Ineuron

Assignment:-3

1.What is the process for loading a dataset from an external source?
Ans:-If you use no data from STATMON as the source for your IBM Z Performance and Capacity Analytics resource information, you can import data from another external source into IBM Z Performance and Capacity Analytics. The external source serves as the primary repository for network resource information, so you must not change the resource information after you update the NW_RESOURCE table with the information. Any changes you make to the resource information using the network administration dialog will not be present in the external source, and therefore will not reflect the actual state of the network. The data must be in the format presented in “External data file format” for the dialog to be able to use it.

Load data from the external source when you are first defining your network to the Network Performance Feature and whenever the network changes.

The following figure illustrates the process of loading data from an external source in the IBM Z Performance and Capacity Analytics database.

Figure 1. Loading data from an external source
Figure 69. Loading data from an external source
If you use external data to create your database, follow these steps:

Set the defaults for network resources in the network administration dialog. You must specify the name of a data set that you have already allocated in the Output data set name field. The dialog requires that this value be present so you can update the database. Because the dialog does not permit you to enter only the value for that field, you must enter values for all fields, even though the dialog does not use the information.
Load the external data into the dialog work table.
Verify the resource information.
Process the database update statements to load the data in the work table into the IBM Z Performance and Capacity Analytics database.
The following figure provides an overview of the windows involved in this process.
Figure 2. Window flow: Loading external data
Figure 70. Window flow: Loading external data

2.How can we use pandas to read JSON files?
Ans:-Introduction:-Pandas is one of the most commonly used Python libraries for data handling and visualization. The Pandas library provides classes and functionalities that can be used to efficiently read, manipulate and visualize data, stored in a variety of file formats.
In this article, we'll be reading and writing JSON files using Python and Pandas.JavaScript Object Notation (JSON) is a data format that stores data in a human-readable form. While it can be technically be used for storage, JSON files are primarily used for serialization and information exchange between a client and server.
Although it was derived from JavaScript, it's platform-agnostic and is a widely-spread and used format - most prevalently in REST APIs.
Creating a JSON File

To create JSON files via Python, data has to be stored in a certain way. There are multiple ways of storing this data using Python. Some of the methods have been discussed in this article. We'll first create a file using core Python and then read and write to it via Pandas.

Creating JSON Data via a Nested Dictionaries

In Python, to create JSON data, you can use nested dictionaries. Each item inside the outer dictionary corresponds to a column in the JSON file.

The key of each item is the column header and the value is another dictionary consisting of rows in that particular column. Let's create a dictionary that can be used to create a JSON file that stores a record of fictional patients:

```
patients = {
        "Name":{"0":"John","1":"Nick","2":"Ali","3":"Joseph"},
                "Gender":{"0":"Male","1":"Male","2":"Female","3":"Male"},
                        "Nationality":{"0":"UK","1":"French","2":"USA","3":"Brazil"},
                                "Age" :{"0":10,"1":25,"2":35,"3":29}
                                }
```

In the script above, the first item corresponds to the Name column. The item value consists of a dictionary where dictionary items represent rows. The keys of the inner dictionary items corresponds to the index numbers of rows, where values represent row values. Since that might be a bit hard to visualize just like that, here's a visual representation:Reading JSON Files with Pandas

To read a JSON file via Pandas, we'll utilize the read_json() method and pass it the path to the file we'd like to read. The method returns a Pandas DataFrame that stores data in the form of columns and rows.

Though, first, we'll have to install Pandas:

```
$ pip install pandas
```

Reading JSON from Local Files

The following script reads the patients.json file from a local system directory and stores the result in the patients_df dataframe. The header of the dataframe is then printed via the head() method:

```
import pandas as pd
patients_df = pd.read_json('E:/datasets/patients.json')
patients_df.head()
```

patients visualized

In the Name column, the first record is stored at the 0th index where the value of the record is John, similarly, the value stored at the second row of the Name column is Nick and so on.

Creating JSON Data via Lists of Dictionaries

Another way to create JSON data is via a list of dictionaries. Each item in the list consists of a dictionary and each dictionary represents a row. This approach is a lot more readable than using nested dictionaries.

Let's create a list that can be used to create a JSON file that stores information about different cars:

```
cars = [
   {"Name":"Honda", "Price": 10000, "Model":2005, "Power": 1300},
      {"Name":"Toyota", "Price": 12000, "Model":2010, "Power": 1600},
         {"Name":"Audi", "Price": 25000, "Model":2017, "Power": 1800},
```

```
        {"Name":"Ford", "Price": 28000, "Model":2009, "Power": 1200},
]
```

3.Describe the significance of DASK?
Ans:-Dask-
Dask is a flexible library for parallel computing in Python.
Dask is composed of two parts:

Dynamic task scheduling optimized for computation. This is similar to Airflow, Luigi, Celery, or Make, but optimized for interactive computational workloads.
“Big Data” collections like parallel arrays, dataframes, and lists that extend common interfaces like NumPy, Pandas, or Python iterators to larger-than-memory or distributed environments. These parallel collections run on top of dynamic task schedulers.
Dask emphasizes the following virtues:
Familiar: Provides parallelized NumPy array and Pandas DataFrame objects
Flexible: Provides a task scheduling interface for more custom workloads and integration with other projects.
Native: Enables distributed computing in pure Python with access to the PyData stack.
Fast: Operates with low overhead, low latency, and minimal serialization necessary for fast numerical algorithms

Scales up: Runs resiliently on clusters with 1000s of cores
Scales down: Trivial to set up and run on a laptop in a single process
Responsive: Designed with interactive computing in mind, it provides rapid feedback and diagnostics to aid humans
Dask is composed of three parts. "Collections" create "Task Graphs" which are then sent to the "Scheduler" for execution. There are two types of schedulers that are described in more detail below.
High level collections are used to generate task graphs which can be executed by schedulers on a single machine or a cluster.Familiar user interface
Dask DataFrame mimics Pandas - documentation

```
import pandas as pd                    import dask.dataframe as dd
df = pd.read_csv('2015-01-01.csv')     df = dd.read_csv('2015-*-*.csv')
df.groupby(df.user_id).value.mean()    df.groupby(df.user_id).value.mean().compute()
```
Dask Array mimics NumPy - documentation

```
import numpy as np                     import dask.array as da
f = h5py.File('myfile.hdf5')           f = h5py.File('myfile.hdf5')
x = np.array(f['/small-data'])         x = da.from_array(f['/big-data'],
                                           chunks=(1000, 1000))
                                           x - x.mean(axis=1)                  x -
x.mean(axis=1).compute()
```
Dask Bag mimics iterators, Toolz, and PySpark - documentation

```
import dask.bag as db
b = db.read_text('2015-*-*.json.gz').map(json.loads)
b.pluck('name').frequencies().topk(10, lambda pair: pair[1]).compute()
```

Dask Delayed mimics for loops and wraps custom code - documentation

```
from dask import delayed
L = []
for fn in filenames:                # Use for loops to build up computation
   data = delayed(load)(fn)         # Delay execution of function
      L.append(delayed(process)(data))  # Build connections between variables
result = delayed(summarize)(L)
result.compute()
```

The concurrent.futures interface provides general submission of custom tasks: - documentation

```
from dask.distributed import Client
client = Client('scheduler:port')

futures = []
for fn in filenames:
   future = client.submit(load, fn)
      futures.append(future)
summary = client.submit(summarize, futures)
summary.result()
```

4.Describe the functions of DASK?
Ans:-
Top level functions
abs(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.absolute.
absolute(x, /[, out, where, casting, order, ...])

This docstring was copied from numpy.absolute.
add(x1, x2, /[, out, where, casting, order, ...])

This docstring was copied from numpy.add.
all(a[, axis, keepdims, split_every, out])

Test whether all array elements along a given axis evaluate to True.
allclose(arr1, arr2[, rtol, atol, equal_nan])
Returns True if two arrays are element-wise equal within a tolerance.
angle(x[, deg])

Return the angle of the complex argument.
any(a[, axis, keepdims, split_every, out])

Test whether any array element along a given axis evaluates to True.
append(arr, values[, axis])
Append values to the end of an array.
apply_along_axis(func1d, axis, arr, *args[, ...])
Apply a function to 1-D slices along the given axis.
apply_over_axes(func, a, axes)

Apply a function repeatedly over multiple axes.
arange(*args[, chunks, like, dtype])

Return evenly spaced values from start to stop with step size step.
arccos(x, /[, out, where, casting, order, ...])

This docstring was copied from numpy.arccos.
arccosh(x, /[, out, where, casting, order, ...])

This docstring was copied from numpy.arccosh.
arcsin(x, /[, out, where, casting, order, ...])

This docstring was copied from numpy.arcsin.
arcsinh(x, /[, out, where, casting, order, ...])

This docstring was copied from numpy.arcsinh.
arctan(x, /[, out, where, casting, order, ...])

This docstring was copied from numpy.arctan.
arctan2(x1, x2, /[, out, where, casting, ...])

This docstring was copied from numpy.arctan2.
arctanh(x, /[, out, where, casting, order, ...])

This docstring was copied from numpy.arctanh.
argmax(a[, axis, keepdims, split_every, out])

Returns the indices of the maximum values along an axis.
argmin(a[, axis, keepdims, split_every, out])

Returns the indices of the minimum values along an axis.
argtopk(a, k[, axis, split_every])

Extract the indices of the k largest elements from a on the given axis, and return them sorted from largest to smallest.
argwhere(a)

Find the indices of array elements that are non-zero, grouped by element.
around(x[, decimals])

Evenly round to the given number of decimals.
array(object[, dtype, copy, order, subok, ...])

This docstring was copied from numpy.array.

asanyarray(a[, dtype, order, like, inline_array])
Convert the input to a dask array.

asarray(a[, allow_unknown_chunksizes, ...])

Convert the input to a dask array.
atleast_1d(*arys)

Convert inputs to arrays with at least one dimension.
atleast_2d(*arys)

View inputs as arrays with at least two dimensions.
atleast_3d(*arys)

View inputs as arrays with at least three dimensions.
average(a[, axis, weights, returned])

Compute the weighted average along the specified axis.
bincount(x, /[, weights, minlength])

This docstring was copied from numpy.bincount.
bitwise_and(x1, x2, /[, out, where, ...])

This docstring was copied from numpy.bitwise_and.
bitwise_not(x, /[, out, where, casting, ...])

This docstring was copied from numpy.invert.
bitwise_or(x1, x2, /[, out, where, casting, ...])

This docstring was copied from numpy.bitwise_or.
bitwise_xor(x1, x2, /[, out, where, ...])

This docstring was copied from numpy.bitwise_xor.
block(arrays[, allow_unknown_chunksizes])
Assemble an nd-array from nested lists of blocks.
blockwise(func, out_ind, *args[, name, ...])
Tensor operation: Generalized inner and outer products
broadcast_arrays(*args[, subok])

Broadcast any number of arrays against each other.
broadcast_to(x, shape[, chunks, meta])
Broadcast an array to a new shape.
cbrt(x, /[, out, where, casting, order, ...])

This docstring was copied from numpy.cbrt.
coarsen(reduction, x, axes[, trim_excess])
Coarsen array by applying reduction to fixed size neighborhoods
ceil(x, /[, out, where, casting, order, ...])

This docstring was copied from numpy.ceil.
choose(a, choices)

Construct an array from an index array and a list of arrays to choose from.
clip(*args, **kwargs)

Clip (limit) the values in an array.
compress(condition, a[, axis])
Return selected slices of an array along given axis.
concatenate(seq[, axis, ...])

Concatenate arrays along an existing axis
conj(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.conjugate.
copysign(x1, x2, /[, out, where, casting, ...])

This docstring was copied from numpy.copysign.
corrcoef(x[, y, rowvar])

Return Pearson product-moment correlation coefficients.
cos(x, /[, out, where, casting, order, ...])

This docstring was copied from numpy.cos.
cosh(x, /[, out, where, casting, order, ...])

This docstring was copied from numpy.cosh.
count_nonzero(a[, axis])

Counts the number of non-zero values in the array a.
cov(m[, y, rowvar, bias, ddof])
Estimate a covariance matrix, given data and weights.
cumprod(x[, axis, dtype, out, method])

Return the cumulative product of elements along a given axis.
cumsum(x[, axis, dtype, out, method])

Return the cumulative sum of the elements along a given axis.
deg2rad(x, /[, out, where, casting, order, ...])

This docstring was copied from numpy.deg2rad.
degrees(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.degrees.
diag(v[, k])

Extract a diagonal or construct a diagonal array.
diagonal(a[, offset, axis1, axis2])

Return specified diagonals.
diff(a[, n, axis, prepend, append])

Calculate the n-th discrete difference along the given axis.
divmod(x1, x2[, out1, out2], / [[, out, ...])
This docstring was copied from numpy.divmod.
digitize(a, bins[, right])

Return the indices of the bins to which each value in input array belongs.
dot(a, b[, out])

This docstring was copied from numpy.dot.
dstack(tup[, allow_unknown_chunksizes])
Stack arrays in sequence depth wise (along third axis).
ediff1d(ary[, to_end, to_begin])
The differences between consecutive elements of an array.

einsum(subscripts, *operands[, out, dtype, ...])

This docstring was copied from numpy.einsum.
empty(*args, **kwargs)
Blocked variant of empty_like
empty_like(a[, dtype, order, chunks, name, ...])
Return a new array with the same shape and type as a given array.
equal(x1, x2, /[, out, where, casting, ...])
This docstring was copied from numpy.equal.
exp(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.exp.
exp2(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.exp2.
expm1(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.expm1.
eye(N[, chunks, M, k, dtype])
Return a 2-D Array with ones on the diagonal and zeros elsewhere.
fabs(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.fabs.
fix(*args, **kwargs)
Round to nearest integer towards zero.
flatnonzero(a)
Return indices that are non-zero in the flattened version of a.
flip(m[, axis])
Reverse element order along axis.
flipud(m)
Reverse the order of elements along axis 0 (up/down).
fliplr(m)
Reverse the order of elements along axis 1 (left/right).
float_power(x1, x2, /[, out, where, ...])
This docstring was copied from numpy.float_power.
floor(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.floor.
floor_divide(x1, x2, /[, out, where, ...])

This docstring was copied from numpy.floor_divide.
fmax(x1, x2, /[, out, where, casting, ...])
This docstring was copied from numpy.fmax.
fmin(x1, x2, /[, out, where, casting, ...])
This docstring was copied from numpy.fmin.
fmod(x1, x2, /[, out, where, casting, ...])
This docstring was copied from numpy.fmod.
frexp(x[, out1, out2], / [[, out, where, ...])
This docstring was copied from numpy.frexp.
fromfunction(func[, chunks, shape, dtype])
Construct an array by executing a function over each coordinate.
frompyfunc(func, /, nin, nout, *[, identity])
This docstring was copied from numpy.frompyfunc.
full(shape, fill_value, *args, **kwargs)
Blocked variant of full_like
full_like(a, fill_value[, order, dtype, ...])
Return a full array with the same shape and type as a given array.
gradient(f, *varargs[, axis])
Return the gradient of an N-dimensional array.
greater(x1, x2, /[, out, where, casting, ...])
This docstring was copied from numpy.greater.
greater_equal(x1, x2, /[, out, where, ...])
This docstring was copied from numpy.greater_equal.
histogram(a[, bins, range, normed, weights, ...])
Blocked variant of numpy.histogram().
histogram2d(x, y[, bins, range, normed, ...])
Blocked variant of numpy.histogram2d().
histogramdd(sample, bins[, range, normed, ...])
Blocked variant of numpy.histogramdd().
hstack(tup[, allow_unknown_chunksizes])
Stack arrays in sequence horizontally (column wise).
hypot(x1, x2, /[, out, where, casting, ...])
This docstring was copied from numpy.hypot.
imag(*args, **kwargs)
Return the imaginary part of the complex argument.
indices(dimensions[, dtype, chunks])
Implements NumPy's indices for Dask Arrays.
insert(arr, obj, values, axis)
Insert values along the given axis before the given indices.
invert(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.invert.
isclose(arr1, arr2[, rtol, atol, equal_nan])
Returns a boolean array where two arrays are element-wise equal within a tolerance.
iscomplex(*args, **kwargs)
Returns a bool array, where True if input element is complex.
isfinite(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.isfinite.
isin(element, test_elements[, ...])

Calculates element in test_elements, broadcasting over element only.
isinf(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.isinf.
isneginf
This docstring was copied from numpy.equal.
isnan(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.isnan.
isnull(values)
pandas.isnull for dask arrays
isposinf
This docstring was copied from numpy.equal.
isreal(*args, **kwargs)
Returns a bool array, where True if input element is real.
ldexp(x1, x2, /[, out, where, casting, ...])
This docstring was copied from numpy.ldexp.
left_shift(x1, x2, /[, out, where, casting, ...])
This docstring was copied from numpy.left_shift.
less(x1, x2, /[, out, where, casting, ...])
This docstring was copied from numpy.less.
linspace(start, stop[, num, endpoint, ...])
Return num evenly spaced values over the closed interval [start, stop].
log(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.log.
log10(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.log10.
log1p(x, /[, out, where, casting, order, ...])
This docstring was copied from numpy.log1p.
log2(x, /[, out, where, casting, order, ...])

5.Describe Cassandra features?
Ans:-Features of Cassandra
Apache Cassandra is an open source, user-available, distributed, NoSQL DBMS which is designed to handle large amounts of data across many servers.