**Ineuron assignment 3**

Question-1: what is the process for loading a dataset from an external source?

Ans: When you load data from an external source, you load it into a suspense table. You can then review the data in the suspense table and modify it. To load data into the suspense table, position the source file or tape, specify the location of the source, and run the appropriate load external data proces.

Data loading is the process of copying and loading data or data sets from a source file, folder or application to a database or similar application. It is usually implemented by copying digital data from a source and pasting or loading the data to a data storage or processing utility.

Data loading is used in database-based extraction and loading techniques. Typically, such data is loaded into the destination application as a different format than the original source location.

For example, when data is copied from a word processing file to a database application, the data format is changed from .doc or .txt to a .CSV or DAT format. Usually, this process is performed through or the last phase of the Extract, Transform and Load (ETL) process. The data is extracted from an external source and transformed into the destination application's supported format, where the data is further loaded.

Question-2: How can we use pandas to read  JSON files?

Ans: Reading JSON Files using Pandas

To read the files, we use read\_json() function and through it, we pass the path to the JSON file we want to read. Once we do that, it returns a “DataFrame”( A table of rows and columns) that stores data. If we want to read a file that is located on remote servers then we pass the link to its location instead of a local path.

JSON is shorthand for JavaScript Object Notation which is the most used file format that is used to exchange data between two systems or web applications. When we are working with files in big data or machine learning we are often required to process JSON files.

1.. pandas read\_json() Syntax.

#Syntax of read\_json()

pandas.read\_json(path\_or\_buf=None, orient=None, typ='frame', dtype=None, convert\_axes=None, convert\_dates=True, keep\_default\_dates=True, numpy=False, precise\_float=False, date\_unit=None, encoding=None, encoding\_errors='strict', lines=False, chunksize=None, compression='infer', nrows=None, storage\_options=None)

2. Read JSON String.

# Import pandas

import pandas as pd

# Read json from String

json\_str = '{"Courses":{"r1":"Spark"},"Fee":{"r1":"25000"},"Duration":{"r1":"50 Days"}}'

df = pd.read\_json(json\_str)

print(df)

# Outputs

#   Courses    Fee Duration

#r1   Spark  25000  50 Days

3. Pandas Read JSON File .

# pandas read JSON File

df = pd.read\_json('courses\_data.json')

print(df)

# Outputs

#  Courses    Fee Duration

#0   Spark  25000  50 Days

#1  Pandas  20000  35 Days

#2    Java  15000

4. Read N Records from JSON File.

# Read JSON file with records orient

df = pd.read\_json('courses.json', orient='records', nrows=2, lines=True)

print(df)

5. Compression & Encoding.

Use compression param to uncompress and load JSON files from {'zip', 'gzip', 'bz2', 'zstd'}.

When using ‘zip’, make sure the ZIP file contains only one data file. Use None value to specify no decompression.

Use encoding param to support custom encoding, by default it uses UTF-8 encoding.

6. Other Params to Read JSON.

dtype – Specify a dict of column to dtype. When True, infer the dtype based on data. If False, then don’t infer dtypes.

convert\_axes – Convert the axes to the proper dtypes.

convert\_dates – If True then all date like columns will be converted to date. If False it doesn’t convert.

keep\_default\_dates – If True, based on column labels it converts the datelike columns.

Question-3: Describe the significance of DASK?

Ans:  Dask is a free and open-source library for parallel computing in Python. Dask helps you scale your data science and machine learning workflows. Dask makes it easy to work with Numpy, pandas, and Scikit-Learn, but that’s just the beginning. Dask is a framework to build distributed applications that has since been used with dozens of other systems like XGBoost, PyTorch, Prefect, Airflow, RAPIDS, and more. It’s a full distributed computing toolbox that fits comfortably in your hand.

If you have larger-than-memory data, you can use Dask to scale up your workflow to leverage all the cores of your local workstation, or even scale out to the cloud.

Behind-the-scenes workings of Dask with schedulers and workers, and

Dask’s diagnostics dashboard and resources for scaling to the cloud.

Brief history of dask:

One pain point we heard time and time again was that people worked with data that fit comfortably on disk but that was too big for RAM, and accelerating NumPy was a common feature request to Continuum/Anaconda at the time. To this end, the purpose of Dask was originally to parallelize NumPy so that it could harness one full workstation computer, which was common in finance shops at the time. There were two technical goals, and a social goal:

Technical: Harness the power of all of the cores on the laptop/workstation in parallel;

Technical: Support larger-than-memory computation, allowing datasets that fit on disk, but not in RAM;

Social: Invent nothing. We wanted to be as familiar as possible to what users already knew in the PyData stack.”

Question-4: Describe the functions of DASK?

Ans:Dask is a flexible library for parallel computing in Python.

Dask is composed of two parts:

1.Dynamic task scheduling optimized for computation. This is similar to Airflow, Luigi, Celery, or Make, but optimized for interactive computational workloads.

2.“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.

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.

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().

Question-5:Describe cassandra's features.

Ans:Apache Cassandra is an open source, user-available, distributed, NoSQL DBMS which is designed to handle large amounts of data across many servers. It provides zero point of failure. Cassandra offers massive support for clusters spanning multiple datacentres.

There are some massive features of Cassandra. Here are some of the features described below:

Distributed:

Each node in the cluster  has same role. There’s no question of failure & the data set is distributed across the cluster but one issue is there that is the master isn’t present in each node to support request for service.

Supports replication & Multi data center replication:

Replication factor comes with best configurations in cassandra. Cassandra is designed to have a distributed system, for the deployment of large number of nodes for across multiple data centers and other key features too.

Scalability:

It is designed to r/w throughput, Increase gradually as new machines are added without interrupting other applications.

Fault-tolerance:

Data is automatically stored & replicated for fault-tolerance. If a node Fails, then it is replaced within no time.

MapReduce Support:

It supports Hadoop integration with MapReduce support.Apache Hive & Apache Pig is also supported.

Query Language:

Cassandra has introduced the CQL (Cassandra Query Language). Its a simple interface for accessing the Cassandra.