Objectives:

1. Load the data file provided in a local instance of MongoDB
2. Extract all data from the ‘sensor\_data’ collection to form the data for ML task.
3. Create a regression-based ML/DL model, to predict the 'Pump Radial Bearing Vibration' column from the data extracted.

* Share your understanding of underlying data using descriptive analytics (You’re encouraged to do data cleaning, EDA et al.)
* Use of proper feature selection process and hyperparameter tuning (if required) is expected.
* You’re allowed to do any feature engineering/feature selection you deem necessary
* Properly comment on the code as and where required

1. Create a script which will load the model and make predictions for data provided to it for testing purposes (sample script attached)

You can send your solution (scripts, model file and any notebooks you used for model creation/EDA) as either a zip file or as a part of a public github repo.

Please feel free to use any and all python packages which are publicly available for the above task, these include: scikit-learn, keras, tensorflow, pytorch, xgboost, catboost, etc. Please try and mention the version of the package used as well for a smooth experience on our side while testing your solutions (There can be breaking changes between versions of the packages).

We will not entertain solutions which use propriety packages such as H2O.

**We are providing you with a few suggestions/directives below on how to approach the problem statement if you have never worked with MongoDB, on how to tackle the data structure, as well as how to create the prediction script.**

**Please also look at FAQs at the end to see how to tackle some simple problems which you may run into**

Setting Up MongoDB locally:

* The **easiest** way to set up MongoDB locally is to follow this guide: <https://www.prisma.io/dataguide/mongodb/setting-up-a-local-mongodb-database>

You’ll also need to install the developer tools to interact with the Db, as mentioned here:  
<https://www.mongodb.com/docs/database-tools/installation/installation/#installing-the-database-tools>

This should have you up and running in less than 10 mins.

* If you do not want to install MongoDB locally, then you can use docker to set up a local instance too (This is only advised for people who are familiar with docker as it is **another layer of complexity** on the above task and is **totally not necessary**). Here’s a link to a tutorial about how to do this: <https://www.linode.com/docs/guides/set-up-mongodb-on-docker/>

You may also have to attach the container with a volume to keep the data persistent.

* Once mongdb is installed, you can load the data dump (First un-compress the provided file) provided in a database using:

***mongorestore --verbose <\path-to-data-dump>***

More details if interested: <https://fedingo.com/how-to-restore-mongodb-dump-in-windows-linux/>

Querying MongoDB:

If anyone has worked with any databases, it should not be too difficult for them to pick up how to work with MongoDB. This is especially true because we only require you to extract data from the database to use it, which should be a very simple query. Nonetheless, here are some resources which might help you with the query as well as the underlying resources required to query the data:

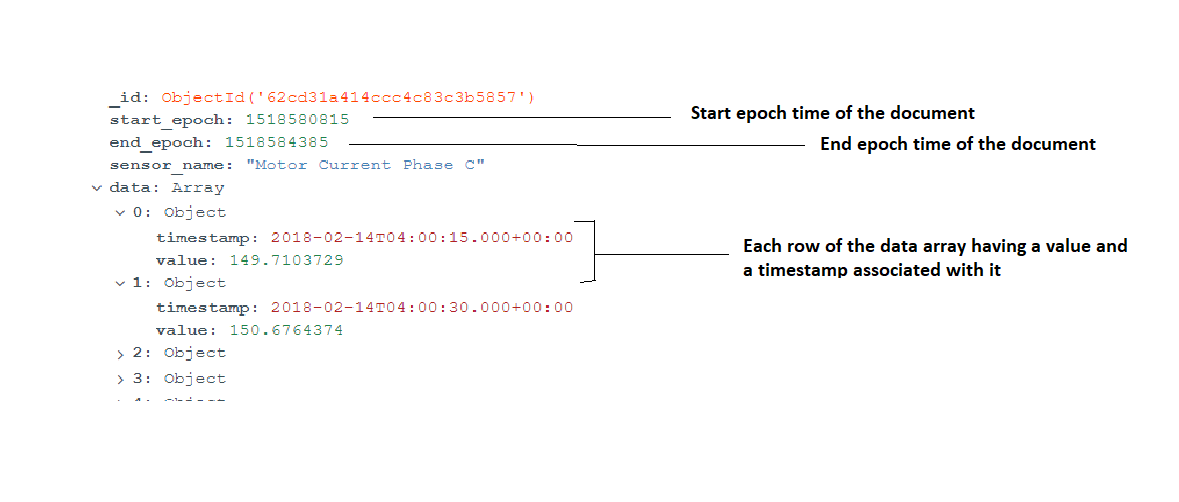
<https://www.analyticsvidhya.com/blog/2020/08/query-a-mongodb-database-using-pymongo/>

<https://www.w3schools.com/python/python_mongodb_query.asp>

Data Structure:

The data is available in the database in a strict structure in the **sensor\_data** collection.

Each document in the collection corresponds to data contained for a specific sensor, with additional information, such as the start and end epoch time of the data present in that document being available. The data field for each document is an array, with each row having a particular timestamp and the value for the sensor incorporated in it.



The database has data from 1 January 2018 0:00:15 (start\_epoch: 1514764815) to 31 December 2019 23:59:45 (end\_epoch: 1577836785).

**Once queried, transform the data into usable format for a regression-based ML task and then go ahead with the model creation.**

Prediction Script Example:

Let's assume that there are 4 features available to us to use: datetime, A, B, C, D and we have to predict E, so sample input to the script will be like (assume Json):

[

{"datetime": "2020-01-01 00:00:00+00:00", "A": 1, "B": 2, "C": 4, "D": 5

},

{"datetime": "2020-01-01 00:00:15+00:00", "A": 3, "B": 5, "C": 2, "D": 5

},

{"datetime": "2020-01-01 00:00:30+00:00", "A": 2, "B": 5, "C": 1, "D": 5

}

]

Now we create a new feature Z as, Z = A+B and then use Z, C and D for our model and the model is saved as model.pkl (pickle file used for illustration here), then the script would have a function like:

import pandas as pd

import pickle

def predict (data, model\_path):

'''

data: data to get prediction on (JSON)

model\_path: path to the model (string)

'''

model = pickle.load(open(model\_path, 'rb'))

data = pd.read\_json(data)

data['Z'] = data['A'] + data['B']

prediction = model.predict(data[['Z','A','B']])

return prediction

Now we expect the same type of function from your end for prediction, if there are any new features created, any encoding done, any data transformation, they should all be incorporated in the script so that when we call the function with the **data** (JSON data) and the **model\_path** then it automatically handles every conversion internally and gives us out the predictions as expected.

**Note**: If you have an encoder et al which is required by the function, assume that it is also available in the same directory as your model, so **for eg**: if model is at ‘/path/challenge/model.pkl’, assume that the encoder is kept at ‘/path/challenge/encoder.pkl’.

FAQs:

**Q. I get InvalidBSON error when querying the data from MongoDB, what should I do?**

**A.** Please add an import statement before querying the data ‘*from datetime import datetime’*.

**Q. I get MemoryError when trying to load data from MongoDB, how do I resolve it?**

**A.** Try querying subset of data (there are unique identifiers called *“sensor\_name”* which can be used to limit the memory utilization in one go. Or you can take advantage of the timestamps available in the *“start\_epoch”* and *“end\_epoch”* parameters. You can save the query files locally and process them later or do the processing ad hoc for each subset. Feel free to use any other method you think suitable to tackle this issue.

**Q. What does the sensor\_name in the dataset mean?**

**A.** The dataset corresponds to a centrifugal pump. So, all the *sensor\_name* values correspond to some physical property being measured on a centrifugal pump. You are free to use this knowledge to explore more about the machine type and or to create different features as you see fit. **No other explanation about the features will be provided.**