randint() , rand() , seed(), linspace() , unique() in NumPy ? please provide explanation with example ?

1. **zeros():** This function creates an array filled with zeros.

Example: ```python import numpy as np

arr = np.zeros((2, 3)) print(arr) Output: [[0. 0. 0.] [0. 0. 0.]] ```

2. **ones():** This function creates an array filled with ones.

Example: ```python import numpy as np

arr = np.ones((2, 3)) print(arr) Output: [[1. 1. 1.] [1. 1. 1.]] ```

3. **eye():** This function creates an identity matrix, which is a square matrix with ones on the diagonal and zeros elsewhere.

Example: ```python import numpy as np

arr = np.eye(3) print(arr) Output: [[1. 0. 0.] [0. 1. 0.] [0. 0. 1.]] ```

4. **diag():** This function extracts the diagonal from a given array or creates a diagonal array from a given list.

Example: ```python import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) diagonal = np.diag(arr) print(diagonal) Output: [1 5 9] ```

5. **randint():** This function generates random integers within a specified range.

Example: ```python import numpy as np

rand_int = np.random.randint(1, 10, size=(2, 3)) print(rand_int) Output: [[4 6 7] [5 3 2]] ```

6. **rand():** This function generates random numbers from a uniform distribution between 0 and 1.

Example: ```python import numpy as np

rand_num = np.random.rand(2, 3) print(rand_num) Output: [[0.43234262 0.87541347 0.23429295] [0.57263289 0.87640961 0.15698545]] ```

7. **seed():** This function is used to set the random seed, allowing reproducibility of random numbers.

Example: ```python import numpy as np

np.random.seed(42) rand_num = np.random.rand(2, 3) print(rand_num) Output: [[0.37454012 0.95071431 0.73199394] [0.59865848 0.15601864 0.15599452]] ```

8. **linspace():** This function generates a linearly spaced array between two specified values.

Example: ```python import numpy as np

arr = np.linspace(1, 5, num=10) print(arr) Output: [1. 1.44444444 1.88888889 2.33333333 2.77777778 3.22222222 3.66666667 4.11111111 4.55555556 5.] ```

9. **unique():** This function returns the unique elements of an array in sorted order.

Example: ```python import numpy as np

arr = np.array([1, 2, 3, 2, 4, 1, 5, 6, 3, 6]) unique_values = np.unique(arr) print(unique_values) Output: [1 2 3 4 5 6] ```

Q.15 How can you reshape your array in NumPy? What is argmin() and argmax() in NumPy ? Please explain with example ?

Ans:

To reshape an array in NumPy, you can use the **reshape()** function. This function allows you to change the shape (dimensions) of an array without modifying its data. Here's an example:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

reshaped_arr = arr.reshape((2, 3))

print(reshaped_arr)
```

Output:

```
[[1 2 3]

 [4 5 6]]
```

In this example, we start with a 1-dimensional array arr and reshape it into a 2-dimensional array with dimensions (2, 3).

The **argmin()** function returns the indices of the minimum values along a specified axis or in a flattened array. Similarly, the **argmax()** function returns the indices of the maximum values. Here's an example to illustrate their usage:

```
import numpy as np

arr = np.array([5, 2, 9, 1, 7])
```

```
min_index = np.argmin(arr)

max_index = np.argmax(arr)

print("Minimum value:", arr[min_index], "at index:", min_index)

print("Maximum value:", arr[max_index], "at index:", max_index)
```

Output:

Minimum value: 1 at index: 3

Maximum value: 9 at index: 2

In this example, we have an array arr, and we use argmin() and argmax() functions to find the minimum and maximum values along the flattened array. We then print the values and their corresponding indices.

Test Series 2

Q.1 What is Pandas ?

Ans:

Pandas is a Python library that provides data manipulation and analysis tools. It offers data structures and functions to efficiently handle and analyze structured data, such as numerical tables and time series data. It simplifies tasks like data cleaning, filtering, grouping, and visualization, making it a powerful tool for working with data in Python.

Q.2 Difference between series and dataframe ?

Ans:

In Python, both **Series** and **DataFrame** are data structures provided by the pandas library for data manipulation and analysis.

A **Series** is a one-dimensional labeled array that can hold any data type. It is similar to a column in a spreadsheet or a dictionary with a key-value pair. Each element in a Series has a unique label called an index.

Here's an example of a Series in Python:

```
import pandas as pd

data = [10, 20, 30, 40, 50]

series = pd.Series(data)

print(series)
```

Output: 0 10 1 20 2 30 3 40 4 50 dtype: int64

A **DataFrame**, on the other hand, is a two-dimensional labeled data structure with columns of potentially different data types. It can be seen as a table, similar to a spreadsheet or a SQL table. A DataFrame consists of multiple Series aligned together.

Here's an example of a DataFrame in Python:

```
import pandas as pd

data = {'Name': ['John', 'Emily', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'London', 'Sydney']}

df = pd.DataFrame(data)

print(df)
```

Output: Name Age City 0 John 25 New York 1 Emily 30 London 2 Charlie 35 Sydney

In summary, a Series represents a single column of data, while a DataFrame is a collection of Series arranged in a tabular form.

Q.3 Difference between loc and iloc ?

Ans:

loc and **iloc** are two indexing methods used in pandas for selecting data from a DataFrame.

loc is label-based indexing, which means that you can select data based on the row and column labels. You can provide the row label(s) and column label(s) as arguments to the loc function to access specific data. For example:

```
import pandas as pd


# Creating a sample DataFrame

data = {'Name': ['John', 'Emily', 'Michael', 'Jessica'],
        'Age': [25, 30, 35, 40],
        'City': ['New York', 'London', 'Paris', 'Tokyo']}

df = pd.DataFrame(data)

# Using loc to select data

print(df.loc[1, 'City'])  # Output: London

print(df.loc[2:3, 'Name'])  # Output:

# 2   Michael

# 3   Jessica

# Name: Name, dtype: object
```

iloc, on the other hand, is integer-based indexing, which means that you can select data based on the integer positions of the rows and columns. You can provide the row index(es) and column index(es) as arguments to the iloc function to access specific data. For example:

```
import pandas as pd


# Using iloc to select data

print(df.iloc[1, 2])  # Output: London
```

print(df.iloc[2:3, 0]) # Output:

2   Michael

Name: Name, dtype: object

In summary, loc uses labels to select data, while iloc uses integer positions.

Q.4 what does value_counts() ?

Ans: value_counts() is a function in Python that is used to count the number of occurrences of each unique value in a Pandas Series or DataFrame column. It returns a new Series containing the counts of each unique value, sorted in descending order.

import pandas as pd

Create a Pandas Series

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

Use value_counts() to count occurrences of each unique value

counts = data.value_counts()

print(counts)

Output: 4 4 3 3 2 2 1 1 dtype: int64

In this example, value_counts() counts the occurrences of each unique value in the Series. The number 4 appears 4 times, the number 3 appears 3 times, the number 2 appears 2 times, and the number 1 appears 1 time. The result is a new Series with the counts, sorted in descending order.

Q.5 how do you handle missing values using pandas ?

Ans: To handle missing values using pandas, you can use various methods. Here's an example:

Let's say you have a DataFrame named "df" with a column called "age" that contains some missing values represented by NaN. You want to handle those missing values.

1. Drop the missing values: You can remove rows that contain missing values using the **dropna()** method. For example: df.dropna(subset=['age'], inplace=True) This will drop all rows that have missing values in the "age" column.
2. Fill missing values with a specific value: You can replace the missing values with a specific value using the **fillna()** method. For example, to fill the missing values in the "age" column with the value 0: df['age'].fillna(0, inplace=True) This will replace all missing values in the "age" column with 0.
3. Fill missing values with the mean: You can fill missing values with the mean value of the column using the **fillna()** method along with the **mean()** function. For example: mean_age = df['age'].mean() df['age'].fillna(mean_age, inplace=True) This will replace the missing values in the "age" column with the mean value calculated from the existing values.

Q.6 How do you merge 2 tables using pandas ?

Ans: To merge two tables using pandas, you can use the **merge()** function. This function combines the columns from two DataFrames based on a common key or keys.

Here's an example:

```
import pandas as pd

# Creating the first DataFrame

data1 = {'ID': [1, 2, 3, 4],

        'Name': ['John', 'Emma', 'Ryan', 'Sophia'],

        'Age': [25, 28, 30, 27]}

df1 = pd.DataFrame(data1)

# Creating the second DataFrame

data2 = {'ID': [1, 2, 3, 5],

        'Department': ['HR', 'Finance', 'Marketing', 'IT']}

df2 = pd.DataFrame(data2)

# Merging the two DataFrames based on the 'ID' column

merged_df = pd.merge(df1, df2, on='ID')

print(merged_df)
```

Output: ID Name Age Department 0 1 John 25 HR 1 2 Emma 28 Finance 2 3 Ryan 30 Marketing

In this example, we have two DataFrames (df1 and df2) with a common column 'ID'. By using the merge() function and specifying the 'ID' column as the key, we merge the two DataFrames based on their matching 'ID' values.

Q.7 How do you connect jupyter notebook to mysql ?

Ans:

Use the following syntax to connect jupyter notebook to mysql:

In Jupiter notebook write the following query:

```
!pip install mysql.connector

import mysql.connector

conn = mysql.connector.connect(host = 'localhost' ,
                    user = 'root',
                    password = '',
                    database = 'database_name')

import pandas as pd

pd.read_sql_query("SELECT * FROM  database_name" , conn)
```

Q.8 What does group by in pandas ?

Ans: The **"group by"** function in pandas is used to split a DataFrame into groups based on specified criteria, such as a column or columns. It allows you to perform operations on these grouped data such as aggregation, transformation, or filtering.

For example, let's say we have a DataFrame containing information about employees:

```
import pandas as pd

data = {'Name': ['John', 'Jane', 'Alice', 'Bob', 'John'],

        'Department': ['Finance', 'HR', 'Finance', 'IT', 'HR'],

        'Salary': [5000, 6000, 5500, 4500, 5500]}

df = pd.DataFrame(data)
```

To group the data by the "Department" column and calculate the average salary for each department, we can use the "group by" function:

```
grouped_df = df.groupby('Department')

average_salary = grouped_df['Salary'].mean()
```

The resulting "grouped_df" object is a GroupBy object that represents the grouped data. We can then apply different aggregation functions, such as "mean", "sum", "count", etc., to obtain the desired results.

In this case, we calculated the average salary for each department by specifying the "Salary" column in the "grouped_df" object.

Q.9 difference between CrossTab vs Pivot table ?

Ans: CrossTab and **Pivot table** are two different functions used for data analysis in Python, specifically in pandas library. They serve similar purposes but have slight differences in functionality.

CrossTab:

- The crosstab() function in pandas is used to compute a cross-tabulation table. It shows the frequency distribution of variables in a tabular form.
- It takes two or more categorical variables as input and computes a table that displays the relationship between them.
- The resulting table presents the frequency counts or normalized values of the variables' combinations.
- This function is typically used to analyze the relationship between two categorical variables.

Example: Let's consider a dataset of online retail sales with two categorical variables: 'Country' and 'Product_Category'. We want to analyze the frequency distribution of product categories for each country.

```
import pandas as pd

data = {

   'Country': ['USA', 'USA', 'Canada', 'Canada', 'USA', 'Canada'],

   'Product_Category': ['Electronics', 'Clothing', 'Electronics', 'Clothing', 'Electronics', 'Clothing']

}

df = pd.DataFrame(data)

crosstab_table = pd.crosstab(df['Country'], df['Product_Category'])

print(crosstab_table)
```

Output: Product_Category Clothing Electronics Country Canada 2 1 USA 1 2

Pivot Table:

- The pivot_table() function in pandas is used to create a pivot table from a given DataFrame.

- It allows us to summarize and aggregate data based on one or more variables, creating a hierarchical table.
- We can specify which variables to use as rows, columns, and values in the resulting pivot table.
- This function is used to analyze numerical or continuous variables based on categorical variables.

Example: Let's consider a dataset of employee records with variables 'Department', 'Gender', and 'Salary'. We want to calculate the average salary for each department and gender.

```
import pandas as pd

data = {
   'Department': ['HR', 'HR', 'IT', 'IT', 'Finance', 'Finance'],
   'Gender': ['Male', 'Female', 'Male', 'Female', 'Male', 'Female'],
   'Salary': [5000, 6000, 7000, 5500, 8000, 7500]
}

df = pd.DataFrame(data)


pivot_table = pd.pivot_table(df, values='Salary', index='Department', columns='Gender', aggfunc='mean')

print(pivot_table)
```

Output: Gender Female Male Department Finance 7500 8000 HR 6000 5000 IT 5500 7000

In summary, the crosstab() function is used to analyze the relationship between categorical variables, while the pivot_table() function is used to summarize numerical data based on categorical variables.

Q.10 In which scenerio what kind of plot you will use ? Numerical ==? , Categorical ==?

Ans:
For **numerical data**, you can use various types of plots such as **line plots, scatter plots, bar plots, histogram plots, box plots**, etc. These plots are useful for visualizing the distribution, trends, relationships, and comparisons within numerical data.

For **categorical data**, you can use **bar plots, pie charts, count plots, and categorical scatter plots**. These plots are helpful for displaying the frequencies, proportions, and comparisons between different categories.

Q.11 What does TimeDelta in Pandas ?

Ans:

The **TimeDelta** function in pandas is used to represent a duration or difference between two dates or times. It allows you to perform arithmetic operations on dates and times. Here's an example to illustrate its usage:

```
import pandas as pd

from datetime import datetime

# Create two datetime objects

start_time = datetime(2021, 7, 1, 10, 0, 0)

end_time = datetime(2021, 7, 1, 11, 30, 0)

# Calculate the time difference
```

duration = end_time - start_time

Create a TimeDelta object using the duration

time_delta = pd.Timedelta(duration)

Print the time difference

print(time_delta)

Output: 1:30:00

In this example, we calculate the difference between start_time and end_time using the subtraction operator. Then, we create a TimeDelta object using pd.Timedelta() and store the result in the time_delta variable. Finally, we print the time difference, which is 1 hour and 30 minutes.

Q.12 How can you extract day , month , year in a date column using pandas ?

Ans: To extract the day, month, and year from a date column using pandas, you can utilize the dt accessor provided by pandas. Here's an example:

import pandas as pd

Assuming you have a DataFrame with a date column called 'date_column'

df = pd.DataFrame({'date_column': ['2022-05-10', '2023-01-15', '2024-09-30']})

Convert the 'date_column' to datetime format

df['date_column'] = pd.to_datetime(df['date_column'])

Extract day, month, and year using dt accessor

df['day'] = df['date_column'].dt.day

df['month'] = df['date_column'].dt.month

df['year'] = df['date_column'].dt.year

Print the updated DataFrame

print(df)

Output: date_column day month year 0 2022-05-10 10 5 2022 1 2023-01-15 15 1 2023 2 2024-09-30 30 9 2024

In the above example, we convert the 'date_column' to a datetime format using pd.to_datetime(). Then, we use the dt accessor (e.g., df['date_column'].dt.day) to extract the day, month, and year components into separate columns 'day', 'month', and 'year', respectively.

Q.13 How can you extract hours , minutes , seconds in a datetime column ?

Ans:

To extract hours, minutes, and seconds from a datetime column in Python using pandas, you can use the dt accessor provided by pandas. Here's an example:

import pandas as pd

Create a DataFrame with a datetime column

data = {'date_time': ['2022-01-01 09:30:45', '2022-01-01 10:45:20', '2022-01-01 14:15:30']}

df = pd.DataFrame(data)

Convert the 'date_time' column to datetime type

df['date_time'] = pd.to_datetime(df['date_time'])

Extract hours, minutes, and seconds using the dt accessor

df['hours'] = df['date_time'].dt.hour

df['minutes'] = df['date_time'].dt.minute

df['seconds'] = df['date_time'].dt.second

print(df)

Output: date_time hours minutes seconds 0 2022-01-01 09:30:45 9 30 45 1 2022-01-01 10:45:20 10 45 20 2 2022-01-01 14:15:30 14 15 30

In the example above, we first convert the 'date_time' column to datetime type using pd.to_datetime(). Then, we use the dt accessor to extract the hours, minutes, and seconds into separate columns 'hours', 'minutes', and 'seconds', respectively.

Q.14 How do you read a excel file , csv file and json file in jupyter notebook using pandas ?

Ans:

1. Reading an **Excel** file:

   import pandas as pd

   df_excel = pd.read_excel('path_to_file.xlsx')

2. Reading a **CSV** file:

   import pandas as pd

   df_csv = pd.read_csv('path_to_file.csv')

3. Reading a **JSON** file:

   import pandas as pd

   df_json = pd.read_json('path_to_file.json')

Q.15 How can you create excel data from a dataframe using pandas ?

Ans:

To create an Excel file from a DataFrame using pandas, you can use the to_excel() function. Here's an example:

import pandas as pd

Create a DataFrame

data = {'Name': ['John', 'Emily', 'Michael'],

'Age': [25, 28, 32],

'City': ['New York', 'London', 'Paris']}

df = pd.DataFrame(data)

Save the DataFrame to an Excel file

```
df.to_excel('data.xlsx', index=False)
```

In this example, we first create a DataFrame df from a dictionary data. Then, we use the to_excel() function to save the DataFrame to an Excel file named 'data.xlsx'. The index=False parameter ensures that the index column is not included in the Excel file.

After executing the code, you will find an Excel file named "data.xlsx" in your current working directory containing the DataFrame's data.