|  |  |
| --- | --- |
| |  | | --- | | **Relative location of CT slices on Human Body** | |

**Problem Statement: -** The data was retrieved from a set of 53500 CT images.   
  
Each CT (slice is described by two histograms in polar space.  
The first histogram describes the location of bone structures in the image whereas the second histogram is the location of air inclusions inside of the body.  
Both histograms are concatenated to form the final feature vector.  
Bins that are outside of the image are marked with the value -0.25.

**Attribute Information: -**

1. Patient ID: Each ID identifies a different patient  
2.Value 0 - Value 241: Histogram describing bone structures  
3.Value 242. – Value 385: Histogram describing air inclusions  
4.Reference: Relative location of the image on the axial axis (class  
value). Values are in the range [0; 180] where 0 denotes  
the top of the head and 180 the soles of the feet.

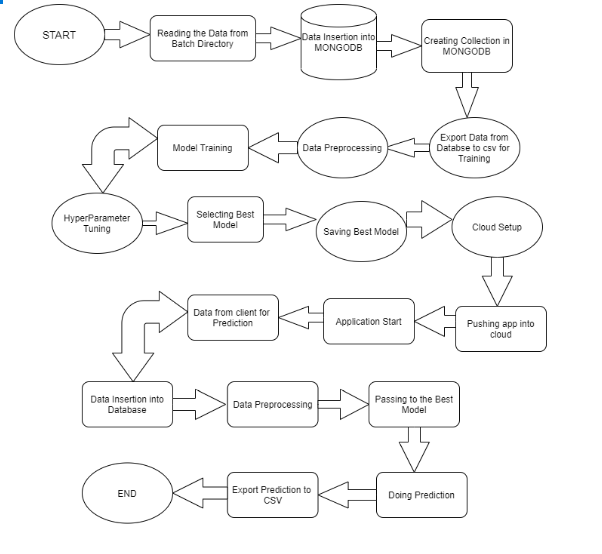
**Data Description: -**

Data contains following attributes:

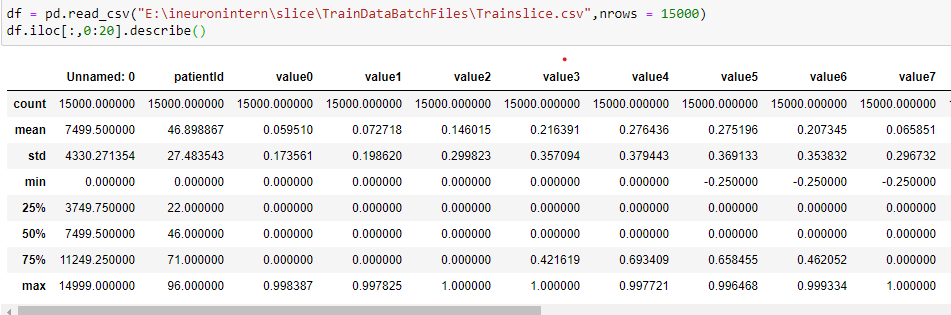
1. Number of Instances: 15000
2. Number of Attributes: 386
3. Attribute information: Given is the attribute names, attribute type, the measurement unit and a brief description. The Reference is the value to predict the location of image on the human body. It is a Regression problem.

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Types** | **Description** |
| Patient ID | Numerical | IDs |
| Value 0-  Value 241 | Continuous | Location of Bone structures |
| Value 242 -  value 385 | Continuous | Location of air inclusions in body |
| Reference | Continuous | Relative Location of image on body |

**1. Architecture: -**



**2. Statistics for some columns of the dataset: -**



**Conclusion from above result:-**

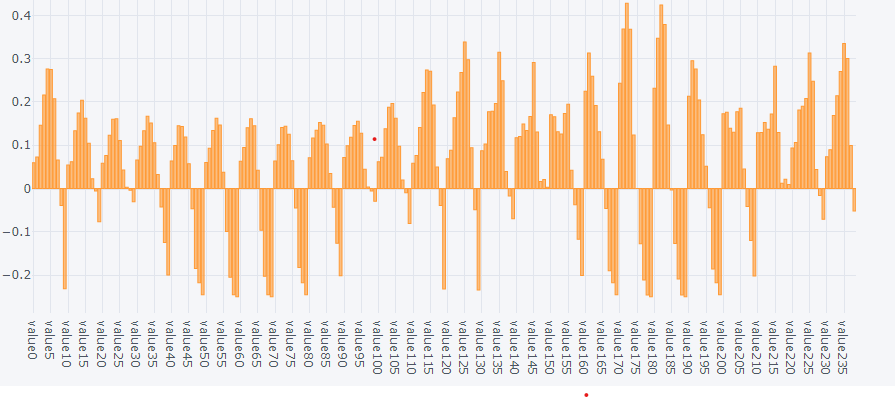
* Mean: Most of the columns are zeroes
* Median : Median for some of the columns are zeros
* Max: Most of the columns values are less than 1 and around
* From above Statistics, we can infer that most of columns are lying between the range of 0 and 1 and mean=median states that columns are symmetrical and most of the data doesn’t show any skewness to the left or right and hence the data is distributed normally.

**3. EDA:-**

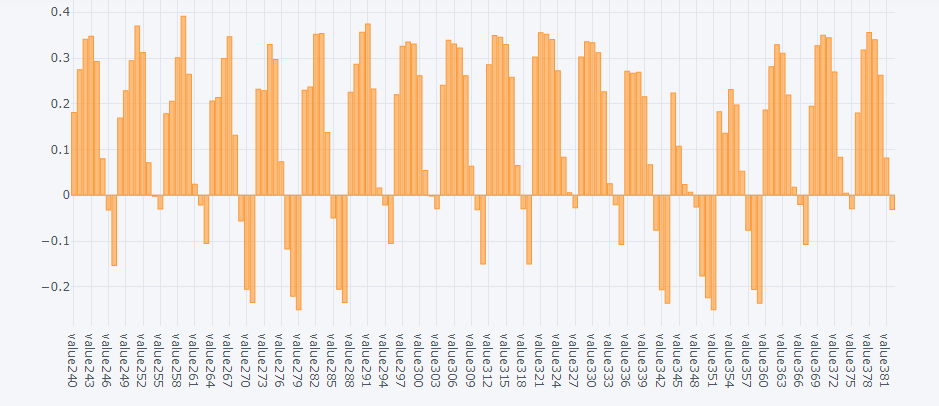
**Bar plot:-**

1. Df.iloc [: , 242:385].mean().iplot(kind = 'bar')

Below plot shows the histograms for the columns value0 up to value241 stating the histograms describing the location of bones and the bins that are in range of -0.25 are the bins that are outside the image location

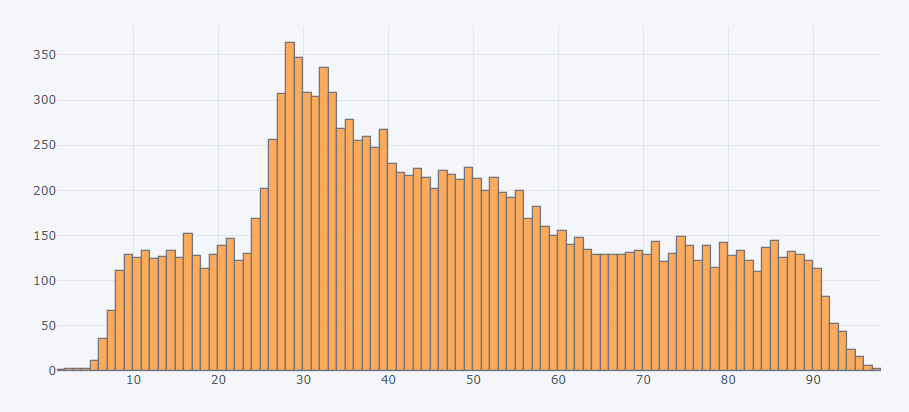


2. Df.iloc [:, 242:385].mean().iplot(kind = 'bar')



The above plot are the histograms that states the air inclusions in the body and the bins that are negative are the outside of the image.

3. Df.iloc [:,-1].iplot(kind = 'hist')

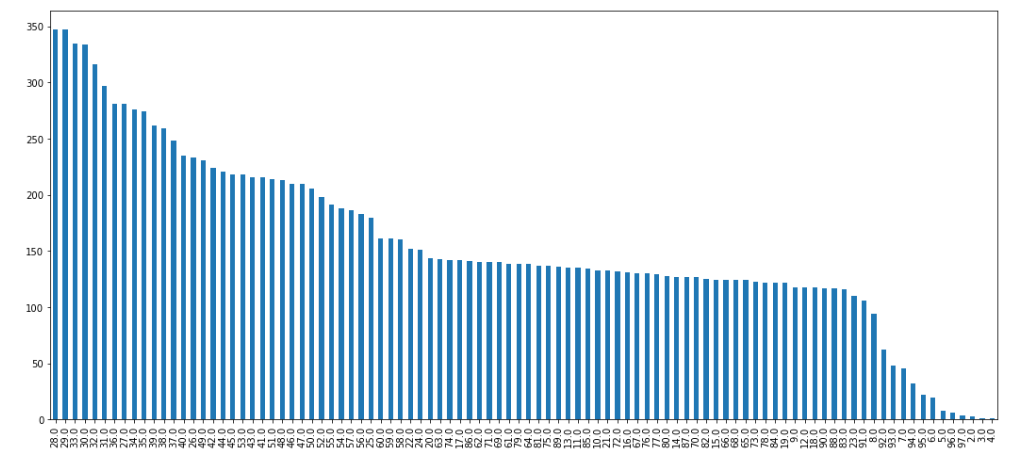


This plot is in the range 0f [0:100] stating the image located in different locations on body and the bin height is showing that how many times does that the particular location on the body is repeated for the different patients showing that the bin around the value 30 had occurred more frequently on the body like as below

4. Df ['reference'] = np.round (Df ['reference'], 0)

Df ['reference'].value\_counts ().plot (kind = 'bar')

This below plot shows the frequency of the locations in the body



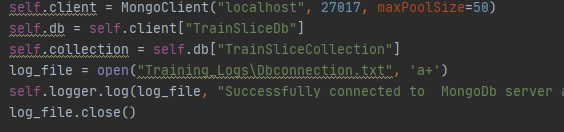
So,finally the location like 28,29,30 after rounding had occurred more frequently and I did this rounding inorder to find the value counts because it will be difficult to understand float values as well as to plot also… but to know exact location we have to consider only the numerical values as they given which tells us about the correct relative location of the image on the axial axis.

1. **Database Operations: -**

1. **Data Insertion in Database: -** In this phase we will transform the NA values and “?” into string data types before inserting the data into the database in order not to face any removal and difficulties also of the data while inserting.

**2. Database Creation and connection: -** Create a database with the given name passed. If the database has already been created, open a connection to the database. Here, I’m using MONGODB database

Below is the code given to build the connection to database.

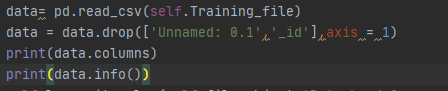


3. Data Insertion in the collection: – Data from the Training Batch folder is inserted into the mongo dB collection.



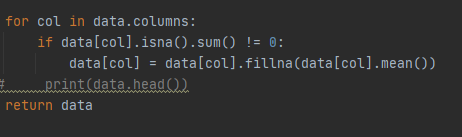
Once data is inserted in the database, we will retrieve all the data to folder Training Database folder with name as Traindf.csv and log file will be generated.

4. Data Export from DB: - The data in a stored database is exported as a CSV file to be used for model training.



5. **Data Pre-processing: -**

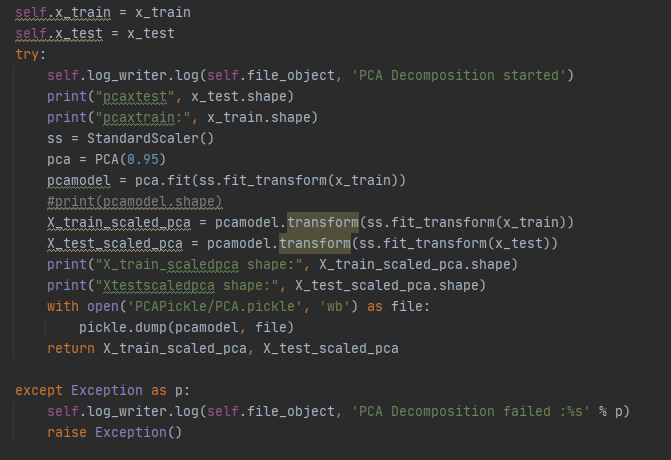
1. Check for null values in the columns: - If present, impute the missing values with mean



1. Separating the label from the data: - Here the independent and dependent variables are separated as below.

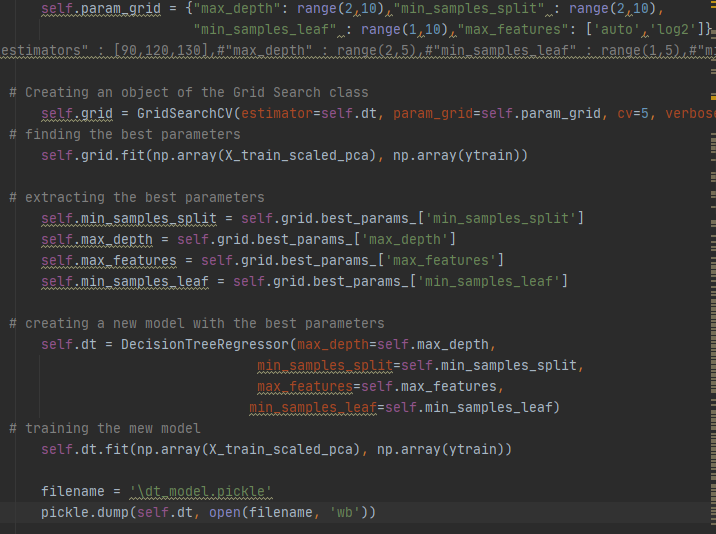


1. Now the train data is splitted into Train and test sets
   1. 
2. Doing PCA: -
   * 1. As I do have 387 columns so in order to do feature reduction and as well as to consider only the important columns that contributes the target,
   1. I choose pca method which in turn data is converted into principal components with explains the variance the most in accordance to the target.so because of this data with 387 columns is reduced into 210 principal components with 95% consideration of variance.

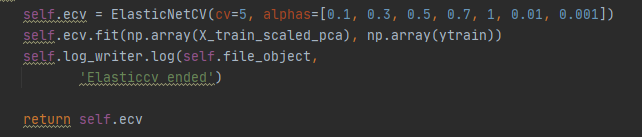


**5. Model Selection:** – After pca, I find the best model from two algorithms, “Decision Tree” and “ElasticnetCV". For both the algorithms are passed with the best parameters derived from GridSearch. We calculate the regression scores (Adjusted R2 square). We compare the models based on this score and select the model with the best score.

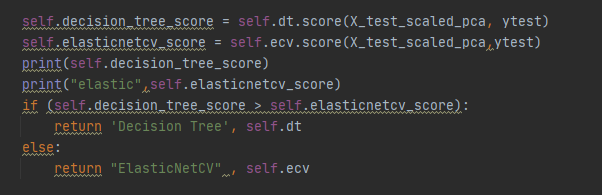
**Decision Tree Regressor:**



This is the ElasticnetCV model applied to the data



Now, we got the best model from these applied models based on the scores of both the models.



Result of model scores are as follows:

ELASTICNETCV = 0.86%

DECISION TREE = 0.91%

**6. Prediction Data Description:**

1. Datatype of columns: - Just checked the datatype of all the columns compared to the train dataset.

2. Null values in columns: – In order to insert the data into the database, just converted all null values into strings to avoid any data insertion errors

**Data Insertion in Database: -**

**1. Database Creation and connection: -** Create a database with the given name passed. If the database is already created, open the connection to the database.

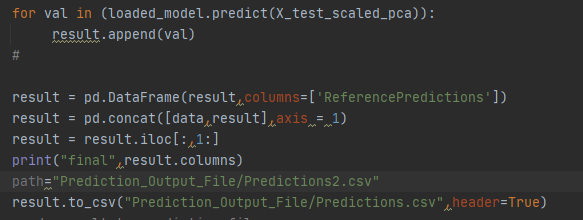
**2. Collection creation in the database: –** A collection is created to insert the data into the MONGODB database.

**3. Insertion of files in the table: -** All the data is inserted into collection.

**4. Data Export from DB: -** The data in the stored database is exported as a CSV file to be used for prediction.

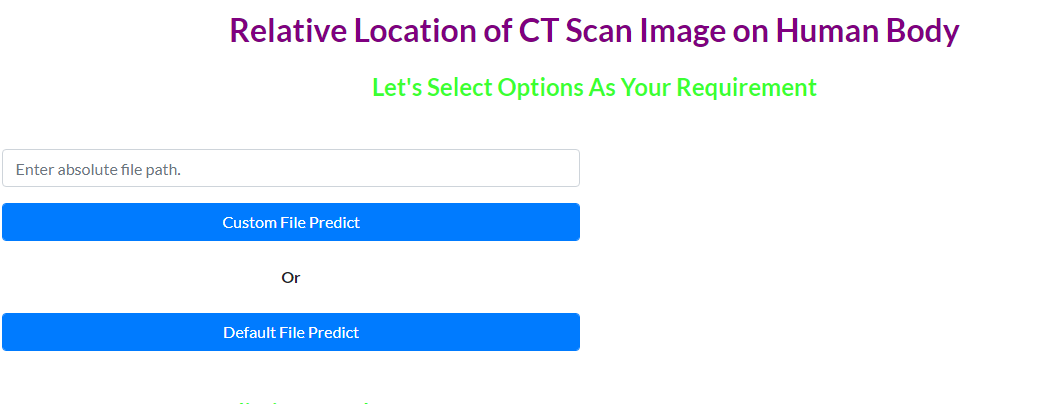
**3. Data Pre-processing: -**

1. Drop the columns not required for prediction.
2. Imputing the missing values with mean
3. Did Standard Scaling in order to bring all the data into 1single scale so that we can apply pca to the data
4. PCA is applied to the data where 387 columns are reduced into 210 principal components with 0.95% variance explaining the target. This is done by unpickling the pca data(train data) and did just transformation
5. Now the data is passed into the final best model which did predictions of the target column known as reference column. This predictions just gives us the relative locations of the image on the axial axis of the body.
6. I appended that column to the test data and as shown below.



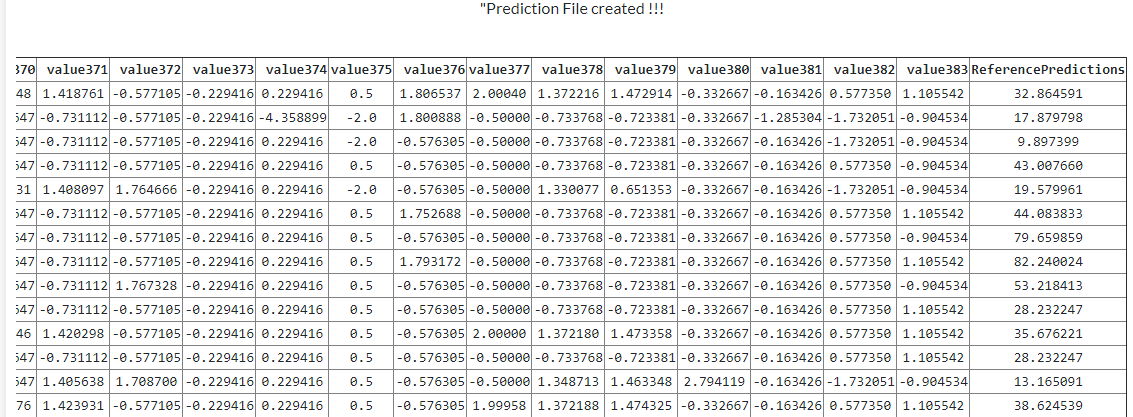
h) The predictions are stored in a separate folder by concatenating the given client data with the results given by the model.

1. Result will be shown on user interface like this



On clicking on default file prediction as given below

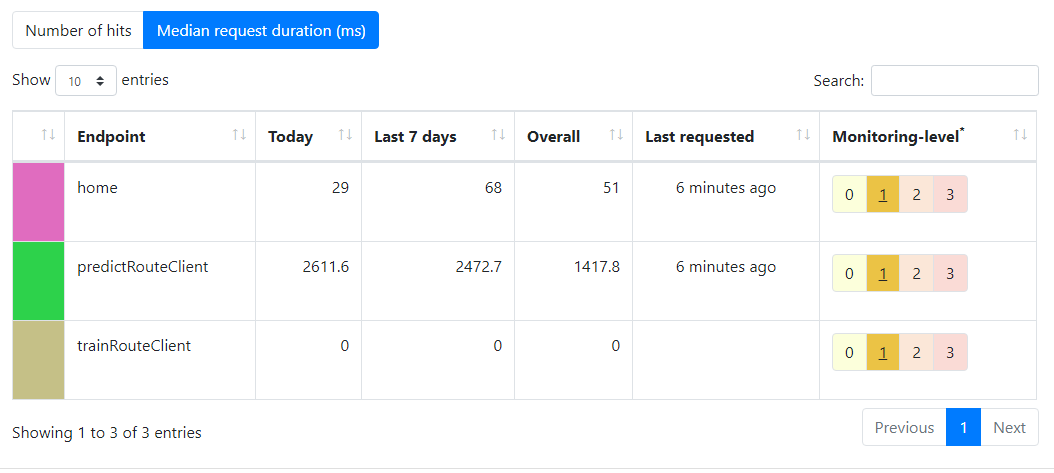
Result of prediction will be shown in prediction page in user interface page



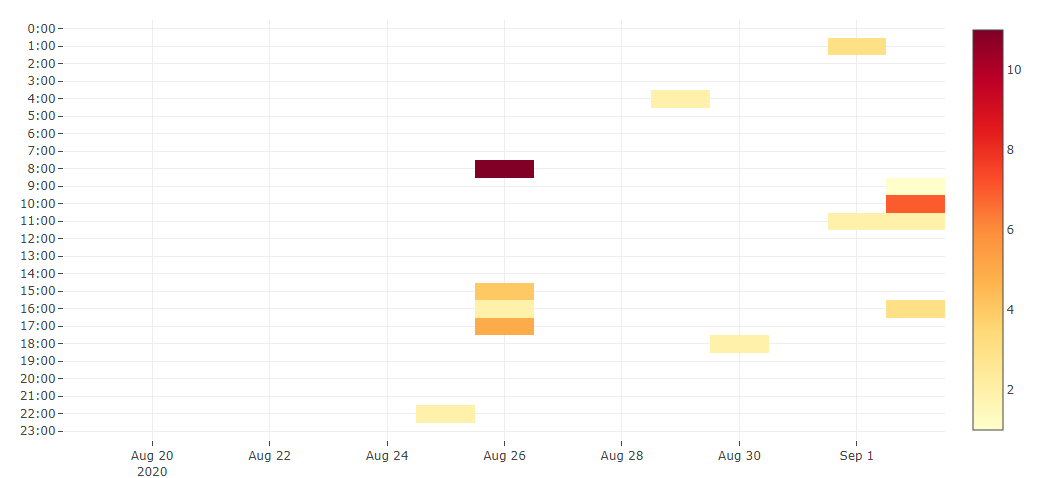
7. FLASK MONITORING DASHBOARD:

When we add dashboard to the URL like ‘localhost:5000/dashboard’ then we get flask dashboard where we have to login with admin as username and password then we will get some key performances about our API like HOURLY API UTILISATION,API PERFORMANCE,DAILY API UTILISATION ETC

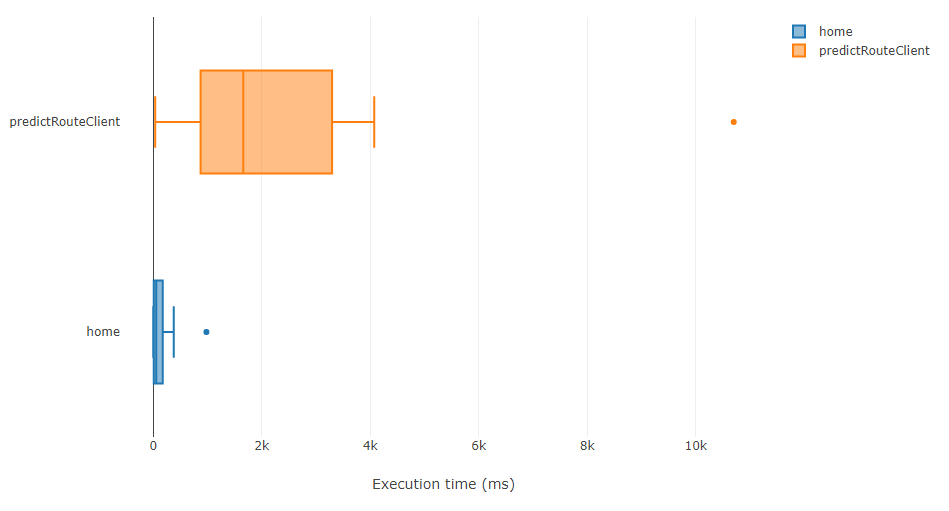
OVERVIEW:



HOURLY API UTILISATION:



API PERFORMANCE:



**6. Deployment into Heroku: -**

1. **Flask App: -**

As we’ll expose the created model as a web API to be consumed by the client/client APIs, we’d do it using the flask framework.

The flow of our flask app will be:





**2. Project Structure: -**

In order to deploy into heroku, there should be some project structure.

So we need to create some files

**PROC File: -**

We need to add a proc file inside the ‘slice’ folder. This folder contains the command to run the flask application once deployed to the server.

web: gunicorn app:app

Here, the keyword ‘web’ specifies that the application is a web application. And the part ‘app: app’ instructs the program to look for a flask application called ‘app’ inside the ‘app.py’ file. Gunicorn is a Web Server Gateway Interface (WSGI) HTTP server for Python.

Add a file called ‘gitignore’ inside the project folder. This folder contains the list of the files which we don’t want to include in the git repository. My gitignore file looks like:

.idea

As I am using PyCharm as an IDE, and it is provided by   
 Intellij Idea community, it automatically adds the idea   
 folder containing some metadata. We don’t include them in   
 our cloud app.

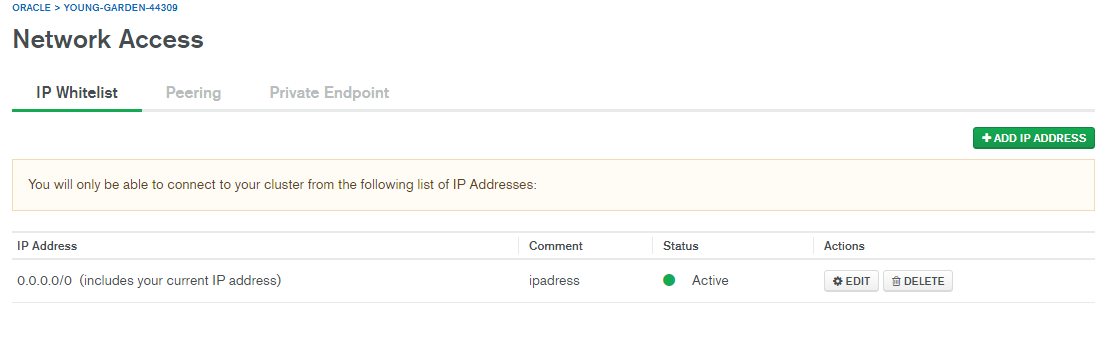
Open a command prompt window and navigate to ‘slice’ folder. Enter the command ‘pip freeze>requirements.txt’.This command generates the requirements.txt file. This requirements.txt file helps the Heroku Cloud app to install all the dependencies before starting the webserver.

**Deployment into Heroku: -**

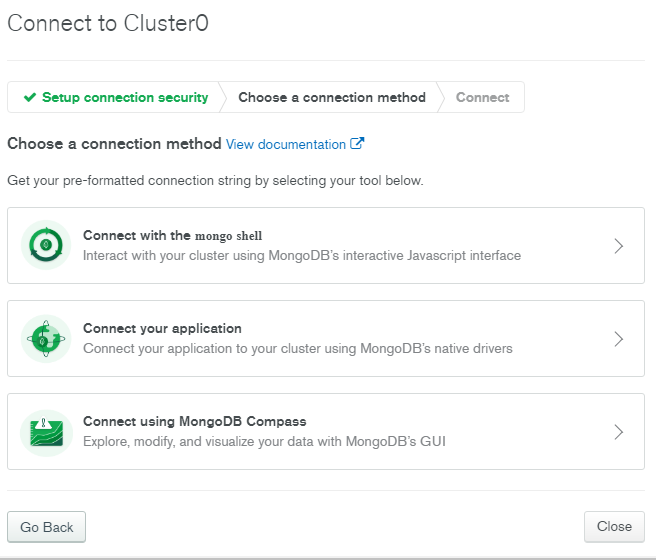
1. After installing the Heroku CLI, Open a command prompt window and navigate to your project folder.
2. Type the command ‘heroku login’ to login to your heroku account as shown below:



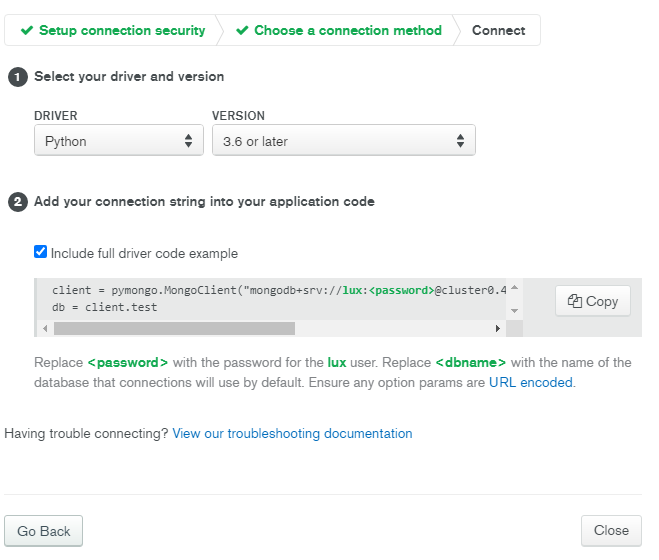
1. After logging in to Heroku, enter the command ‘heroku create’ to create a heroku app. It will give you the URL of your Heroku app after successful creation.
2. Before deploying the code to the Heroku cloud, we need to commit the changes to the local git repository.
3. Go to the ‘https: //mlab.com/login’ page and signup to use the MONGODB database.
4. Signup with your corresponding mail or with goggle account.
5. Now I created a free cluster with AWS service by selecting corresponding Zone as INDIA-MUMBAI which is free tier.
6. Now go to database access tab on the left and click on NEW DATABASE USER. Here I selected password Authentication Method, so I should give some username and password to connect to the database and importantly need to select read and write to any database option from the dropdown box given under Database User Privileges because if we not select this, then we don’t have read and write access to the database.
7. Select the Network Access and click on ADD IP ADDRESS and in Whitelist Entry type ‘0.0.0.0/0’ to made the app run at any IP Address.



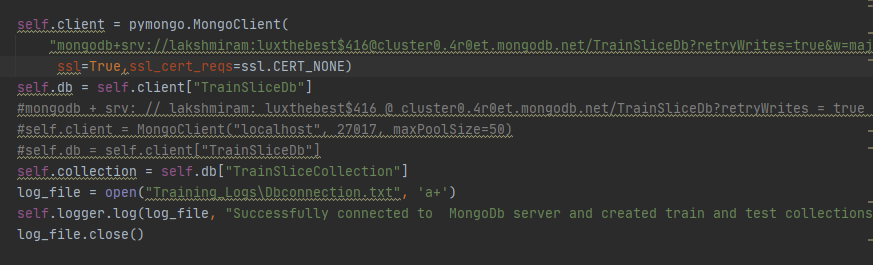
1. Again select clusters tab on the left and select connect tab.



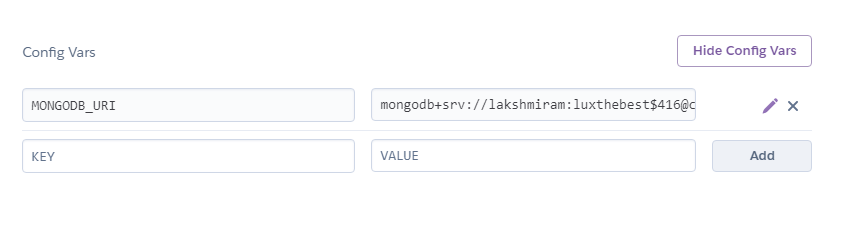
1. Select connect your application option and select python in the DRIVER dropdown box and I selected the version as 3.6 or later in the VERSION dropdown box.

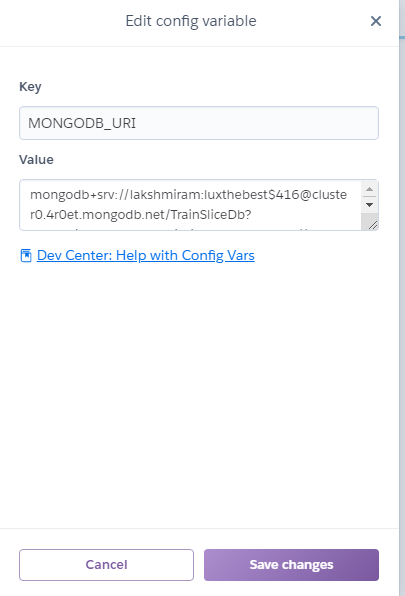


1. By ticking the option include Full Driver code example, we will get the code to connect to cloud MONGO DB instance and I have to replace MONGO DB connection in PyCharm with Cloud MONGODB instance code copied and need to replace the code with the username and password with the username and password given in the Database Access and also the name of the Database which you need to create.

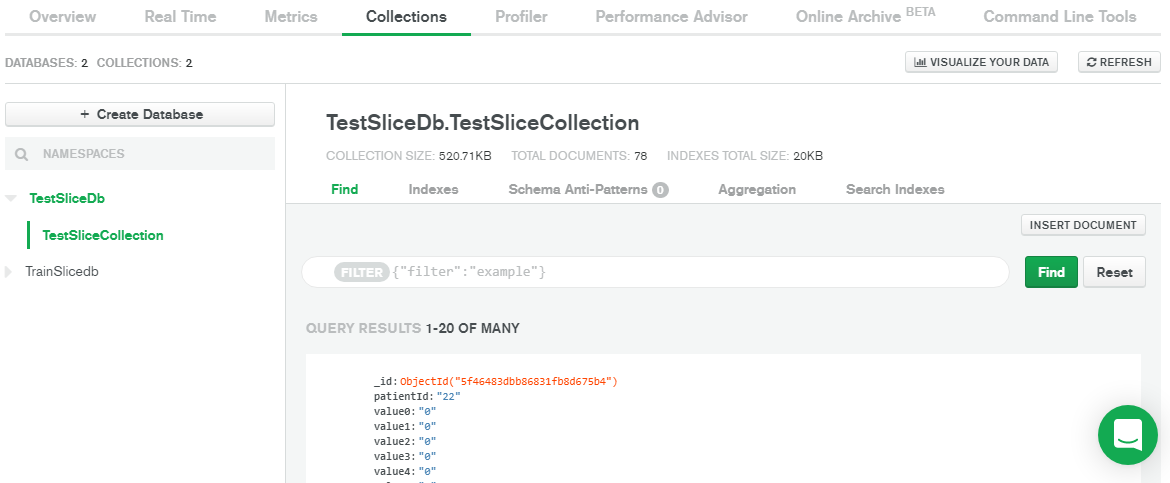


1. Copy the command with particular username and password which is in PyCharm and go the settings tab in the app page, click on Reveal Config Vars and there type MONGODB\_URI and click on edit command and paste the command “mongodb+srv://lakshmiram:luxthebest$416@cluster0.4r0et.mongodb.net/TrainSliceDb?retryWrites=true&w=majority;mongodb+srv://lakshmiram:luxthebest$416@cluster0.4r0et.mongodb.net/TestSliceDb?retryWrites=true&w=majority” and click on save changes.

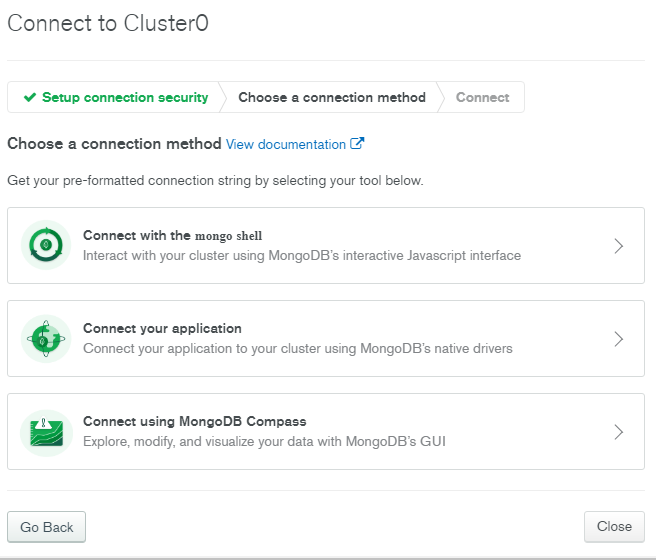




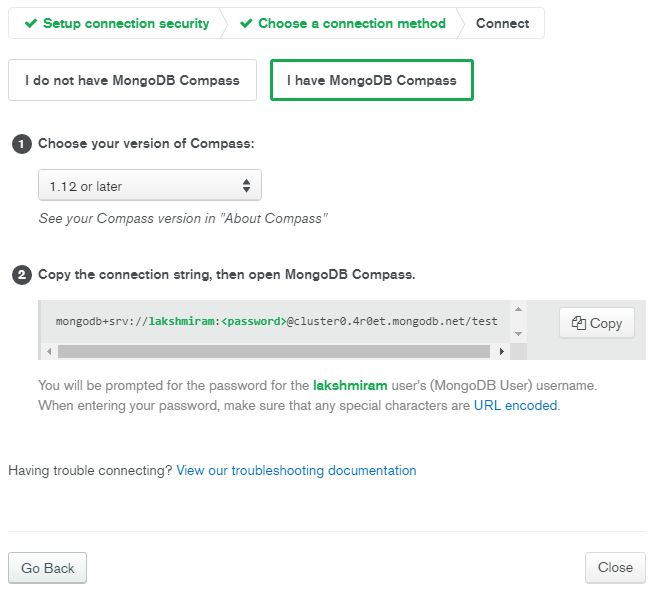
1. Now again come to command prompt and type the command ‘git init’ to initialize a local git repository as shown below:
2. Enter the command ‘git status’ to see the uncommitted changes
3. Enter the command ‘git add .’ to add the uncommitted changes to the local repository.
4. Enter the command ‘git commit -am "first commit"’ to commit the changes to the local repository.
5. Enter the command ‘git push heroku master’ to push the code to the heroku cloud.
6. After deployment, heroku gives you the URL to hit the web API.
7. This is my app URL, ‘<https://dashboard.heroku.com/apps/young-garden-44309>’
8. Type heroku open command or click on Open app in heroku page ‘<https://dashboard.heroku.com/apps/young-garden-44309>’ which will redirect to Prediction Page.
9. In order to check collections created in the MONGODB click on collections where it will redirect to Collections page, click on Refresh tab we created by running all the above commands.



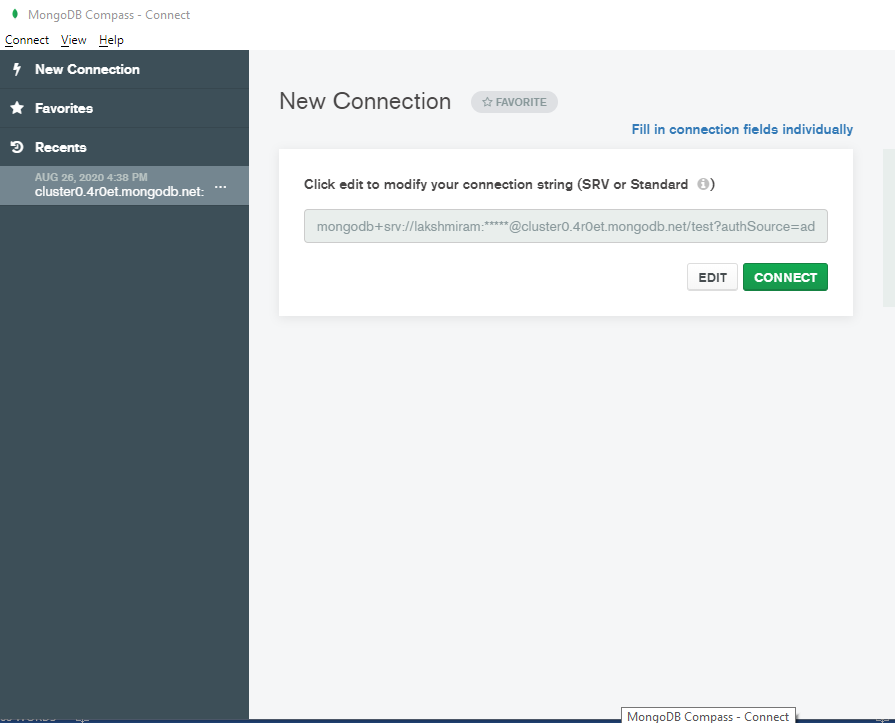
1. Once your application is deployed successfully, enter the command ‘heroku logs --tail’ to see the logs.



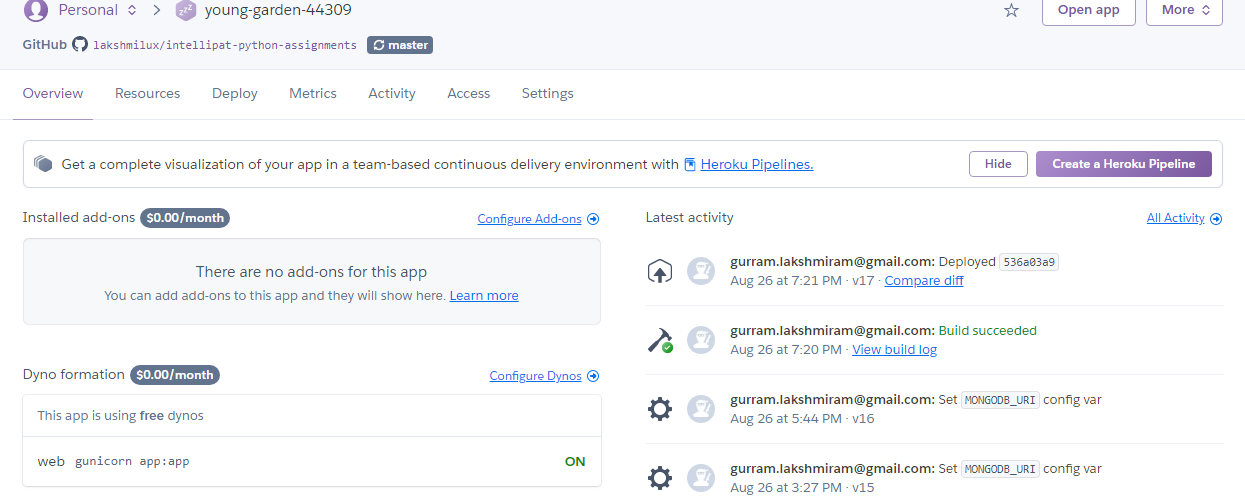
1. We can also connect to cluster0 using the above Connect using MongoDB Compass option. Just open MONGODB COMPASS installed in your pc. Once Opening of Compass, have to paste your connection string (SRV or Standard option) which is obtained as shown below.



1. By pasting the MONDODB Connection String/SRV into MongoDB Compass, it will be connected to the cloud instance of MONGODB Cluster0.



1. By running PyCharm, Collections will be created in the MongoDb Compass which will be reflected as well as in cloud cluster0.
2. Finally the heroku app page will be as below.



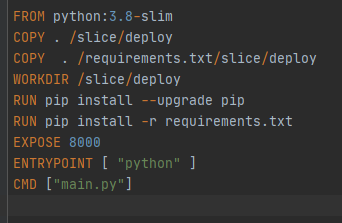
1. When you click open app in the heroku page, it will redirect us to the Final Prediction Page as below.

**7. Dockerization: -**

Docker is a tool designed to make it easier to run, deploy, and run applications by using containers. Linux containers contain applications in a way that keep them isolated from the host system that they run on.

Containers allow a developer to package up an application with all the parts needed, such as libraries and other dependencies and ship it out as a 1 package which make the developers and system environments into production in a fast and replicable way.

We need to create a Dockerfile in PyCharm slice folder as below:



FROM directive is pointing to python: 3.8-slim. This is tells Docker what base image to use for the container, with Python version 3.8, which in this case is 3.8. [Docker Hub](https://hub.docker.com/_/python/) has base images for slim Linux, a minimalist Linux distro, which helps keep the images for Docker small.

Copy directive is for the Folder where U need to run Dockerization and the WORKDIR should also be at the same location.

RUN directive that is calling PyPi (pip) and pointing to the requirements.txt file. This file contains a list of the dependencies that the application needs to run. Because Flask is a dependency, it is included as such in the requirements.txt with a simple reference *requirements.txt*

The remaining directives in the Dockerfile are pretty straightforward. The CMD directive tells the container what to execute to start the application. In this case, it is telling Python to run index.py. The COPY directive simply moves the application into the container image, WORKDIR sets the working directory, EXPOSE exposes a port that is used by Flask.

To build the image, run Docker build from a command line or terminal that is in the root directory of the application.

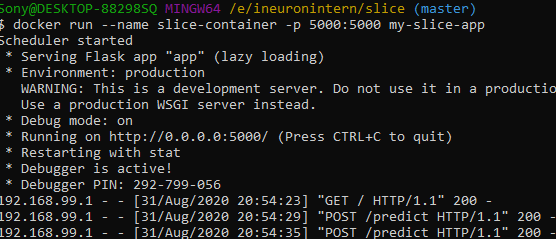
docker build --tag my-slice-app .

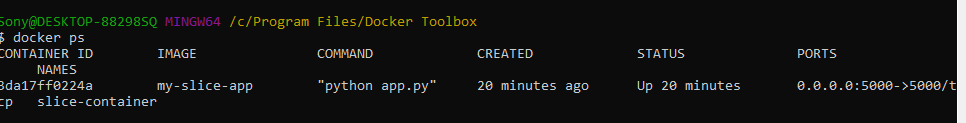
This will “tag” the image my-python-app and build it. After it is built, you can run the image as a container.

docker run --name python-app -p 5000:5000 my-slice-app

This starts the application as a container. The –name parameter names the container and the -p parameter maps the host’s port 5000 to the containers port of 5000. Lastly, my-slice-app refers to the image to run. After it starts, you should be able to browse to the container.

My Docker running on port ‘[**http://192.168.99.100:5000/**](http://192.168.99.100:5000/) **’** as docker host IP address is 192.168.99.100 and I have to add 5000 as containers port





1. The above image will give information about the docker containers and also the image name as well as the ports.