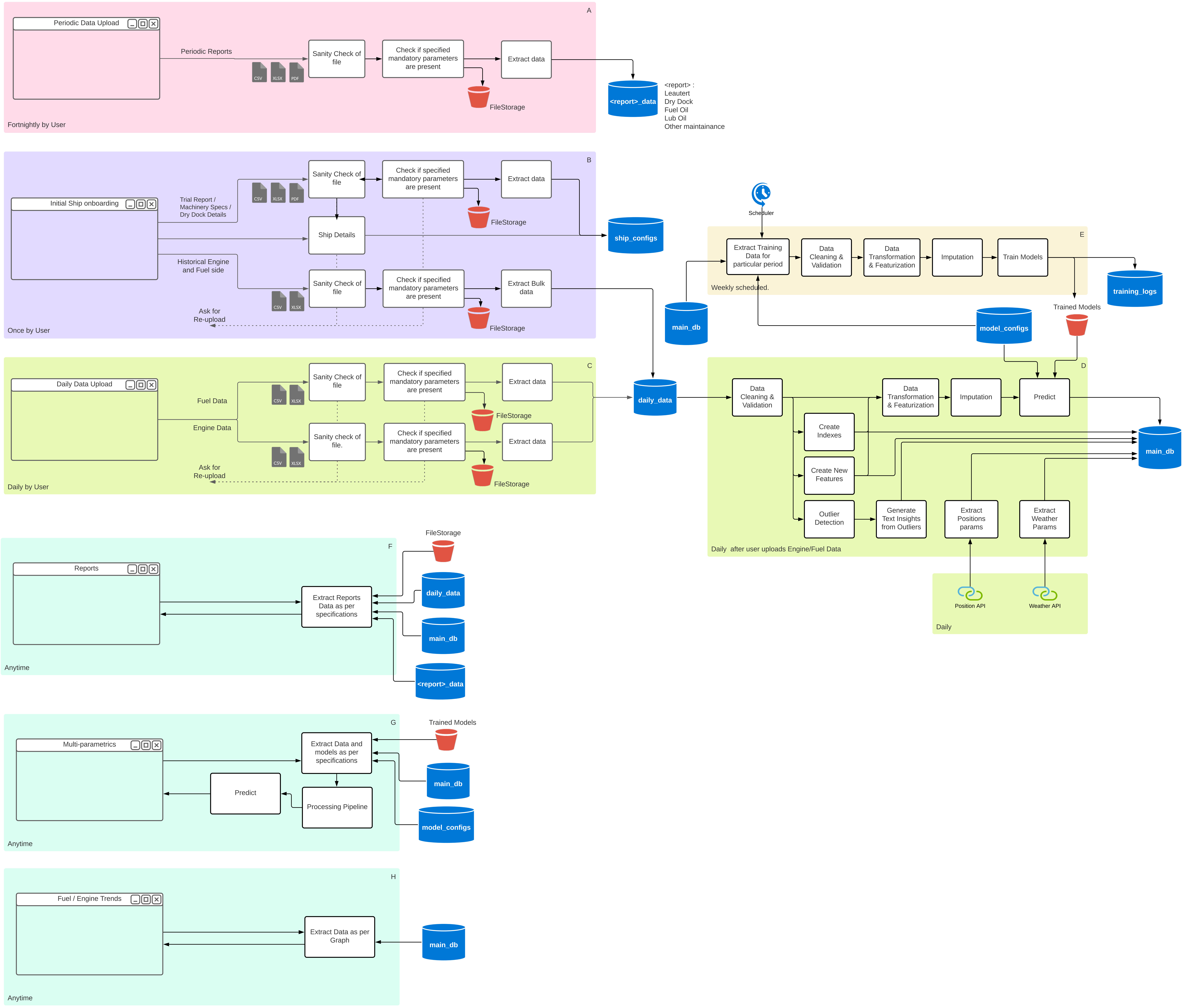
A
Periodic Data Upload
Periodic Reports
CSV
XLSX
PDF
Sanity Check of file
Check if specified mandatory parameters are present
Extract data
FileStorage
<report>_data
<report> :
Leautert
Dry Dock
Fuel Oil
Lub Oil
Other maintainance
Fortnightly by User
B
Initial Ship onboarding
Trial Report / Machinery Specs / Dry Dock Details
CSV
XLSX
PDF
Sanity Check of file
Check if specified mandatory parameters are present
Extract data
FileStorage
Ship Details
ship_configs
Historical Engine and Fuel side
CSV
XLSX
Sanity Check of file
Check if specified mandatory parameters are present
Extract Bulk data
FileStorage
Ask for Re-upload
Once by User
Scheduler
E
Extract Training Data for particular period
Data Cleaning & Validation
Data Transformation & Featurization
Imputation
Train Models
training_logs
Weekly scheduled.
main_db
model_configs
Trained Models
C
Daily Data Upload
Fuel Data
Engine Data
CSV
XLSX
Sanity Check of file
Check if specified mandatory parameters are present
Extract data
FileStorage
Sanity check of file.
Check if specified mandatory parameters are present
Extract data
FileStorage
Ask for Re-upload
Daily by User
daily_data
D
Data Cleaning & Validation
Data Transformation & Featurization
Imputation
Predict
Create Indexes
Create New Features
Outlier Detection
Generate Text Insights from Outliers
Extract Positions params
Extract Weather Params
main_db
Daily after user uploads Engine/Fuel Data
Position API
Weather API
Daily
F
Reports
Extract Reports Data as per specifications
FileStorage
daily_data
main_db
<report>_data
Anytime
G
Multi-parametrics
Extract Data and models as per specifications
Trained Models
main_db
model_configs
Processing Pipeline
Predict
Anytime
H
Fuel / Engine Trends
Extract Data as per Graph
main_db
Anytime

Block Diagram Short Description

Block A

User uploads various Reports related to Machinery and Maintainance. These reports are uploaded fortnightly. Reports might be in different format and sometimes OCR can also be required for scanning and extracting the text. There are around 5-6 types of reports and each one of them will have separate collection.

Block B

When new ships needs to be onboarded on aranti, we need few details of ships and we call it ship configurations. Various Ship configurations:

1. List of variables ship will be uploading
2. Limits(High and Low) of these variables
3. Static Data of Ship like (Length, Mfg Year etc) around 30

Also, historical data is uploaded for last ~2 years in the form on Excel/CSV files. This data will be extracted and stored in daily_data collection. To distinguish between historical data vs current daily data, flag `historical` will be set. Historical data can come into two files viz. Fuel and Engine data.

Block C

Users upload daily data (Engine and Fuel), which is parsed from source files such as xls/csv and stored in collection daily_data. Daily data can come up to 6 times per day. After daily data is received, subsequently block D is executed which processes all the data.

Block D

Block D is executed immediately after receiving daily data from block C. Block D encapsulates the entire ML pipeline, insights generator, data pre-processing, and predictions from stored models from Block E. This block also fetches API data from external services for weather and vessel position. Everything is stored in main_db.

Block E

Models are trained on scheduled basis like weekly/daily. After training, models are stored in the static storage like S3 in the pickle form which will fetched when needed from BLock D and G.

Block F

Will simply extract reports from the database.

Block G

On the basis of User Selection, on the fly prediction values will be calculated. Stored models will be used to predict

Block H

Extract Data from main collection and pass it to front end.

Data Flow

How data will be used. Examples:

ML Models

ML Models are trained on a scheduled basis. These models need training data of past few months. Example of what kind of data will be used for this training:

Example model:

- Scheduler (like a CRON job or Jenkins) will trigger model trainng .
- Model picks up configurations from `model_configs` collection.
- Trained on 6 training features : [w_force, sea_st, rpm, amb_tmp, sw_temp, sw_res] and One target feature [speed].
- Uses data of say last 6 months to train model i.e. ~180 samples.
- Total Shape of data would be 180 * (6 training+1target) = 180×7 = 1260
- So these 180 samples of 7 features would be needed for training from collection `main_db` .   In `main_db` collection, Each document is a *Daily Data* for a particular ship at a particular date and contains all around~ fields for that date. Refer schema. So for last 6 months data, we need to fetch 180 documents from this collection. Each document will have 7 of those features(Out of ~200 total), so it will fetch `processed` field for each 7 of them from 180 documents.
- Once this data is fetched from collection, model is trained as per configs from `model_configs` . Logs of training will be stored in `training_logs`.Values predicted from the model will be stored in the same collection from which training data was fetched. i.e. `main_db` in the field `predictions`. If *online* predictions are also required, then this model pickle file is stored in the object storage like S3.

Plots

On a Front End, different features are shown as a time-series. Refer existing UI. Let's just take an example of current page FUEL Trends and see what kind of data will be needed there.

For Graphs

- We have total 6 Time-series subplots and each subplots hold multiple series. Noted below are all those 6 subplots with names of series. Data for these features will be fetched from `main_db`

  Let's just see fields first.

  1. speed - `processed`, `predicted.m12[0]`,`predicted.m12[1]`,`predicted.m12[2]`, `is_outlier`
  2. hfoc24- `processed`, `predicted.m12[0]`,`predicted.m12[1]`,`predicted.m12[2]`, `is_outlier`
  3. hfoc- `processed`, `predicted.m12[0]`,`predicted.m12[1]`,`predicted.m12[2]`, `is_outlier`
  4. slip- `processed`, `predicted.m12[0]`,`predicted.m12[1]`,`predicted.m12[2]`, `is_outlier`
  5. rpm - `processed`,`is_outlier`
  6. Weather
     1. sea_st - `processed`,`is_outlier`
     2. w_force - `processed`,`is_outlier`
     3. swell - `processed`,`is_outlier`
     4. current - `processed`,`is_outlier`
     5. w_dir - `processed`,`is_outlier`
     6. rel_deg - `processed`,`is_outlier`

Now, descriptions for each field:

`processed`: Cleaned values.

`predicted`: Predicted values with confidence interval. Hence the indexing.

`is_outlier`: Boolean series of denoting whether data is an outlier or not.

Total number of series required : 23 floats + 10 boolean = 33 as per above list. By default, time series is shown for 365 Days. so 365 * 33 =- 12065 unique points.

So, 365 documents will be needed to fetch and from each document those 33 fields for a day will be needed. There are few other options as well like - Comparisons which will require addional 4-5 series.

This was about graphs. Now along with graphs, tables are shown on left and right panels which show data for the day. It changes as user moves the cursor on time series for different dates.

For Tables:

Current Example: On left pane, fuels are show and on right pane other variables are shown. Now these are flexible and not yet decided and some fields might not have data for everyday. (Like Fuels). For each feature, we are showing reported and expected value (`processed` and `predicted.m12[1]`)

1. Left Table- 10 Fields - `processed` , `predicted.m12[1]`
2. Right Table -20 Fields - `processed` , `predicted.m12[1]`

Total = (20+10)*2*365 = 21k points.

This was example for Fuel Trends. Similarily we have Engine Trends and Daily Data where we have different features to be shown in a bit different format, but they all come from the same collection `main_db` .

Multiparametrics

Multiparamterics is nothing but Online Machine Learning on specific parameters. Here, trained models are prestored is static object storage lilke S3 and are used to predict values on the fly.

Because they are on the fly, data needs to be fetched from collection as per the user inputs on UI and then inference engine runs on that data. These infered values are shown on plots.

Reports

There'll be 10-15 kinds of reports. User can request data for specific duration,specific fields, and few other conditions.. This data will be presented to user in form of tables/excel/pdf files. Mostly data will be fetched from `main_db`

Discussion:

1. Which date format be used ? Few available options are MongoDB inbuilt date fields like Date() and ISODate(), or epoch format, or even a string based date ? We need flitering mostly *only* on Date and not DateTime right now, For faster querying which would be best.
2. `main_db` will be updated afterwards during calculations of predictions, outliers etc.

3. Two ways to stucture daily data in `main_db` which will heavily affect performance :

Examples (Only relevant fields shown here for brevity)

1. Nested Array of Embedded Document

```
{
   "ship_imo": 9876543,
   "date": Date("2016-05-18T16:00:00Z"),
   "ship_name": "RMTCourier",
   "data": [
   {
         "identifier":"rpm",
         "name": "RPM",
         "reported":70,
         "processed": 70,
         "is_outlier": False,
         "preprocessor_results":"Passed",
         "z_score": -2.1,
         "unit":"rpm",
         "statement":"RPM is Low",
         "predictions":{
                "m3":[71,72,73],
                "m6": [71,72,73],
                "m12": [71,72,73],
                "ly": [71,72,73],
                "dd": [71,72,73]
         }
   },
  {
         "identifier":"speed",
         "name": "Speed",
         "reported":70,
         "processed": 70,
         "is_outlier": False,
         "preprocessor_results":"Passed",
         "z_score": -2.1,
         "unit":"rpm",
         "statement":"RPM is Low",
         "predictions":{
                "m3":[71,72,73],
                "m6": [71,72,73],
                "m12": [71,72,73],
                "ly": [71,72,73],
                "dd": [71,72,73]
         }
   }
   ],
}
```

1. Nested Embedded Document

```
{
   "ship_imo": 9876543,
   "date": Date("2016-05-18T16:00:00Z"),
   "ship_name": "RMTCourier",
   "data": {
       "rpm":{
             "name": "RPM",
             "reported":70,
             "processed": 70,
             "is_outlier": False,
             "preprocessor_results":"Passed",
             "z_score": -2.1,
             "unit":"rpm",
             "statement":"RPM is Low",
             "predictions":{
                    "m3":[71,72,73],
                    "m6": [71,72,73],
                    "m12": [71,72,73],
                    "ly": [71,72,73],
                    "dd": [71,72,73]
             }
       },
      "speed":{
             "name": "Speed",
             "reported":70,
             "processed": 70,
             "is_outlier": False,
             "preprocessor_results":"Passed",
             "z_score": -2.1,
             "unit":"rpm",
             "statement":"RPM is Low",
             "predictions":{
                    "m3":[71,72,73],
                    "m6": [71,72,73],
                    "m12": [71,72,73],
                    "ly": [71,72,73],
                    "dd": [71,72,73]
             }
       }
   }
}
```

Pros and Cons of #1 and #2:

1. In #1, all individual features are collected inside `data` field. But individual fields are listed as an array of Embedded Document, not identifed by field. In this case it will be to maintain schema. As new feature is added, it will appended to the list of Embedded Documents. But a lot harder to query. Because there's no field to directly search for MongoDB.

   Say we want 'rpm' `processed` value for 365 days. So firstly, 365 documents need to be fetched for that ship. Then from each document, `data` field is selected and then, there are 200 Embedded Documents(for 200 features) inside `data` from which document which has `identier:'rpm'` needs be extracted for getting `processed` value. This searching would take time.

MongoDB Query : (w/o dates for now)

```
db.main_db.find({"ship_imo": 9876543,"data.identifier":"rpm"},{"data.processed":1})
```

2. In #2, all individial fields have their own identifer as field name itself. So although it will harder to maintain schema if new features are added, it will be lot easier and fast to fetch the data.

   With the same example above for data of 365 days of rpm, MongoDb doesn't need to traverse through all Embedded Documents to get at rpm processed value. Directly `data.rpm.processed` would give the value.

   MongoDB Query : (w/o dates for now)

   ```
   db.main_db.findOne({"ship_imo": 9876543},{"data.rpm.processed":1});
   ```

File - main_db.py

```
document = {
   "ship_imo": 9876543,
   "date": Date("2016-05-18T16:00:00Z"),
   "ship_name": "RMTCourier",
   "historical":False,
   "daily_data": [
   {
         "identifier":"rpm",
         "name": "RPM",
         "reported":70,
         "processed": 70,
         "is_outlier": False,
         "preprocessor_results":"Passed",
         "z_score": -2.1,
         "unit":"rpm",
         "statement":"RPM is Low",
         "predictions":{
                "m3":[71,72,73],
                "m6": [71,72,73],
                "m12": [71,72,73],
                "ly": [71,72,73],
                "dd": [71,72,73]
         }
   },
  {
         "identifier":"speed",
         "name": "Speed",
         "reported":70,
         "processed": 70,
         "is_outlier": False,
         "preprocessor_results":"Passed",
         "z_score": -2.1,
         "unit":"rpm",
         "statement":"RPM is Low",
         "predictions":{
                "m3":[71,72,73],
                "m6": [71,72,73],
                "m12": [71,72,73],
                "ly": [71,72,73],
                "dd": [71,72,73]
         }
   }
   ],
   "weather_api": [
       {
           "identifier":"tempC",
           "value":25
       },
       {
           "identifier":"swell",
           "value":6
       }
   ],
    "position_api": [
       {
           "identifier":"lat",
           "value":18.520430
       },
       {
           "identifier":"long",
           "value":73.856743
       }
   ],
    "indexes": [
       {
           "identifier":"index1",
           "value":5
       },
       {
           "identifier":"index2",
           "value":8
       }
   ],


[
}
```

File - daily_data.py

```
"""
Collection to store recieved daily data as it is.

Process in short:
User uploads DD(Daily Data) for particular ship on particular data.
Before cleaning or analysing, first we have to store data as it is as we might
need to afterwards for compliance.
So this collection is used to store received data as it is.

Whether to use fields names provided by user in DD file or to use fields names
decided by us is not yet concluded.
As different ships might have different naming conventions. Example say for
Wind Force it could be 'w_force','wind','wind_force'
in their file, but for it to be consistent we can store with our identifiers.
"""

# Method 1  - Data in Array of Objects
document = {
    "ship_imo": 9876543,
    "ship_name": "RMTCourier",
    "date": Date("2016-05-18T16:00:00Z"),
    "historical":False,
    "nav_data_details":{
                "upload_datetime": Date("2016-05-18T16:00:00Z"),
                "file_name":"daily_data19June20.xlsx",
                "file_url":"aws.s3.xyz.com",
                "uploader_details":{"userid":"xyz","company":"sdf"},
    }
    "engine_data_details":{
                "upload_datetime": Date("2016-05-18T16:00:00Z"),
                "file_name":"daily_data19June20engine.xlsx",
                "file_url":"aws.s3.xyz.com",
                "uploader_details":{"userid":"xyz","company":"sdf"},
    }
    "data_available_nav": ['rpm','speed','w_force'],
    "data_available_engine": ['er_temp','er_hum','jwc1_fwin_temp'],

    "data_nav": [
    {       "identifier":"rpm",
            "reported": 70,

    },
    {       "identifier":"speed",
            "reported": 10,
    },
    ],
    "data_engine": [
    {       "identifier":"er_temp",
            "reported": 70,
    },
    {       "identifier":"er_hum",
            "reported": 10,
    },
    ],
}
```

File - report_data.py

```
# Generic Report. Each report will have it's own collection.


document = {
    "ship_imo": 9876543,
    "ship_name": "RMTCourier",
    "date": Date("2016-05-18T16:00:00Z"),
    "historical":False,
    "upload_datetime": Date("2016-05-18T16:00:00Z"),
    "file_name":"lub_oil_data_xyz.xlsx",
    "file_url":"aws.s3.xyz.com",
    "uploader_details":{"userid":"xyz","company":"sdf"},
    "data_available": ['viscocity','temp','basicity'],
    "data": [
    {       "identifier":"viscocity",
            "reported": 70

    },
    {       "identifier":"temp",
            "reported": 46
    }
    ]
}
```

File - ship_configs.py

```python
"""
Few terms:

Daily Data: User upload data for their ship daily on Aranti. Data is right now
in the form on Excel file.

"""




"""
Stores all static data about each ship. There are around 50 types of static
fields for
each of the ship, like length, year, port etc. Those are stored here.

Also, each ship might have different machinary, say about 10 systems with total
 200 fields which we
will recieve daily in form of excel file. So field 'data_available' will have
list of all those fields
which are expected to recieve daily.

Suppose user wants to add new machinery to their ship i.e. he wants to upload
new kind of daily data.
Say new cooler is fitted, then this 'data_available' should be modified.

Each time when preprocecssor code will be running, it will check this '
data_available' field to see which of the
fields to extract from dailydata.

A new ship document will be added to this collection once new ship is oboarded
on Aranti.

"""


document = {
        "ship_imo":987654, #MongoDB Index FIeld. Single Index
        "ship_name":"RMT Courier",
        "added_on": Date("2016-05-18T16:00:00Z"),#Date when ship was added
        "grt":1234, #static field #1
        "length":1234, #static field #2
        "50th_field":1234 #Upto 50 static fields of ship data like above two.
        "data_available_nav": ['rpm','speed','w_force'],
        "data_available_engine": ['er_temp','er_hum','jwc1_fwin_temp'],
        "limits_nav":[
        {
        "identifier":"w_force",
        "max":14,
        "min":0,
        },
        {
        "identifier":"speed",
        "max":50,
        "min":0,
        },
        ],
         "limits_engine":[
         {
         "identifier":"rpm",
         "max":120,
         "min":50,
         },
         {
         "identifier":"er_temp",
         "max":20,
         "min":60,
         },
         ],
 }
```

```python
"""
Sample for Multiparametric model configurations. Similar for time- trending.

"""

document = {"name":"speedfoc",
          "training_features":["speed",'w_force',"slip","others"],
          "target_feature":"foc",
          "model_filename":"speedfoc.pickle",
          "durations":["3m","6m","9m","12m","dd"],
          "model_type":"abc",
          "confidence_alpha":0.95,
          "standardize":True,
          "polynomials":2,
          "polynomials_interaction":True,
          }
```

File - training_logs.py

```
"""
Sample model log - Stores information whenever model is trained.
"""

document = {
    "name":"speedfoc",
    "ship_imo":9876543,
    "creation_date": Date("2016-05-18T16:00:00Z"),
    "duration":"3m",
    "model_filename":"speedfoc.pickle",
    "model_url":"aws.s3.xys",
    "training_data":100,
    "training_score":0.87,
}
```