PYTHON FOR DATA SCIENCE

ASSIGNMENT – 1

MCQ Questions

1. Create a list called “Stationery” with the below data

Product = ['Pencil', 'Pen', 'Eraser', 'Pencil Box', 'Scale']

Price= [5, 10, 2, 20, 12]

Brand = ['Camlin', 'Rotomac', 'Nataraj', 'Camel', 'Apsara']

Stationery = [Product, Price, Brand]

The command to add “Notebook” as the first element inside the first level of the list “Stationery” is:-

a) Stationery[0].append('Notebook')
b) Stationery[0].insert(0,'Notebook')
c) Stationery[0][1] = "Notebook"
d) Stationery[0].extend('Notebook')

Ans : b) Stationery[0].insert(0,'Notebook')

2. The function used to perform k-Nearest Neighbors classification is: -
   a) sklearn.KNN
   b) sklearn.KNearestClassifier
   c) sklearn.neighbors.KNeighborsClassifier( )
   d) sklearn.neighbors.KNeighborsRegressor( )

Ans : c) sklearn.neighbors.KNeighborsClassifier( )

Descriptive Answers

3. Explain the process and best practices for identifying and imputing missing values in a Pandas DataFrame, including the differences in handling numerical and categorical variables.

Answer:

In Pandas, missing values are typically represented by NaN (*Not a Number*). To identify missing values, the isna() or isnull() functions are used, returning a DataFrame of Boolean values (True for missing entries). To count missing values per column, use DataFrame.isnull().sum(). For example, cars_data2.isnull().sum() gives the count of missing values in each column.

To subset rows with missing values, use missing = cars_data2[cars_data2.isnull().any(axis=1)], which returns only those rows that have one or more missing values.

When imputing (filling) missing values, best practices differ based on the variable type:

- For numerical variables, the missing values are usually filled using either the mean or the median, depending on the data's distribution. For example, to impute the mean for the 'Age' column:

python

cars_data2['Age'].fillna(cars_data2['Age'].mean(), inplace=True)

For the 'KM' column, if the data is skewed, the median is preferable:

python

cars_data2['KM'].fillna(cars_data2['KM'].median(), inplace=True)

- For categorical variables, the missing values are typically filled with the mode (the most frequent value), determined with value_counts().index:

python

cars_data2['FuelType'].fillna(cars_data2['FuelType'].value_counts().index[0], inplace=True)

Similarly, for other categorical columns, mode() can also be used.

After imputation, check for any remaining missing values using DataFrame.isnull().sum() to ensure all have been handled.

For convenience and automation, a lambda function can impute all columns at once, using the mean for numerical columns and the mode for categorical ones:

python

cars_data3 = cars_data3.apply(**lambda** x: x.fillna(x.mean()) **if** x.dtype=='float' **else** x.fillna(x.value_counts().index[0]))

4. What are the common sequence data operations in Python, and how are they applied differently to various sequence types like lists, arrays, dictionaries, and sets?

Answer:

Common Sequence Operations in Python

Python provides several fundamental operations for sequence data types—such as indexing, slicing, concatenation, multiplication, insertion, removal, and clearing—which are applied with some variations depending on the sequence type.

Indexing and Slicing

- Indexing accesses an element by its position. Supported by ordered sequences like strings, lists, tuples, arrays, and ranges. For example, list[index] retrieves the element at a given index.
- Slicing extracts a subset of elements using the slice(start, stop, step) syntax or shorthand list[start:stop:step]. Works on strings, lists, tuples, arrays, and ranges.
- Sets and Dictionaries do not support direct indexing or slicing because sets are unordered and dictionaries are key-value mappings.

Concatenation and Multiplication

- Concatenation using + or += combines sequences of the same type, for instance, joining two lists or tuples.
- Multiplication using * repeats the sequence elements. For example, list * 2 doubles the list contents.
- These operations are not valid on sets or dictionaries, and arrays require concatenation with another array of the same type.

Insertion and Removal

- Insertion: Lists and arrays support methods like .insert(index, element) to add elements at specific positions.
- Removal: Methods like .pop(index), .remove(value) remove elements. .pop() without an index removes the last item.
- Dictionaries remove key-value pairs using .pop(key).
- Sets use .add(value), .remove(value), and .discard(value) for manipulation (no indexing).

Clearing and Deletion

- .clear() empties lists, dictionaries, or sets entirely.
- The del keyword removes entire objects or specific elements/slices from lists or dictionaries but not directly applicable to sets.