DABBEL®

Feel free to go above and beyond to show off the extent of your skills when answering the following questions! Where necessary, use code, comments, charts, etc. to answer the questions.

Also, set a random seed to guarantee reproducibility of results where applicable.

You may use a Python Notebook or plain Python code. Please add detailed instructions on how to run and set up the code in an isolated environment (docker container, virtualenv, Anaconda, etc.) on a Unix-based System (Linux, Mac, WSL, etc.). Somebody with a factory fresh computer should be able to run it.

Question 1 :

The dataset for this question can be found here and the details of the columns are as follows :

1. Age : The age of the primary beneficiary.
2. Sex : Gender of the insurance contractor.
3. BMI : Body-mass index
4. Children : Number of children covered by health insurance.
5. Smoker : Specifies whether the insured person smokes or not.
6. Region : The beneficiary's region of residence.
7. Charges : Individual medical costs billed by health insurance.

1. Apply all the necessary data preprocessing techniques (including insightful visualizations) to produce a dataset that can readily be used to train a regression model. Pay special attention to how the sex, smoker, and regions columns are handled in your preprocessing pipeline.
2. Build and train a regression model that can be used to predict Charges for prospective insurance beneficiaries.
3. With appropriate visualizations, justify your choice of columns/features for training the regression model.
4. In code, demonstrate how to measure the performance of your regression model.

Question 2:

Implement a function that calculates and returns a measure of the similarity between 2 lists of items. The output of the function should be bounded between 0 and 1 such that if 1 is returned, it means the 2 lists are equal and if 0 is returned, the 2 lists are completely different.

Make up 3 similarity measures:

- One that does an item-wise comparison and takes order of items into account
- Same as above, but without taking order into account

- One that looks at the lists globally
- The measures cannot be equality (i.e., item 1 == item 2) or length (i.e., len(item 1) == len(item 2))
- The lists do not need to have the same length and the measure should reflect this
- Your implementation must be case-insensitive

Evaluate your implementation on the following inputs and print the calculated similarities.

1. ['A','b','c','d'] and ['e', 'F', 'G', 'h']
2. ['a','B','c','d'] and ['d', 'C', 'A', 'H']
3. ['A','B','C','D'] and ['a', 'b', 'c', 'd']
4. ['banana','Orange','pear','Avocado'] and ['mango', 'strawberry', 'apple', 'orange']
5. ['D','a','B','b', 'e', 'L'] and ['B', 'B', 'l', 'e', 'a', 'd']
6. ['A','B','E','D'] and ['a', 'b', 'c']

Question 3

data = [{ "sensor_name": "temp_001", "status": "active", "value": 20.5 }, { "sensor_name": "hum_010", "status": "inactive", "value": 80 }, { "sensor_name": "Temp_002", "status": "inactive", "value": 18.0 }, { "sensor_name": "TEMP_005", "status": "active", "value": 23.0 }, { "sensor_name": "hum_011", "status": "active", "value": 65 }, { "sensor_name": "tEmP_009", "status": "active", "value": -12.0 }, { "sensor_name": "tEmP_029", "status": "inactive", "value": 15.5 }]

Using the dataset described above, write code in Python to achieve the following use cases :

1. Implement a function get_temperature_values(dataset :Dict) ->List that returns a list of values from all active temperature sensors. Note : Names of temperature sensors ALWAYS contain "temp" regardless of whether the characters are uppercase or lowercase.
2. Write a Python expression/statement to print the number of inactive temperature sensors in the dataset.
3. Implement a function average_temperature_values that returns the average temperature value using the function get_temperature_values implemented in step 1.