A Project Report on

Epidemic Detection using Machine Learning

Submitted in partial fulfilment of the requirement for the 8th semester

Bachelor of Engineering
in
Computer Science & Engineering

VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM

Submitted by

ADITYA	[1DS16CS009]
AKASH P	[1DS16CS012]
HRITHIK H V	[1DS16CS037]
LIKHITH R	[1DS16CS049]

Under the guidance of

Dr. Vindhya P Malagi
Professor, Dept. of CSE, DSCE

Prof. Prasad A M
Professor, Dept. of CSE, DSCE

Department of Computer Science and Engineering
DAYANANDA SAGAR COLLEGE OF ENGINEERING
BANGALORE - 560078

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

DAYANANDA SAGAR COLLEGE OF ENGINEERING

Shavige Malleshwara Hills, Kumaraswamy Layout, Bangalore-560078

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project entitled **Epidemic Detection Using Machine Learning** is a bonafide work carried out by **Aditya [1DS16CS009], Akash P [1DS16CS012], Hrithik HV [1DS16CS037], Likhith R [1DS16CS049]** in partial fulfilment of 8th semester, Bachelor of Engineering in Computer Science & Engineering under Visvesvaraya Technological University, Belgaum, during the year 2019-2020.

Prof. Prasad A M	**Dr. Ramesh Babu D R**	**Dr. C P S Prakash**
(Internal guide), Professor, Department of Computer Science & Engineering, DSCE	Vice Principal & HOD, Department of Computer Science & Engineering, DSCE	Principal, DSCE
Signature:	Signature:	Signature: ....

Name of the examiners:	Signature:
1.	1. ....
2.	2. ....

Abstract

Epidemic Detection with Machine Learning aims to predict outbreaks and potential epidemics with a good lead time and accuracy while reducing burden on healthcare experts who need to handle large numbers of reports to make such a prediction. The solution was implemented with the use of Flask, SQLite and Google Charts. Inputs are collected from medical institutions and stored in a centralized database. These inputs are processed by various machine learning models as per the user's liking and a report is generated that provides time series predictions for various diseases in different regions. These reports are rendered as graphs and charts using Google Charts API and made available to all users.

The solution performed admirably with a good accuracy and 7 day lead time. Reports were detailed and concise, and modularity of machine learning algorithms maintained.

Acknowledgement

With the pleasure of successful completion of this project titled "Epidemic Detection using Machine Learning", we would like to thank and acknowledge the following:

We would like to take this opportunity to express our gratitude to Dr. C P S Prakash, Principal, Dayananda Sagar College of Engineering, for permitting us to utilize all the necessary facilities of the institution and paving the way for innovation.

We extend our gratitude to our Vice Principal and HOD of Computer Science & Engineering, Dayananda Sagar College of Engineering, Dr. Ramesh Babu D R, for his support and encouragement.

We also extend our gratitude to our project guide Prof. Prasad A M, Dept. of Computer Science, Dayananda Sagar College of Engineering. He has been very helpful and insightful in the hurdles faced and keen-eyed in the design decisions made during the course of the project. His encouragement and guidance was a key factor to the success of this project.

We would also like to thank our project coordinator Dr. Vindhya Malagi, Dept. of Computer Science, Dayananda Sagar College of Engineering for her excellent guidance and resource allocation for project reviews and other project related activities.

ADITYA	[1DS16CS009]
AKASH P	[1DS16CS012]
HRITHIK H V	[1DS16CS037]
LIKHITH R	[1DS16CS049]

List of Figures

Contents

Introduction

1.1 Outbreak and Epidemic Detection

An outbreak is a sudden increase in occurrences of a disease in a particular time and place. An epidemic is the rapid spread of disease to a large number of people in a given population within a short period of time. The difference between the two is usually the scale of spread and number of infections in a time period.
Outbreak and Epidemic Detection deals with using large amounts of disease surveillance data to monitor and track the spread of a disease. By finding patterns of spread or uncontrolled occurances of a certain disease, an outbreak can be determined. Outbreaks are tricky to determine as many common infectious diseases have seasonal behavior where they peak in a specific time of year and remain inactive for the rest.
Medical support and healthcare is one of the biggest and risky sectors in the society. A lot of expertise and experience is required to be able to correctly determine an outbreak from a simple spread due to ignorance or seasonal behavior. Such expertise is not usually available at hand in every medical centre to catch outbreaks in early stages. Using computers to solve this problem is a viable solution.
The problem of determining an outbreak from normal behavior has been of interest lately due to increasing population densities posing higher risk of an outbreak. A good example of this is the SARS-CoV-2 pandemic occurring at the time of this report. Early detection and control of the situation in an outbreak improves the situation by reducing spread, reducing burden on medical facilities and occasionally reducing deaths.
An automated outbreak and epidemic detection system can pick up on patterns of outbreaks in early stages allowing for preparations. Since expertise and experience is not readily available at all locations, computerization allows for reliable and widespread monitoring. The results of such a system can be further verified by a professional reducing the workload of identifying potential outbreaks on them. Faster and earlier detection means lower infections, mortality and smaller spreads. An average person is less likely to be sick which improves productivity.

1.2 Machine Learning

Machine learning is a branch of computer science that deals with the concept of a computer that learns to be better at a task with more experience. In other words, machine learning deals with algorithms that make estimated guesses at the result of a new input by learning from lots of determined inputs and their corresponding outputs.

Machine learning algorithms can be classified into a few categories including: Regression, Classification, Dimensionality reduction and Clustering. Regression if the determination of one factor based on other variables, Classification is determining if an input belongs to one or more categories. Dimensionality reduction involves reducing the complexity of a large problem to a simpler one. Clustering is segregating inputs into different clusters.

Artificial Neural Networks (ANN) are a subset of machine learning algorithms that are designed to model human/animal brains. They consist of several layers of individual classifiers or neurons that produce an output if their input matches a threshold. ANNs are excellent at handling several different inputs to produce one or many outputs. They can also be modified to handle time series data.

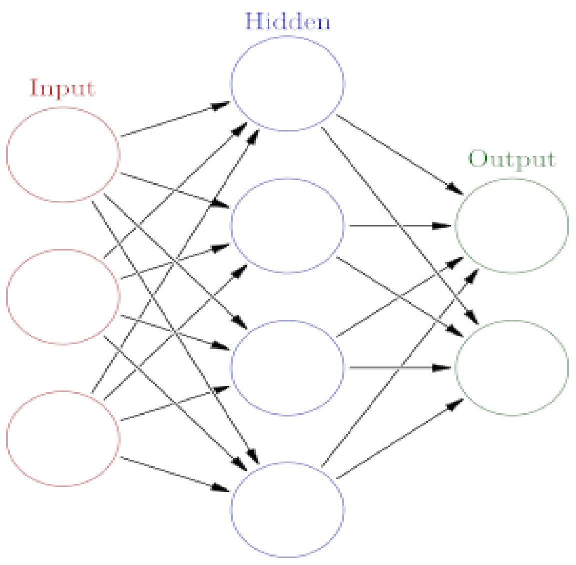


Figure 1.1: Structure of an Artificial Neural Network

In ANN and ANN based algorithms, the basic layers are common - input layer, hidden layer and output layer. The hidden layer is an abstraction to multiple layers of neurons in different configurations. The number of neurons in the input layer is equal to the number of features in the input data. The output layer consists of as many neurons as outputs. However, outputs are usually binary and several hundred neurons usually constitute a decision.

Each neuron or node consists of several inputs, weights corresponding to each input and a thresholding or activation function that determines the output. Each neuron or node is connected to all the nodes in the previous layer. RNN and CNN are variants of ANN.

1.3 Organization of this Project Report

This project report is organized as follows:
In Chapter 2, we discuss the problem statement, existing solutions and our proposed solution. Chapter 3 deals with the details and ideas obtained through a literature survey on the algorithms used to determine outbreaks and predict disease trends. Chapter 4 outlines the architecture of our solution, the design and data flow. It also reflects our design decisions. In Chapter 5, we provide the implementation details of the solution. We detail over the dataset, machine learning models, interconnections and creating a practical application. We provide the testing methodology and test results of our solution in Chapter 6.In Chapter 7, we refine the machine learning models by experimenting to determine the optimal parameters. Chapter 8 summarizes our proposed solution with future enhancements. References section contains the references and attributions to the content used in the project and this report. The code used for the project is available in Appendix A.

Problem Statement and Proposed Solution

2.1 Problem Statement

To engineer a system that can reliably predict and determine disease trends and outbreaks from previous disease surveillance records with an acceptable accuracy while providing for data collection, storage and reporting in the same system.

2.2 Existing Systems

The existing systems can be classified as manual, automated or hybrid:
Manual systems - These consist of one or more experts in the field of epidemiology and medicine who go through thousands to millions of medical reports and statistical figures to determine an outbreak or epidemic. Disease surveillance data, be it on paper or in digital form is analysed by a panel of experts regularly to determine which ones and in which regions need attention. This type of a system requires expertise and is not completely reliable, but is more useful in determining solutions to an outbreak.
Automated systems - These use computers to decide based on statistics, if a disease in a region is an outbreak or not. The statistics used by these systems are created by experts in the field of statistics and epidemiology and have significant correlation between an outbreak and the statistic. These systems on their own cannot provide solutions, and the statistics are prone to false alarms. Ex: CDC's C1,C2 and C3 algorithm, Negative Binomial
Hybrid systems - These include both automated systems that are used in conjunction with manual systems to obtain better results. The automated systems use algorithms to determine and predict outbreaks. These results are used by experts to validate the system's prediction and also to provide solutions. These systems reduce the burden on specialists, while using their expertise in the right task. This is the method used by most modern countries and our proposed solution is a hybrid one too.

2.3 Proposed Solution

Our proposed solution is a system that handles data input, processing, training and predictions with report generation. This would be a web based application for the most widespread compatibility and usability. Data is collected through web forms at various medical institutions and stored in a central database. Every day (or on demand) a report is generated for the data collected till the previous day. This report also contains predictions made by the machine learning component visualized as charts and graphs.
The machine learning component forms the core of the outbreak detection and trend prediction capabilities of our solution. Once a week, the machine learning models update based on the new data received over the past week. This allows them to get better at determining trends and outbreaks.
Our proposed solution uses an artificial neural network based predictor to predict data for the upcoming week. It then uses the prediction combined with the previous week's data to classify it as either an outbreak or not using a classifier. Each disease has its own set of prediction models and classification models to better understand the behaviour of each disease.
The models have a training and testing phase. The training phase uses historical data that is labelled by experts as outbreak or not. A recent subset of this data is withheld for testing. In testing this subset is provided as input and the accuracy obtained. The parameters are tweaked for accuracy.

2.4 System Requirements

The requirements for our proposed system is as follows:

- Accept various disease surveillance records and patient records from medical institutions through an easy user interface
- Store and analyse the data collected and cache the results of analysis
- Generate reports based on the analysed data and display them in a simple and understandable manner
- Use the data to generate predictions of future disease trends and perform recommendations about possible outbreaks
- System should be centralized and allow for an offline workflow on the medical institutions end
- Machine learning tasks should be performed on the server and not consume end user resources for predictions
- System should support multiple machine learning models, a default algorithm would perform the predictions, while others can be used to find improved models

Literature survey

3.1 Machine Learning models used for Trends prediction

Many different algorithms have been used to determine the future trend of a time series. These include Autoregressive Integrated moving Average (ARIMA), Long Short Term Memory (LSTM) networks, Ordinary Least Squares (OLS), and many other neural network inspired models.

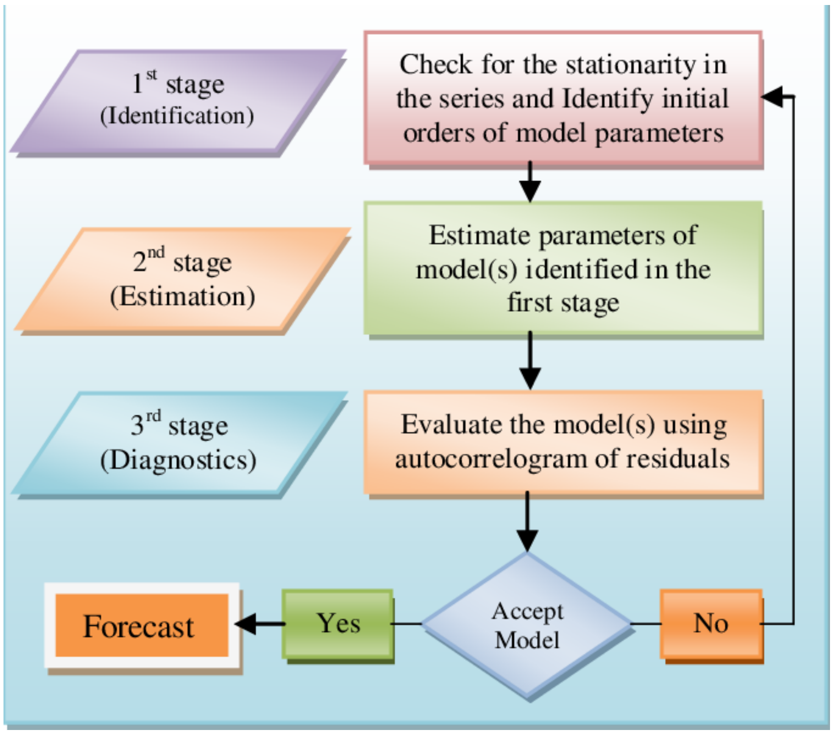

1st stage (Identification)
Check for the stationarity in the series and Identify initial orders of model parameters
2nd stage (Estimation)
Estimate parameters of model(s) identified in the first stage
3rd stage (Diagnostics)
Evaluate the model(s) using autocorrelogram of residuals
Forecast
Yes
Accept Model
No

Figure 3.1 Selection process for machine learning algorithms

Trend prediction models use a subset of previous data to predict upcoming information. For example, the number of cases in the past week can be used as input to predict the cases for the next day. ARIMA and LSTM use these techniques.

3.2 Machine Learning Models used for Outbreak Prediction

Outbreak prediction is a form of classification problem. The number of cases and trends are used to decide if it is an outbreak or not. Classifiers such as Support Vector Classifier (SVC), Multi-Layer Perceptron networks are used in the task of classification.

Architecture and Design

4.1 System Overview

The system is designed to run on a server, in a pseudo client-server architecture. The system consists of several client systems located at medical institutions and a central server connected to a database. Client systems can be classified as Admins or Users. This is a software level classification to provide permissions and tasks.
The system runs a normal web server with scheduled two background threads. The first background thread is tasked with generating the report every midnight. The second background thread is tasked with retraining the machine learning models at sunday midnight with the new data that has been accumulated over the week. This allows the models to improve over time as diseases also evolve their spread patterns.
Users use the client end of the system to add disease surveillance data such as disease name and number of cases found in a region. They can also access the reports generated everyday, but cannot generate their own report at the current time, and cannot add new locations or choose the prediction algorithm. New locations for health centers can only be added by users with database write access.

The system is designed to decouple the producers of data(PrimaryHealthCenters) with the consumers of data(Health officials). As the recent COVID-19 pandemic has taught healthcare officials throughout the world, even a week's delay in collecting critical medical data can lead to disastrous consequences. Hence, PHC users can only upload data to the system. The system runs Machine Learning Algorithms periodically and generates predictions about the disease every midnight. The consumers of data(healthcare professionals) can examine the reports and predictions to make informed decisions about pandemics and how to deal with them.
The system consists of two end-users. PHC users whose responsibility is solely to upload data. A given location can have multiple PHC users.

4.2 System Architecture

4.2.1 System Block Diagram

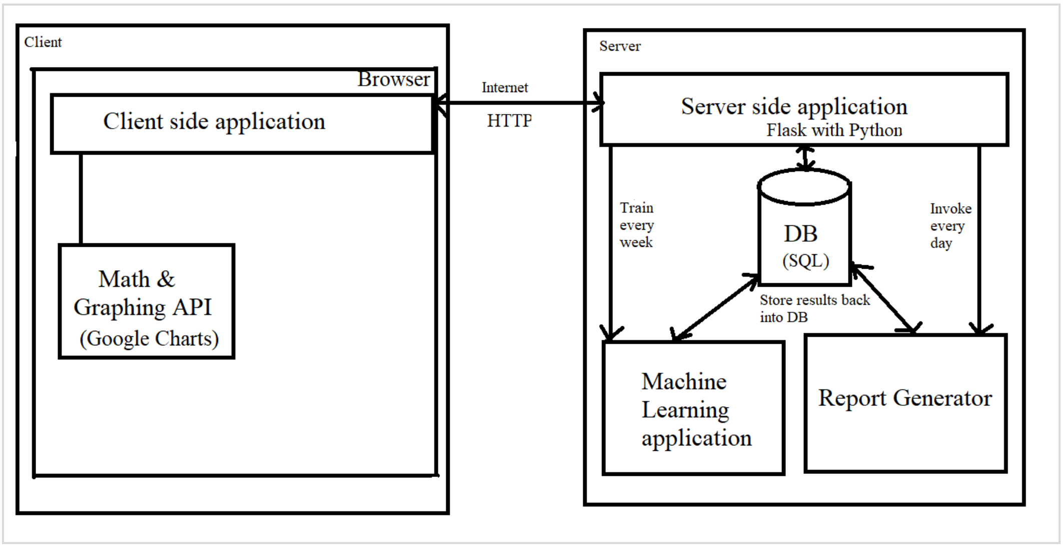


Figure 4.1 System Block Diagram

4.2.2 Use Case Diagram

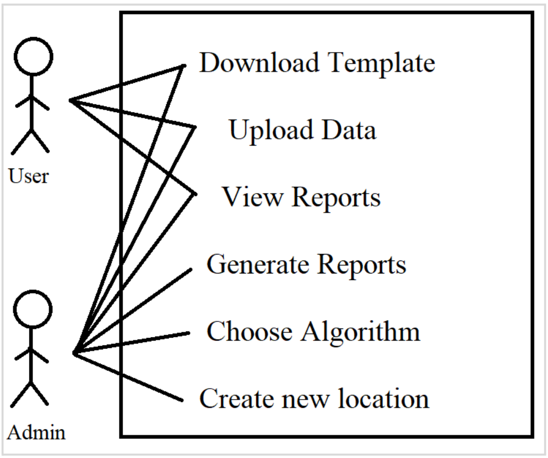


Figure 4.2 Use Case Diagram

4.2.3 Data Flow Diagram

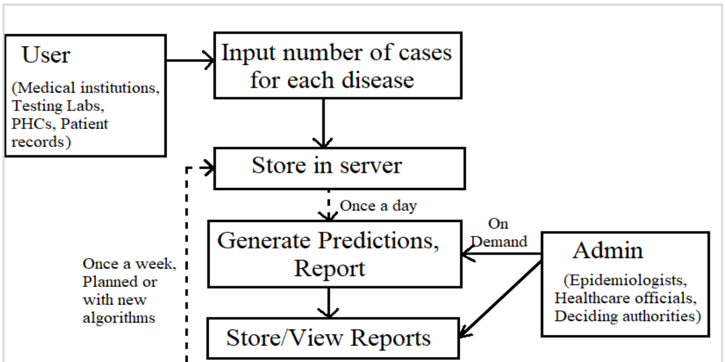

User
(Medical institutions, Testing Labs, PHCs, Patient records)
Input number of cases for each disease
Store in server
Once a day
Generate Predictions, Report
Store/View Reports
Once a week, Planned or with new algorithms
On Demand
Admin
(Epidemiologists, Healthcare officials, Deciding authorities)

Figure 4.3 Data Flow Diagram

4.2.4 Database Structure

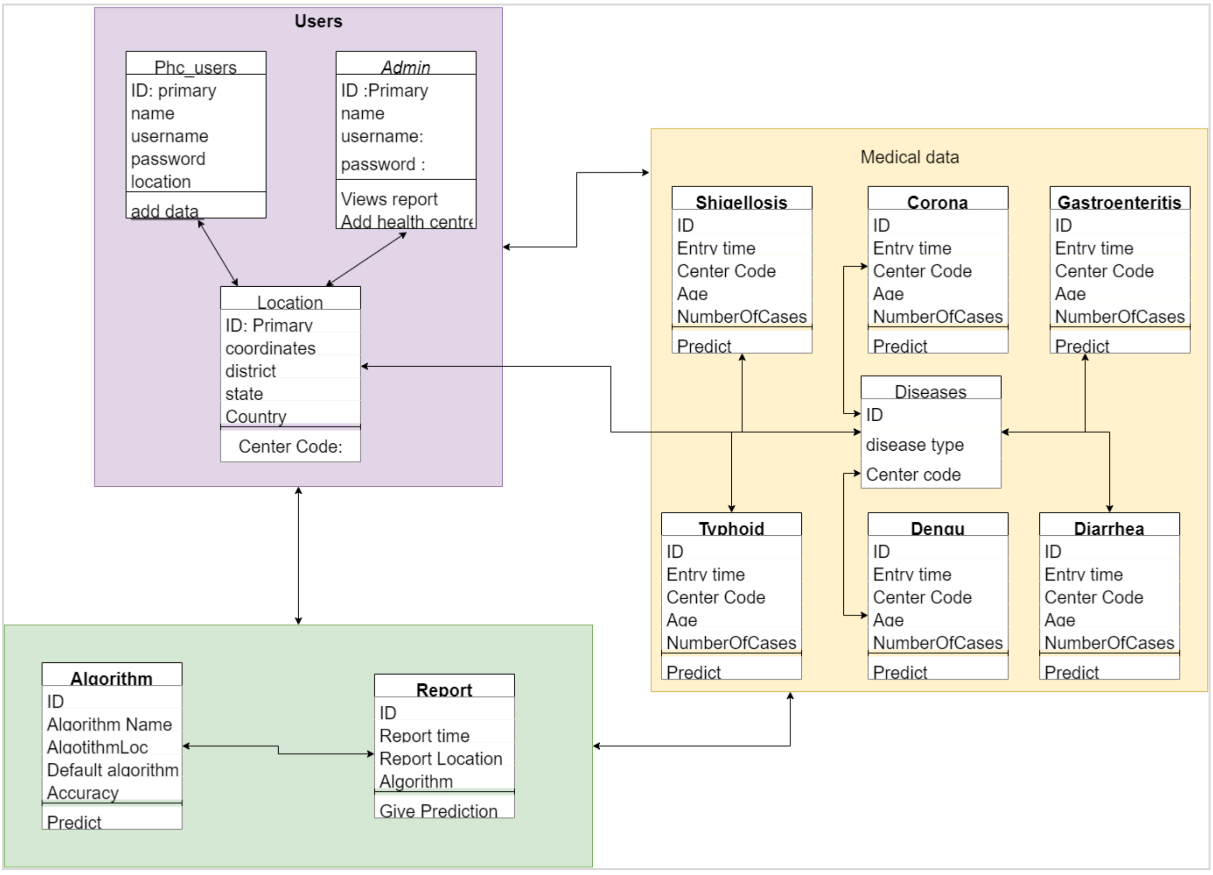


Figure 4.4 Database Structure

Implementation

5.1 Implementation Platform

5.1.1 Hardware

Implementation of this solution does not require any specific hardware configuration. The task of machine learning training can be sped up with the use of a GPU or TPU. The following hardware was used to develop and test the system.
CPU: Intel Core Mobile Series
RAM: 8GB
For an actual deployment of this solution, it is recommended that a 4-6 core CPU or a GPU be dedicated to the machine learning component. The scale of data used in this project is about 50MB and was limited to few diseases only. An actual deployment would have to cater to thousands of clients, and store terabytes of disease surveillance data.

5.1.2 Software

The proposed solution utilises the following software and APIs:

- Flask Framework - For the web server that serves web pages as the UI
- SQLite Database - For storing the data collected from the clients. It is recommended that a full scale SQL database such as MySQL or Cloud SQL be used in an actual deployment due to memory limitations of SQLite.
- SQLAlchemy - Flask interface for database communications
- HTML with CSS and JavaScript - Forms the front end UI that is served as a web page
- Google Charts - Graphing API for the reports generated
- Standard python libraries, Pandas and Matplotlib - For processing, calculations and testing capabilities. For calculating key metrics about the ML models
- Keras API with Tensorflow - For LSTM machine learning model. Works better if used with a GPU or TPU.
- Scikit Learn API - For preprocessing, and the MLP machine learning model
- Statsmodels API - For the ARIMA machine learning model

5.2 Implementation Details

5.2.1 Organization of Implementation Files

Implementation files are organised with the following directory hierarchy. Flask uses a Model-View-Controller architecture, model represents the data and database tables, view represents the UI and interface available to the user that displays the data, and

controller is the business logic that controls the view and model based on events and actions.
Due to the MVC nature of Flask, three folders - models, templates and handlers - representing model, view and controllers respectively exist. “app.py” represents the start of the Flask program. This is the equivalent of main() in C/C++. It contains the routing rules that connect a web page URL to a web page resource. “config.py” contains the configuration of the application. “datastore”
also contains the configuration for the database. Requirements.txt contains the required dependencies of the program. Template.csv is the template for data entry at medical institutions. Test1.db is the SQLite database that is used by the application. Coronavirus_reports contains the generated reports for

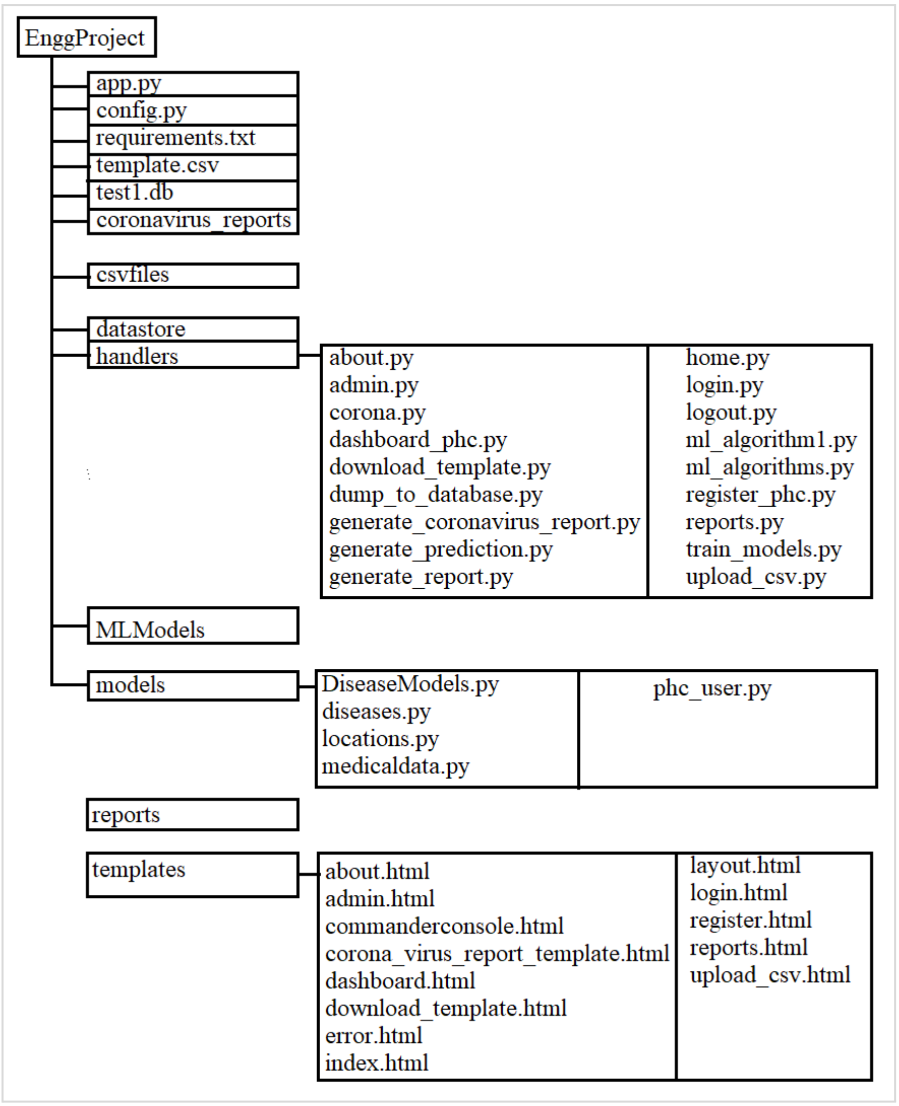


Figure 5.1 Infrastructure organisation

SARS-CoV-2 which is one of the diseases we attempt to provide time series predictions for. Csv files is the upload folder for the input provided by medical institutions into the server, once they are processed and their contents saved into the database, it is cleared. Handlers define the handlers or controllers for the views or UI provided by the application. Notable among these are corona.py - that contains the ML model training and result generation for the coronavirus predictions. Others include, generate_prediction, which routes a request for generating a report to the right prediction algorithm. Generate_report.py collates data from multiple diseases and their predictions for each region and derives statistics and graphs from it. Ml_algorithms contains machine learning algorithms and optimised parameters to generate predictions. MLModels contains the saved ML models for faster predictions. Models contain the database tables as represented by SQLAlchemy. Reports contains the generated reports that are stored for faster access. These are not deleted automatically. Templates contains the view templates for Flask. These templates are filled in with data to form UI views. Reports however do not utilize templates.

5.2.2 Dataset collection, processing, and storage

The dataset used is week wise cases for several diseases collected from a health official. The data contains cases for Bangalore, Mysore, Mangalore and Hassan districts of Karnataka, India. The weekly data was processed by cubic interpolation to yield data in the form of daily case numbers. Some of the diseases in this dataset were non-communicable, or had zero cases for most of a year. These were removed, and the validity of the interpolation was verified by checking that it follows the same distribution and aggregate matching.

The PHC users upload data about their location in the form of a CSV file. Data for each disease has to be uploaded in a separate CSV file. The name of the CSV file should be in the format DISEASENAME_DD_MM_YYYY.csv. A template for the CSV file can be downloaded by the PHC user, which is a CSV file containing the appropriate name and column names the system expects the disease to have. Every disease can have different columns. As of today, we expect the same data fields to be available for all diseases.

The template file is created dynamically. The download_template.py handler takes a disease_name and sends a file to the browser with the appropriate name and columns. An example template filename would look like “DIARRHEA_01_01_2020.csv”, with column names, “EntryTime”, “Age”, “NumberOfCases”. The PHC user would then edit the filename as appropriate and then fill in the data. When he is ready, he uploads the CSV file.

When the data is uploaded by the PHC user, it is first validated. The filename is verified and the mandatory column names for a disease are validated. The validation for the data is done in the upload_csv.py handler file. After the data is successfully validated, it is dumped to the database. The database we use is SQLite3. This part is done by dump_to_database.py file. The dump_to_database method takes a filename and a tablename. It dumps the contents of the file and dumps them to the table. It doesn’t do any validation, because it assumes the data has already been validated earlier in the flow.

Data is stored in a SQLite Database. This database was chosen due to its ease of use in prototyping. Each disease is stored in its own separate table. The predictions of each disease has its own table. This was made to allow for diseases having additional data fields that may be available for a disease that can affect predictions. For example, A malaria case tested positive with the rapid test is less accurate than a confirmed diagnosis with a microscopy test.

5.2.3 Configuration of Machine Learning Models

Two machine learning models have been developed to tackle the problem of disease trend prediction.

- Autoregressive Integrated Moving Average (ARIMA) has been used for the prediction of SARS-CoV-2 trends. Since this disease is in its first pandemic and the data available is only in this time, no outbreak prediction is applied to it.
- Long Short Term Memory (LSTM) algorithm has been used for prediction of common disease trends. These diseases are predicted with their previous data. The prediction and previous data is used together to decide if an outbreak exists. The Multi-Level Perceptron network (MLP) has been used to make this possible.
- The parameters for the machine learning models are tuned for the best accuracy.

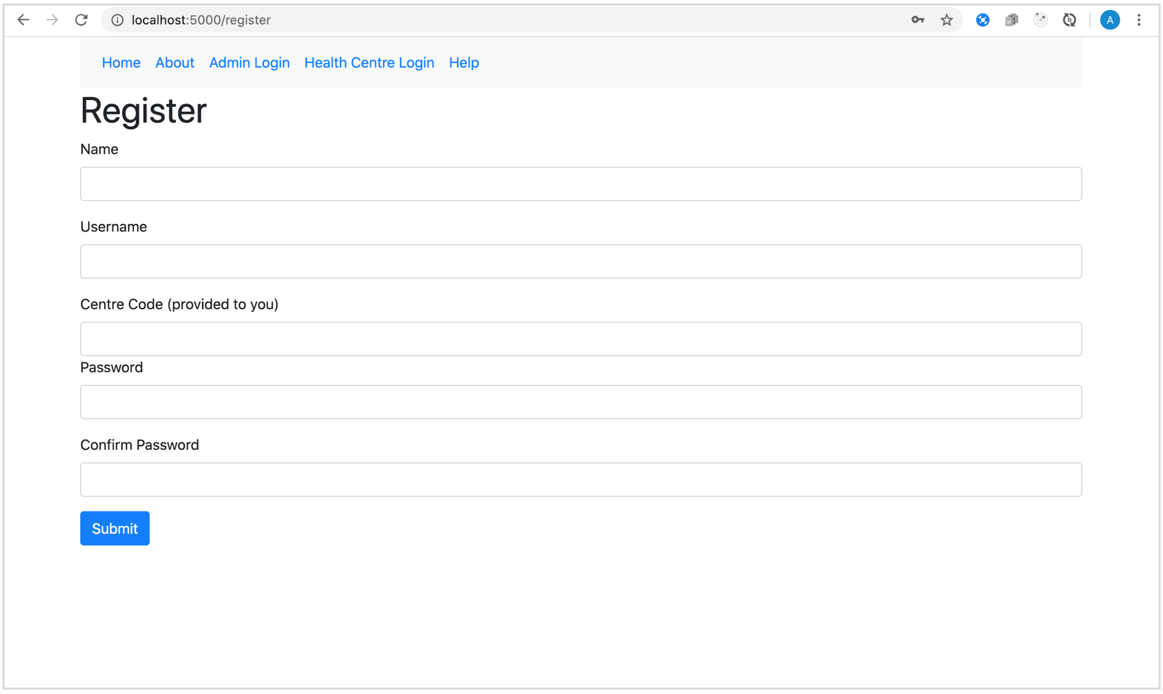
5.2.4 Training of Machine Learning Models

The dataset has data from January 2016 to December 2019. In order to facilitate time based prediction and testing for future dates, this dataset was time shifted to January 2017 to December 2020. This also results in a 75:25 split for training and testing when performed around January 2020 and 80:20 split at around May 2020. The following steps were taken for training the machine learning models:

1. Load data from the CSV file for one disease - Load the new cases per day and outbreak flag into a pandas dataframe. Split it into 75:25 for training:testing.
2. Create a LSTM network and a MLP classifier. Create a Min-Max Scaler
3. Scale the input data with the scaler
4. Using the scaled data, train the LSTM network with batches of 7 consecutive rows (days) as input and retrieve one output value that represents the next day
5. Feed the scaled data into MLP classifier with the outbreak flag and retrieve corresponding outputs.
6. Compare the output to the actual values and optimize parameters to better represent the disease
7. Store the network weights, configuration, classifier configuration and scalers.
8. Store an entry for the disease, locations of ML Model and scaler in the database

5.2.5 User Interface and Workflow

A PHC user first registers with the system. The registration functionality is provided by the */register* route.


localhost:5000/register

Home About Admin Login Health Centre Login Help

Register

Name

Username

Centre Code (provided to you)

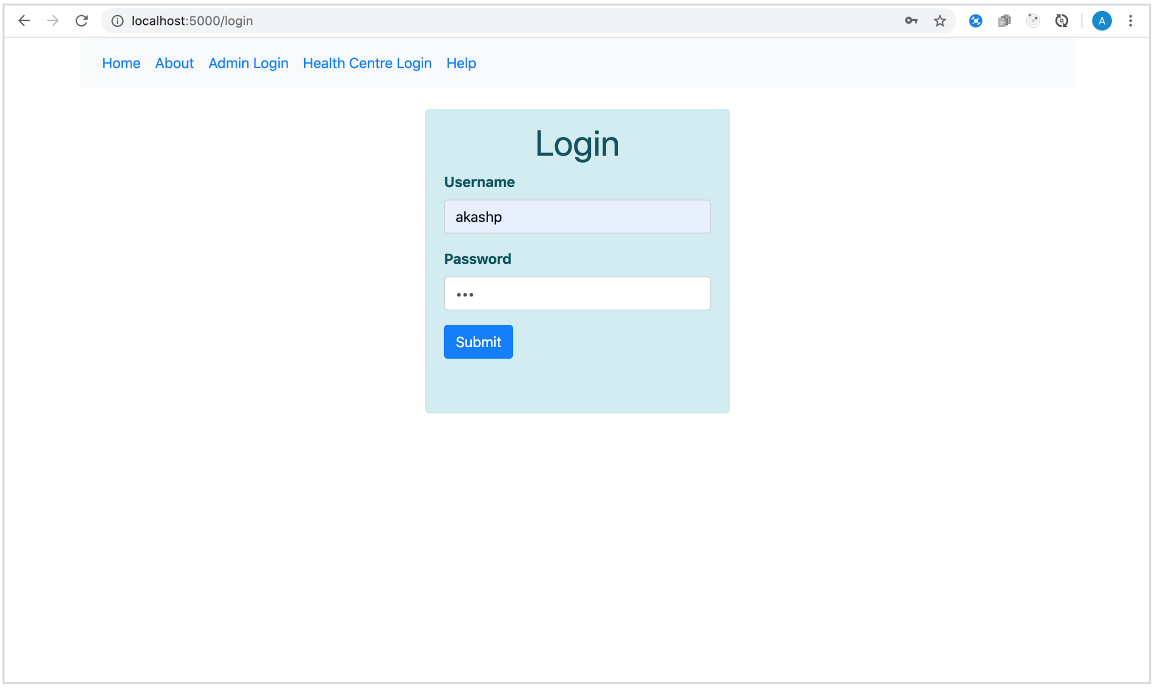
Password

Confirm Password

Submit

Figure 5.2: User Registration Page

After successfully registering, he may then login. The login functionality is provided by the */login* page for PHC users.


localhost:5000/login

Home About Admin Login Health Centre Login Help

Login

Username

akashp

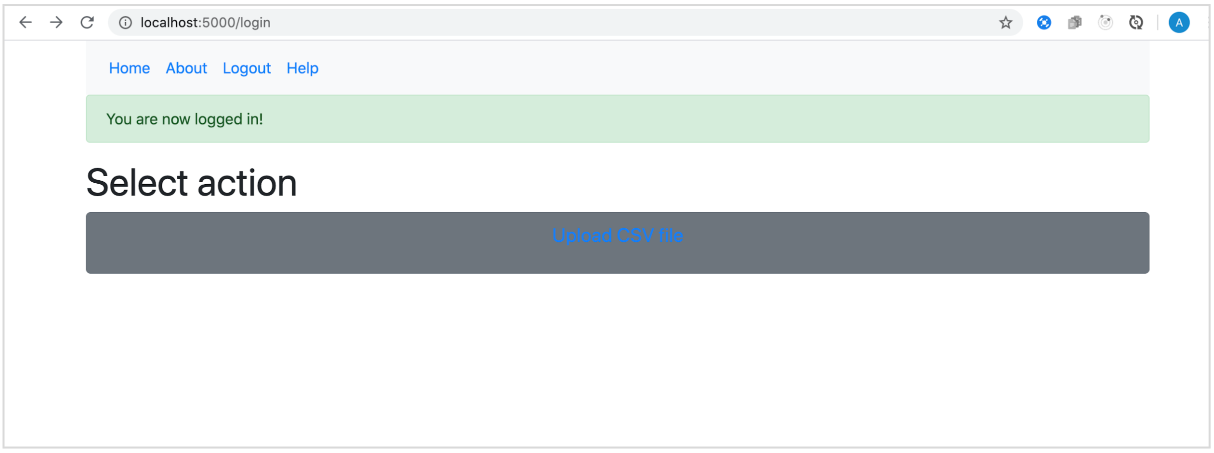
Password

•••

Submit

Figure 5.3: Login Page

He will then land on the dashboard page, which is provided by the */dashboard* route. This page currently supports just one option, i.e, to upload data.

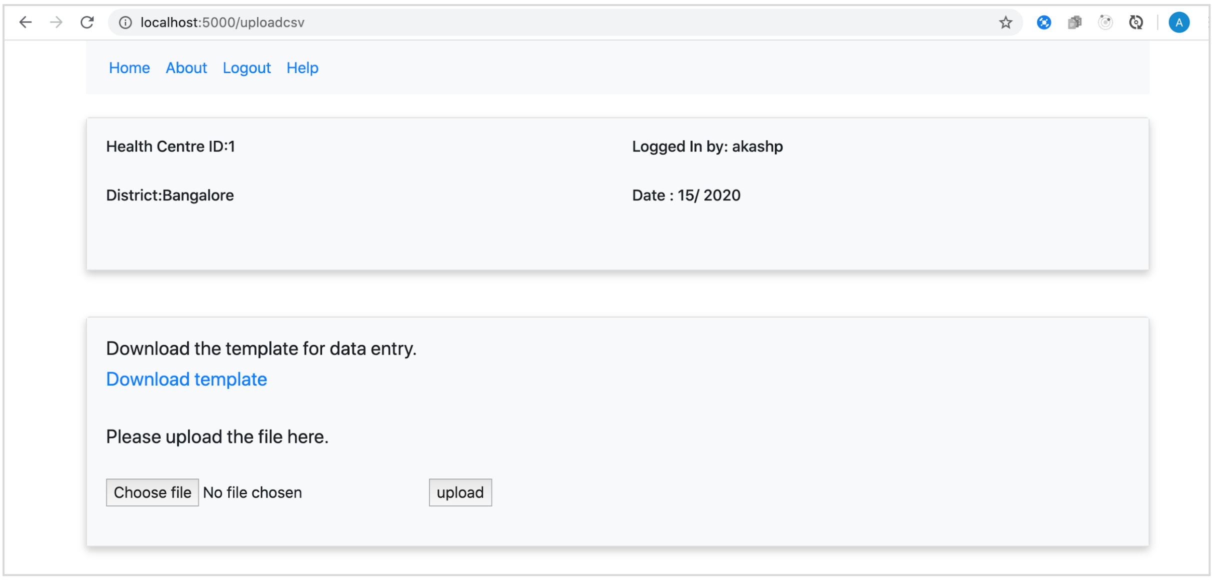


You are now logged in!

Select action

Figure 5.4: Dashboard with Upload CSV option

When the user clicks on the 'Upload CSV' button, he is taken to a page which has functionality to upload CSV files.



Health Centre ID:1

Logged In by: akashp

District:Bangalore

Date : 15/ 2020

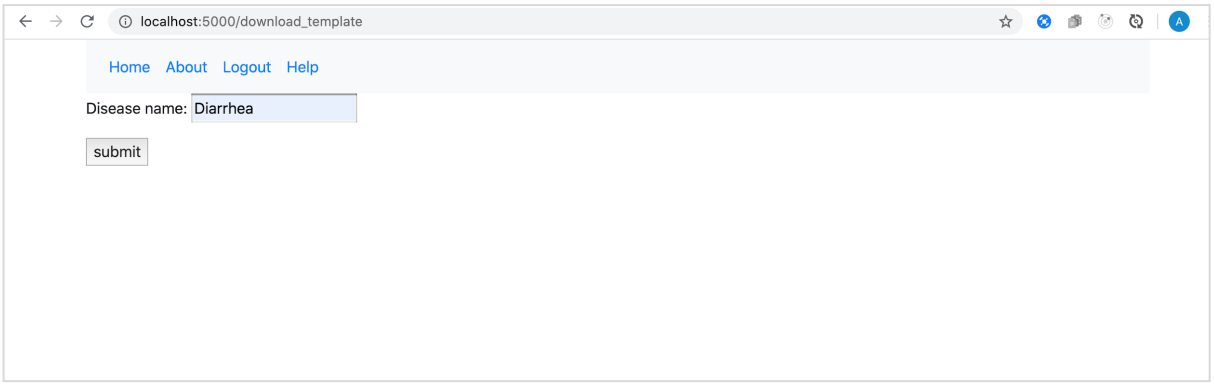
Download the template for data entry.

Download template

Please upload the file here.

Figure 5.5: Download and Upload CSV

The PHC user can click on "Download template" to download a template. The screen looks like this:

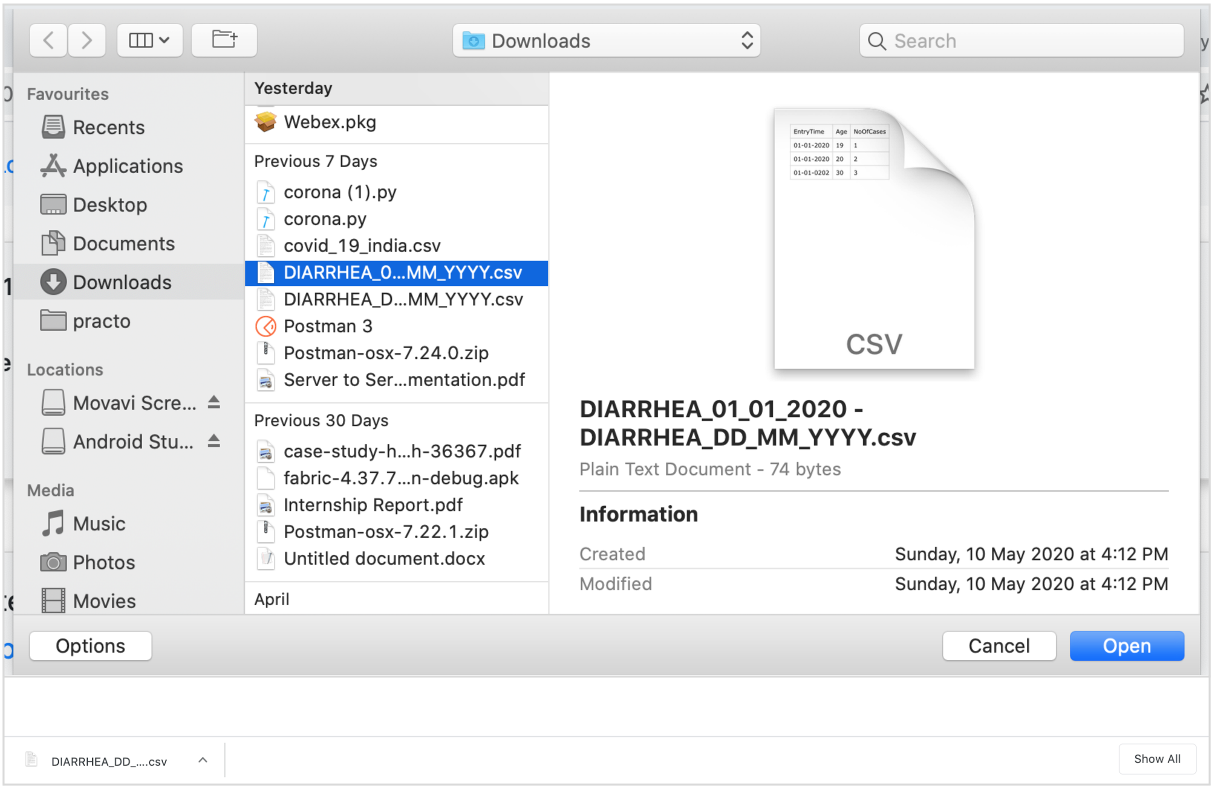


Disease name: Diarrhea

Figure 5.6: Download Template by disease

After clicking on submit, the template is downloaded as shown, only if the disease name is valid:

The user then selects a file from his filesystem, with the appropriate fields:



DIARRHEA_01_01_2020 - DIARRHEA_DD_MM_YYYY.csv

Plain Text Document - 74 bytes

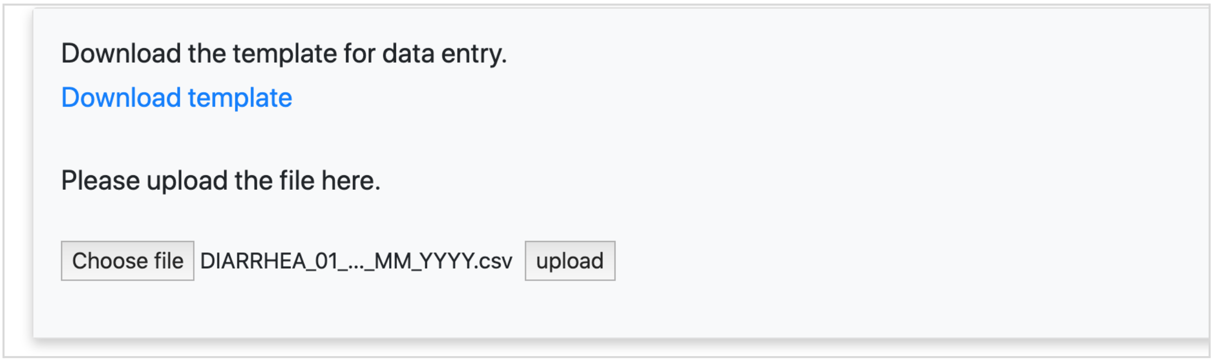
Information

Created Sunday, 10 May 2020 at 4:12 PM

Modified Sunday, 10 May 2020 at 4:12 PM

Figure 5.7: Upload CSV with data

And clicks on submit:

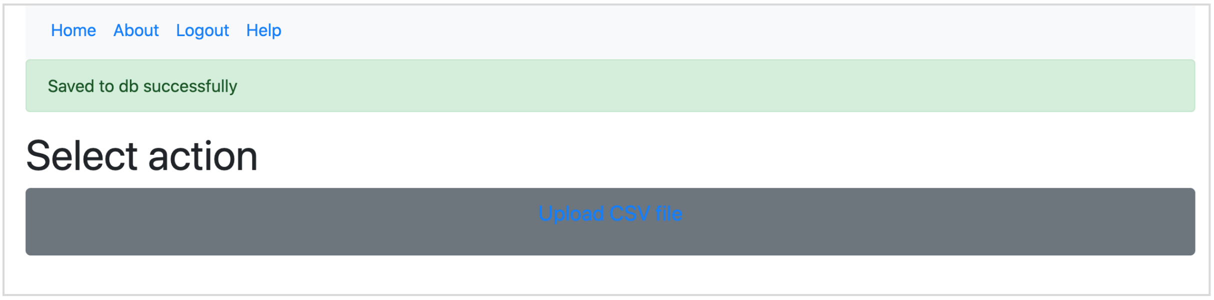

Download the template for data entry.

Download template

Please upload the file here.

Figure 5.8: Confirm upload CSV

The data is saved to the database successfully if the data validation succeeds, otherwise it fails.



Saved to db successfully

Select action

Figure 5.9: Data Upload successful notification

The role of the PHC user is done.

The admin will then look at reports and predictions made by the system. But first, he will have to login with his credentials. New admins can only be added to the system by manual database entry.

The admin login page is served by the */admin_login* route.

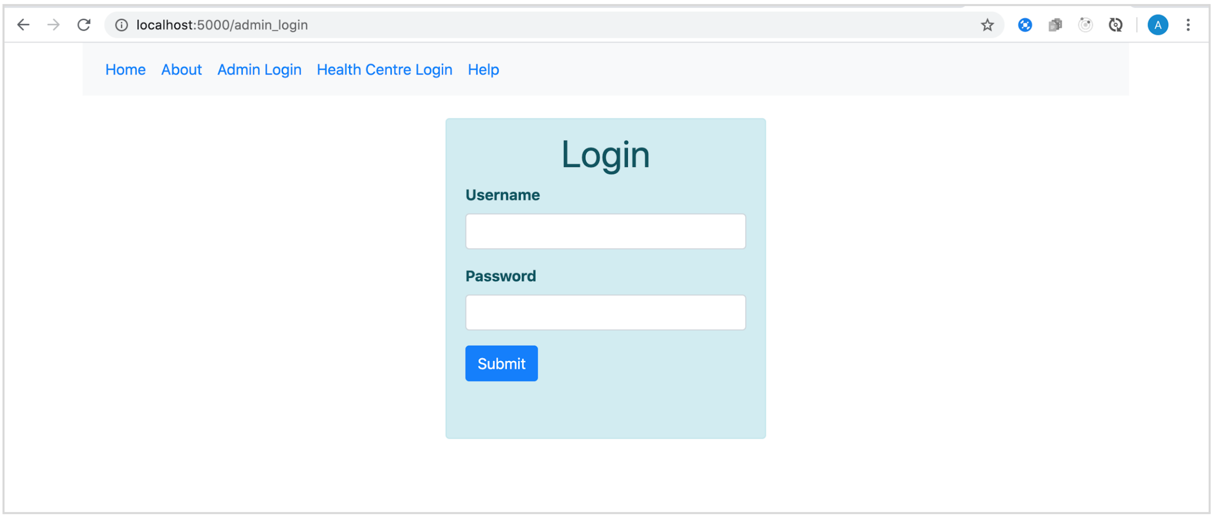
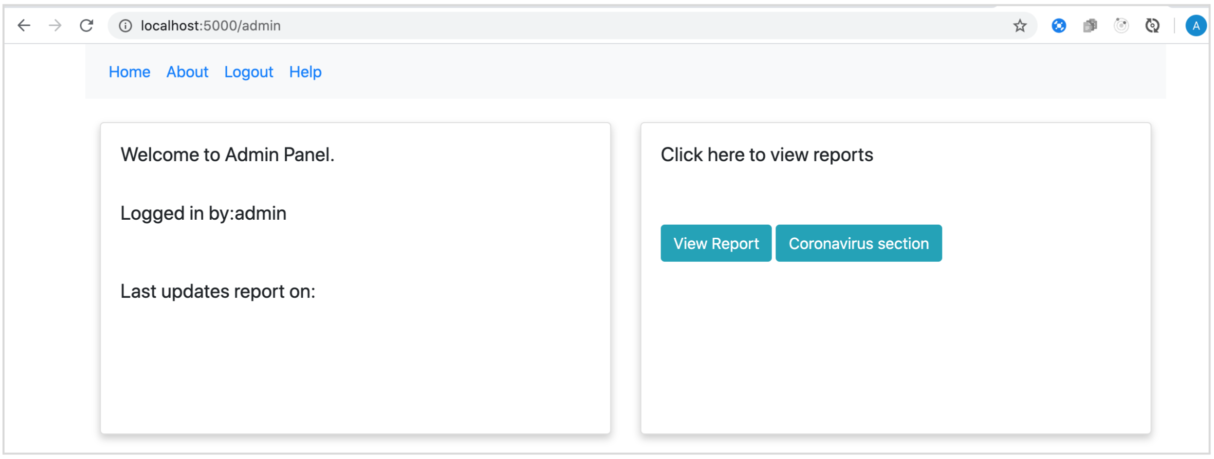


Figure 5.10: Admin Login Page

The */admin* looks like this:



Welcome to Admin Panel.

Logged in by:admin

Last updates report on:

Click here to view reports

Figure 5.11: Admin Dashboard with view report option

5.2.6 Report Generation - Standard Report

Reports are generated automatically once a day at midnight. Reports are classified into two different types - standard report and SARS-CoV-2 reports. Standard reports are designed for use in everyday scenarios with multiple disease data combined in one report. SARS-CoV-2 reports focus only on that disease and do not provide any outbreak analysis. A standard report consists of the following information:

- Actual and predicted data in the same report. A tab selector is provided for the selection of actual and predictions.
- Age distribution of all cases received in past week
- Disease distribution among the cases received in past week
- Disease distribution among cases for top 10 regions in past week
- Trends for top 10 diseases in top 10 regions for the past 4 weeks
- Trends for all diseases in all regions for past 8 weeks with filtering capability

Reports are full of visual representations of the data. Google Charts API is used to generate these graphs due to their low overhead on the server and client. Predictions contain mostly similar charts, but without age distribution. The trend charts (line charts) are for the next predicted week and past 3 or 7 weeks.

Report generation is performed in the generate_reports() function. This function is called after the predictions are made. The function combines data from the individual disease tables having names in the format <Disease> and <Disease>_Pred for actual data and predicted data respectively. These are temporarily stored in medical data and medicaldata_pred till the report generation is complete. Once report generation completes, these tables are cleared to save space.

Data is loaded into pandas DataFrames and processed to follow the data format supported by Google Charts API. Multiple levels of analysis are performed on the data and a short summary of key points is populated in the top of the page. Age, Disease, and region distribution charts are accompanied by their respective data in the form of sortable tables. These are arranged in an intuitive and semantic format through the use of borders, spacing and colors.

The 8 week trendline (or 7 weeks + 1 week prediction) chart has three types of filtering controls. These controls change the underlying data to modify the chart. The first is the region filter. Region filter allows for specifying one single region for which all diseases and upto 8 weeks are displayed. The second filter is the time range filter. This allows for a specific time range to be selected allowing for zooming in and out of time frames. The last filter is a disease filter. This filter allows the selection of one or more diseases that will be displayed in the chart. This ensures that low case count diseases are not obscured away.

The generate_reports() function converts data stored in an SQL server into a python format for processing and then into a javascript format for use in charts.

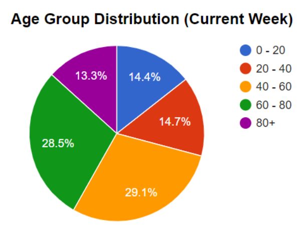
PandemicDetect Reports Predictions

Report Generated at: 15-05-2020 12:38

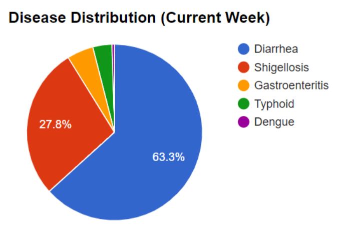
Useful Stats:

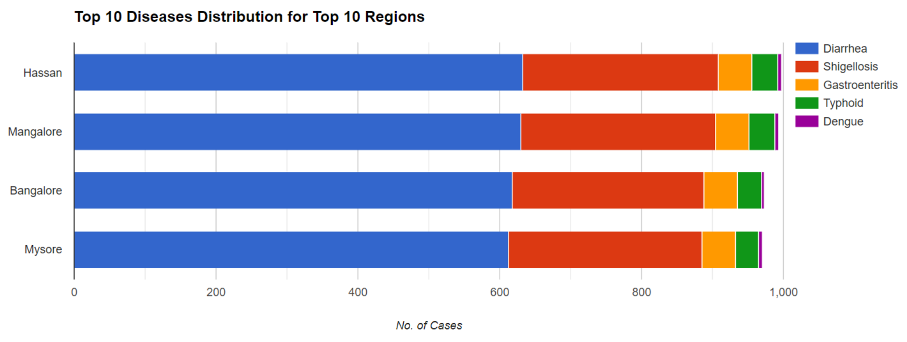
1. Number of new cases this week: 3933 cases
2. Change in cases from last week: +72.2% (+1649 cases)
3. Most infectious disease: Diarrhea
4. Most infected age group: 40 yrs - 60 yrs
5. Disease showing most improvement from last week: Typhoid
6. Disease showing least improvement from last week: Diarrhea
7. Region with highest cases this week: Hassan
8. Region with least cases this week: Mysore

Age Group (years)	No. of Cases	Percentage (%)
0 - 20	568	14.4
20 - 40	577	14.7
40 - 60	1,144	29.1
60 - 80	1,120	28.5
80+	524	13.3

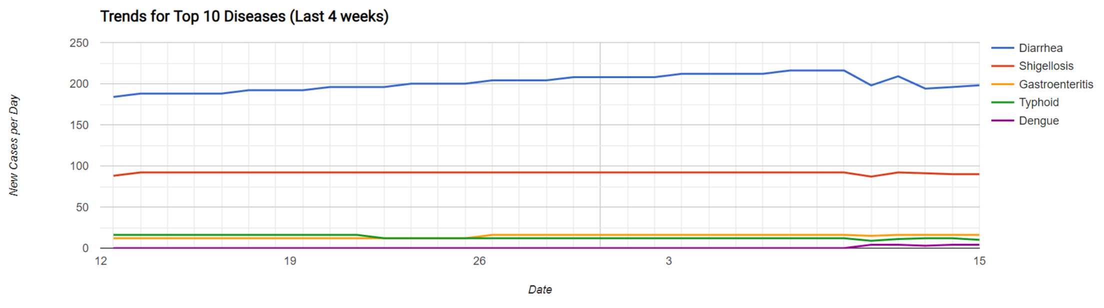


Disease	No. of Cases	Percentage (%)
Diarrhea	2,491	63.3
Shigellosis	1,094	27.8
Gastroenteritis	191	4.9
Typhoid	138	3.5
Dengue	19	0.5





Region	Diarrhea	Shigellosis	Gastroenteritis	Typhoid	Dengue
Hassan	632	276	48	36	5
Mangalore	630	274	48	36	5
Bangalore	617	271	47	34	4
Mysore	612	273	48	32	5



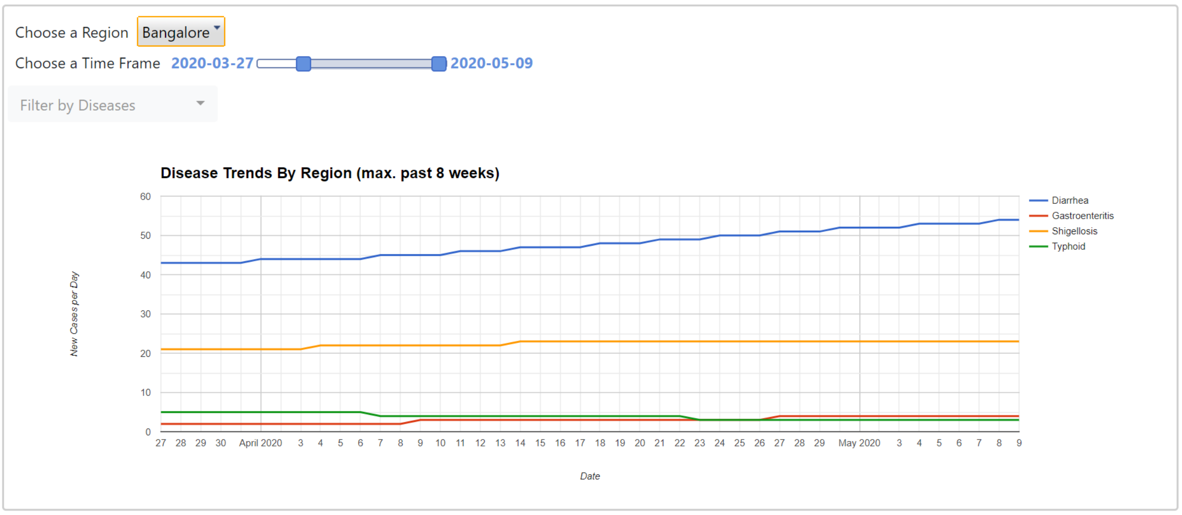
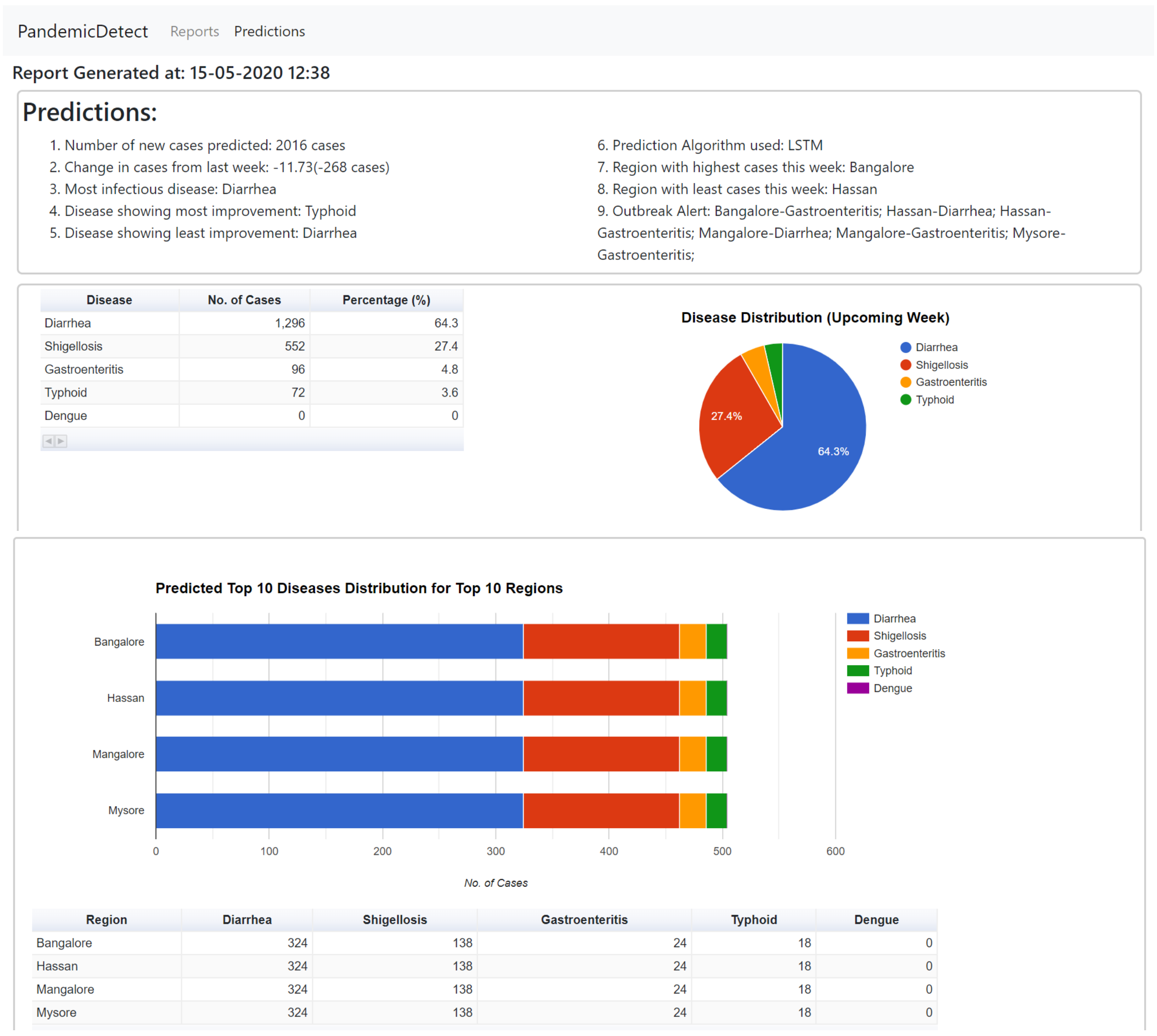


Figure 5.12: Reports Tab in standard report

Predictions Tab would contain all the predictions made by the algorithms. Here the algorithm used is LSTM.



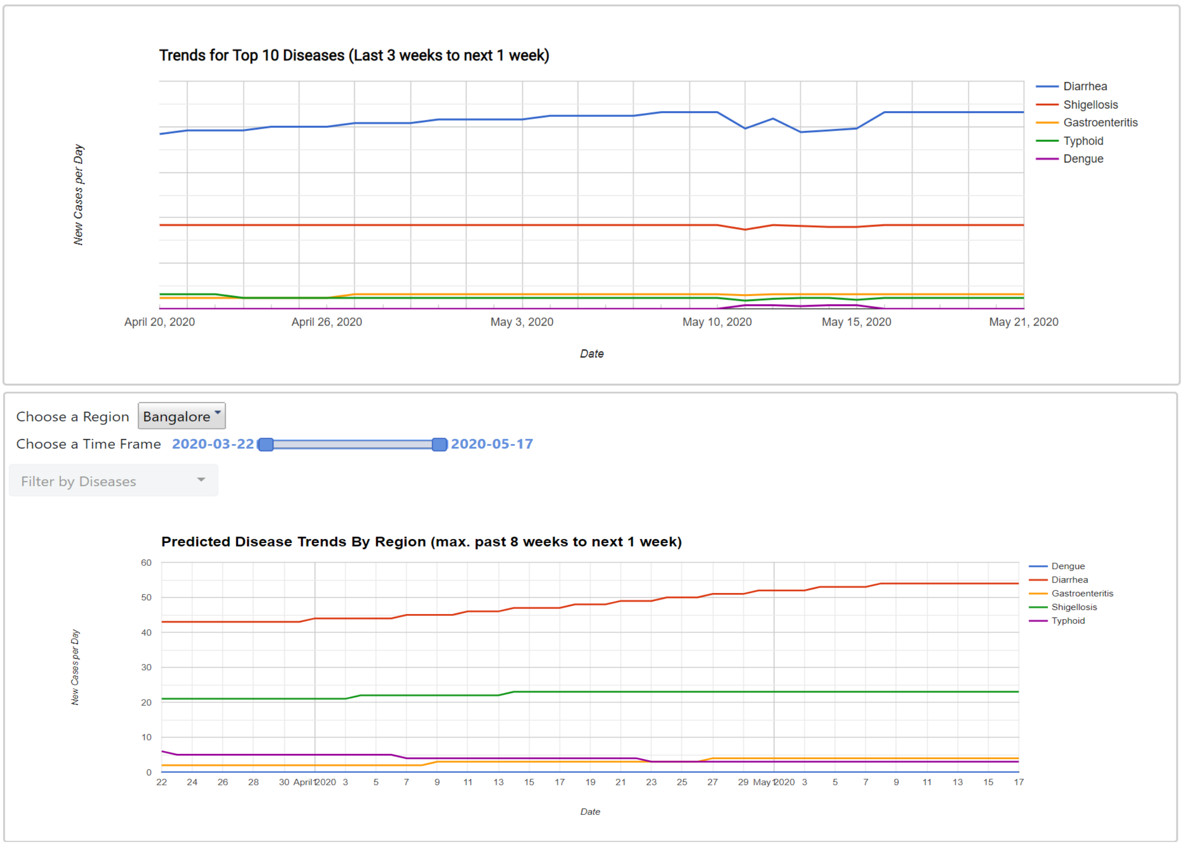
Report Generated at: 15-05-2020 12:38

Predictions:

1. Number of new cases predicted: 2016 cases
2. Change in cases from last week: -11.73(-268 cases)
3. Most infectious disease: Diarrhea
4. Disease showing most improvement: Typhoid
5. Disease showing least improvement: Diarrhea
6. Prediction Algorithm used: LSTM
7. Region with highest cases this week: Bangalore
8. Region with least cases this week: Hassan
9. Outbreak Alert: Bangalore-Gastroenteritis; Hassan-Diarrhea; Hassan-Gastroenteritis; Mangalore-Diarrhea; Mangalore-Gastroenteritis; Mysore-Gastroenteritis;

Disease	No. of Cases	Percentage (%)
Diarrhea	1,296	64.3
Shigellosis	552	27.4
Gastroenteritis	96	4.8
Typhoid	72	3.6
Dengue	0	0

Region	Diarrhea	Shigellosis	Gastroenteritis	Typhoid	Dengue
Bangalore	324	138	24	18	0
Hassan	324	138	24	18	0
Mangalore	324	138	24	18	0
Mysore	324	138	24	18	0

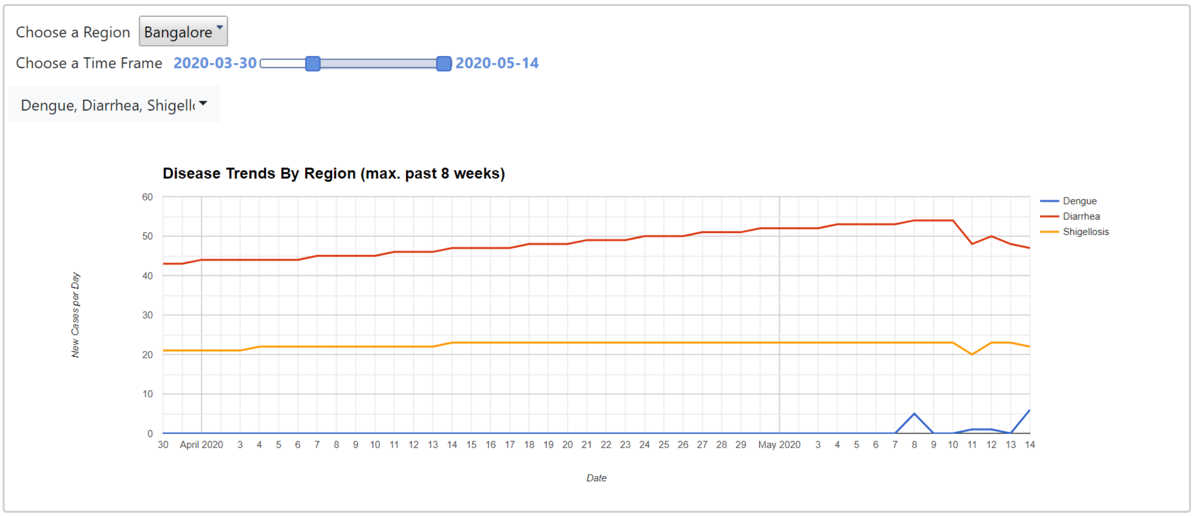


Choose a Region Bangalore

Choose a Time Frame 2020-03-22 2020-05-17

Filter by Diseases

Figure 5.13: Predictions Tab in standard report



Choose a Region Bangalore

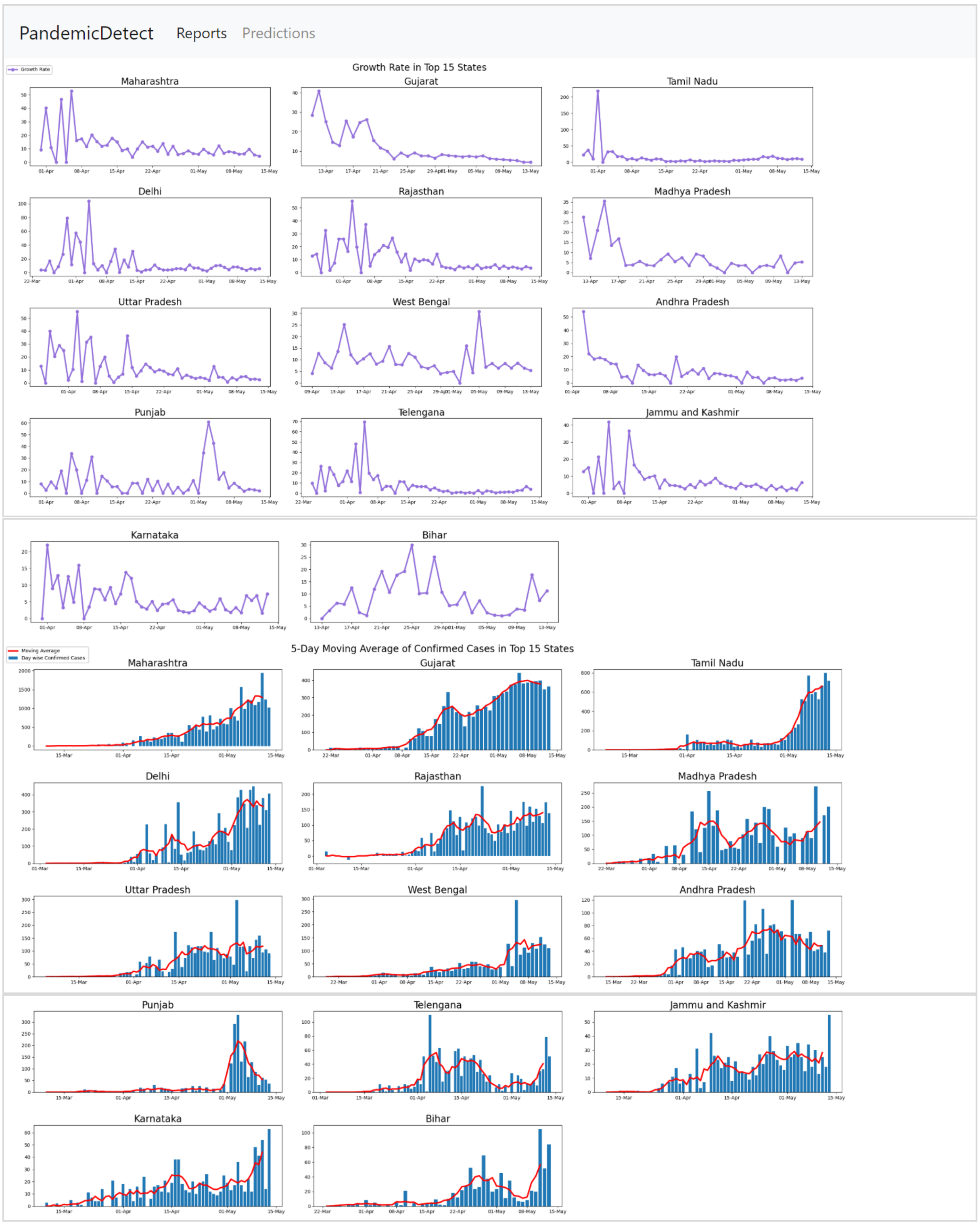
Choose a Time Frame 2020-03-30 2020-05-14

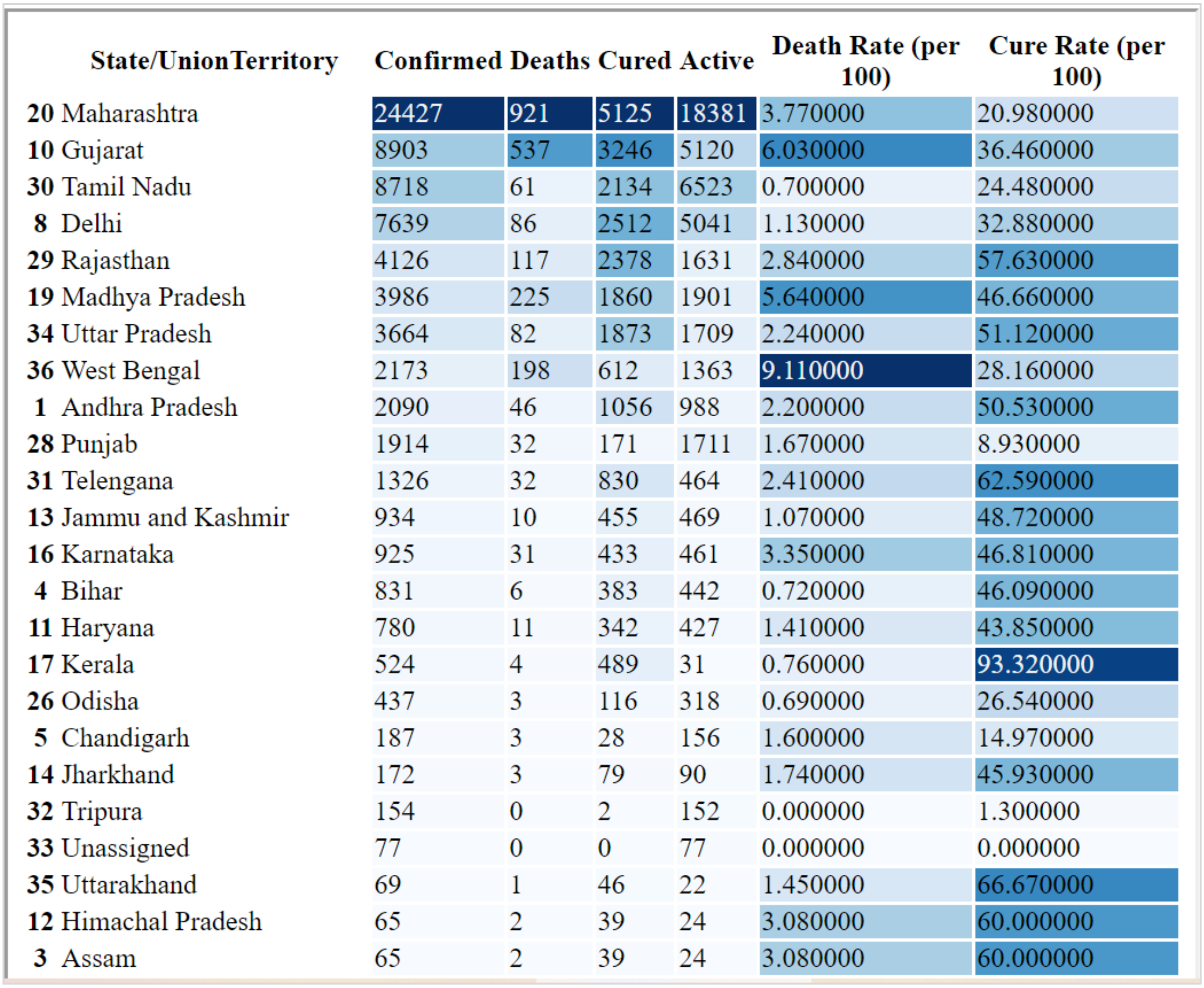
Dengue, Diarrhea, Shigelli

Figure 5.14: Configurable options for 8 week trendline chart

5.2.7 Report Generation - SARS-CoV-2 Reports

SARS-CoV-2 Reports focus mainly on the SARS-CoV-2/Novel Coronavirus disease and its trends. It consists of the trendlines for all the states since the start of the pandemic. The report consists of a reports tab and predictions tab similar to that of the standard report. It however does not provide age and region breakups and outbreak analysis. The predictions are generated using the ARIMA algorithm and plotted for the selected state. This chart is available in the predictions tab along with the state selection option.





	State/UnionTerritory	Confirmed	Deaths	Cured	Active	Death Rate (per 100)	Cure Rate (per 100)
20	Maharashtra	24427	921	5125	18381	3.770000	20.980000
10	Gujarat	8903	537	3246	5120	6.030000	36.460000
30	Tamil Nadu	8718	61	2134	6523	0.700000	24.480000
8	Delhi	7639	86	2512	5041	1.130000	32.880000
29	Rajasthan	4126	117	2378	1631	2.840000	57.630000
19	Madhya Pradesh	3986	225	1860	1901	5.640000	46.660000
34	Uttar Pradesh	3664	82	1873	1709	2.240000	51.120000
36	West Bengal	2173	198	612	1363	9.110000	28.160000
1	Andhra Pradesh	2090	46	1056	988	2.200000	50.530000
28	Punjab	1914	32	171	1711	1.670000	8.930000
31	Telengana	1326	32	830	464	2.410000	62.590000
13	Jammu and Kashmir	934	10	455	469	1.070000	48.720000
16	Karnataka	925	31	433	461	3.350000	46.810000
4	Bihar	831	6	383	442	0.720000	46.090000
11	Haryana	780	11	342	427	1.410000	43.850000
17	Kerala	524	4	489	31	0.760000	93.320000
26	Odisha	437	3	116	318	0.690000	26.540000
5	Chandigarh	187	3	28	156	1.600000	14.970000
14	Jharkhand	172	3	79	90	1.740000	45.930000
32	Tripura	154	0	2	152	0.000000	1.300000
33	Unassigned	77	0	0	77	0.000000	0.000000
35	Uttarakhand	69	1	46	22	1.450000	66.670000
12	Himachal Pradesh	65	2	39	24	3.080000	60.000000
3	Assam	65	2	39	24	3.080000	60.000000

Figure 5.15: Reports Tab in SARS-CoV-2 report

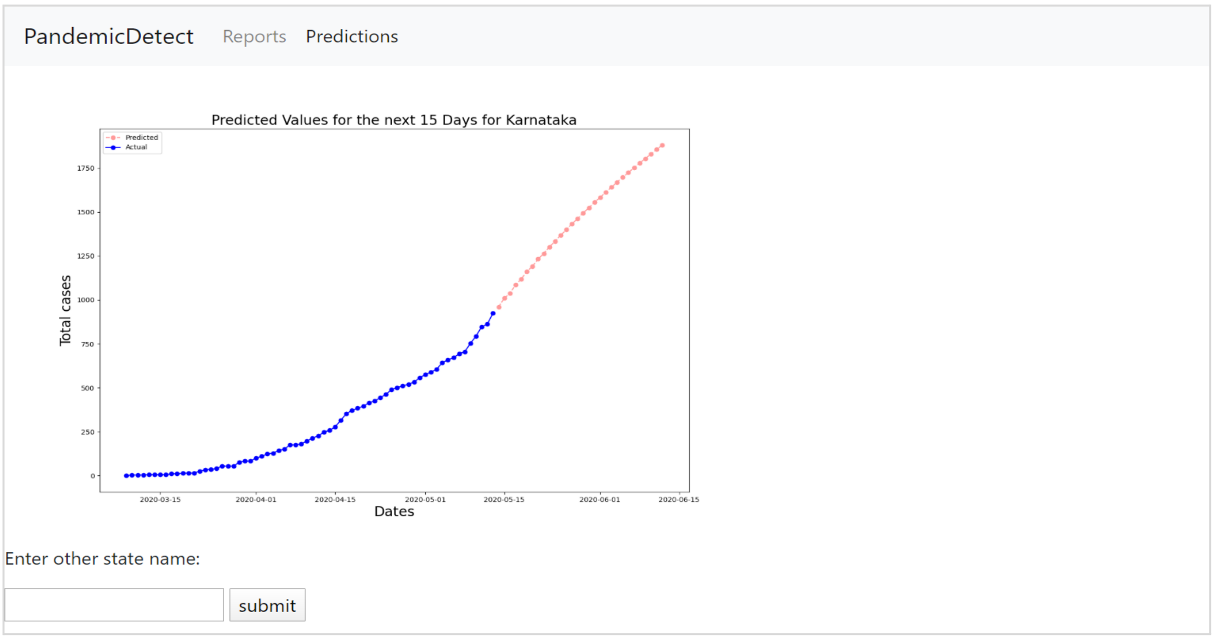


Figure 5.16: Predictions Tab in SARS-CoV-2 report

Testing

6.1 Testing with training dataset

- Training and testing dataset is a 75:25 split from the initial dataset. The 75% split is used for training and the rest is used for testing.
- The split is not made at random. This is because the dataset available uses data from 2016 onwards. To be able to determine the output for a set of inputs, the input order must be preserved.
- Data from 2016-2019 is used for training and 2019+ data is used for testing. This series is time shifted to 2017-2020 to simulate fresh data input.
- All the models scored a high accuracy score for the training set of about 85%.

LSTM+MLP training dataset accuracy: 85%

6.2 Testing with test dataset

- All the models scored a good accuracy score for the testing dataset of about 85%.
- Testing dataset is the 25% of remaining data from the dataset. This is used in batches of 7 consecutive values as inputs.
- A test input batch is provided to the LSTM model. It predicts the next day value. This value is fed again into the system with latest 6 of the original input batch. This chaining process takes place till 7 outputs are generated. This forms the data for the next week. The values of the next input batch are compared to the predicted output for accuracy.

LSTM+MLP testing dataset accuracy: 82%

Experimentation and Results

7.1 Results

- Machine learning models performed excellently with accuracy about 85%.
- Reports generated are informative and well laid out
- Predictions can be generated with any algorithm, and a default algorithm is used for daily predictions

7.2 Establishing Optimal Parameters - LSTM

7.2.1 Epochs

Epoch dependance on accuracy and R^2 score was measured by varying epochs from 1 to 500 with the following values for other parameters: Time lag = 7; LSTM Nodes = 50

It was noted that accuracy did not improve much after 200, and improved till 200. R^2 score, however, stayed constant after about 75 epochs.

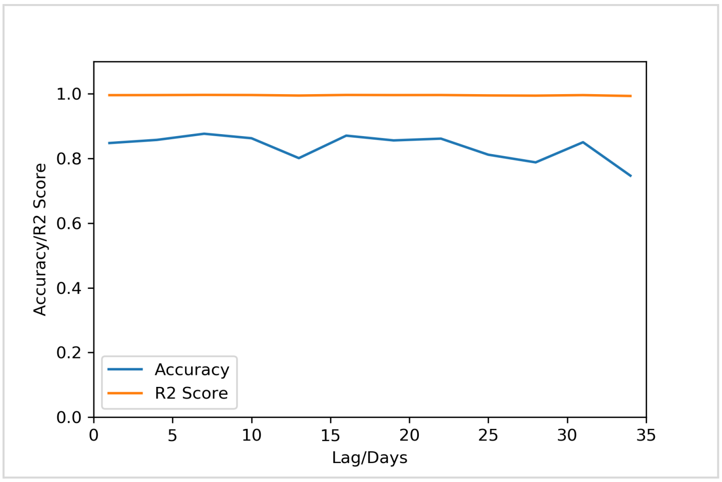


Figure 7.1: Plot of Accuracy and R^2 Score versus days of lag

7.2.2 Time lag

Time lag is the amount of historical data provided as input to predict the next day value as output. Accuracy decreased with increasing time lag. Time lag was varied from 1 to 35 days with the other parameters: Epochs = 200; LSTM Nodes = 50

It is evident that accuracy decreases with increasing time lag, at least for the diseases we studied. Accuracy peaks at about 7 days and again at about 16 days. But the difference between accuracies of 7 day lag and 16 day lag are not much different. R^2 Score stayed constant and irrespective of the lag.

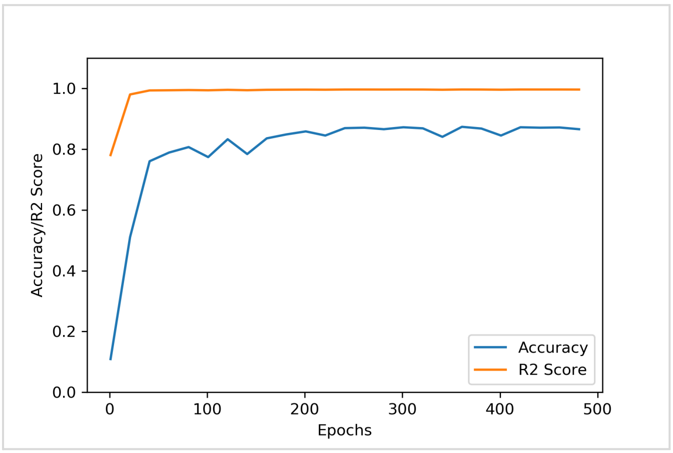


Figure 7.2: Plot of Accuracy and R^2 Score versus epochs of training

7.2.3 LSTM Layer Nodes

When configuring the model, one of the parameters required is the number of LSTM nodes in the network. The number of nodes represents the hidden complexity of the dataset. The other parameters were: Epochs = 200; Time lag = 7.

It was noted that accuracy and R2 score drastically improve till about 10 nodes. Then with a maxima at 20, there is fluctuation in the trend, but overall accuracy decreases with more nodes after 40. R^2 Score stayed constant after the initial rise.

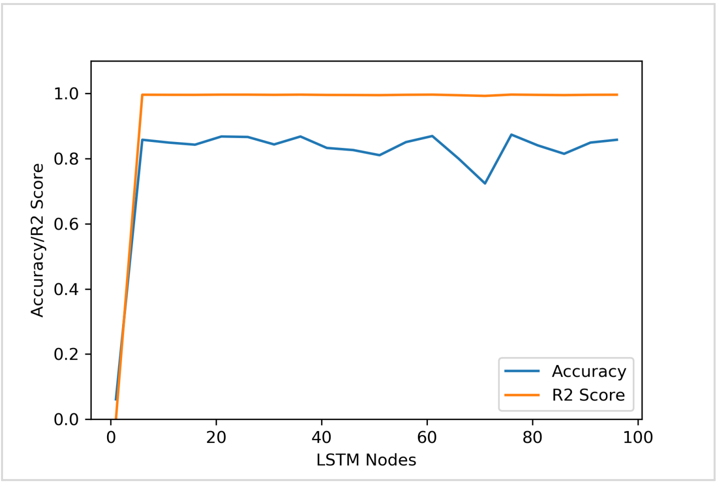


Figure 7.3: Plot of Accuracy and R^2 Score versus number of LSTM nodes

7.3 Establishing Optimal Parameters - MLP

7.3.1 Iterations/Epochs

Accuracy and R2 score were measured with the hidden layer sizes as (10,4). By varying max_iter from 1 to 100000, the following results were obtained.
Accuracy and R2 Score follow similar trends, they have a minor peak at about 5000 iterations and then settle between 20000 and 40000 iterations. After 40000 iterations, there seems to be inconsistent behavior in accuracy and R2 trends. A value of 30000 was chosen to be optimal.

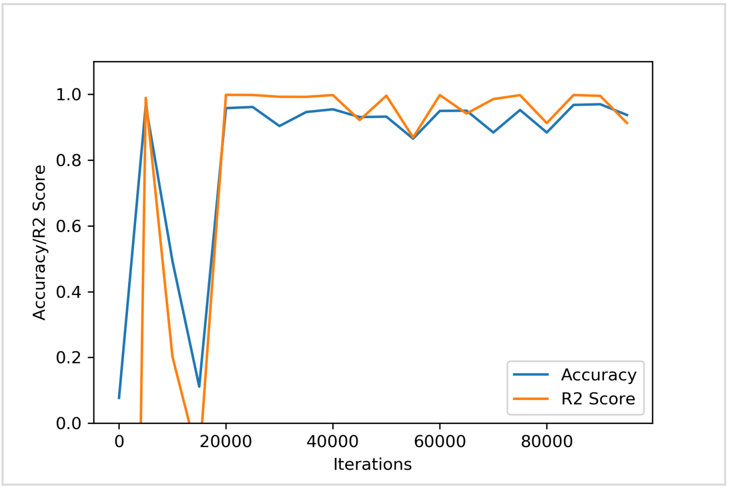


Figure 7.4: Plot of Accuracy and R^2 Score versus number of training iterations

Conclusion

The proposed system uses Machine Learning, specifically, time series machine learning combined with a classifier to predict the number of new cases in the upcoming week and the possibility of an outbreak from the data collected over the last week. This data is used to train the machine learning models automatically, once a week, to improve its prediction capabilities with new data. The objectives of this project were met by the following:

1. Provides an automated mechanism to allow for preparedness for outbreaks through the use of machine learning
2. Provides simple monitoring tools that allow for professionals and novice users to be able to use the analysis to understand the situation
3. Fast and efficient predictions due to storage of model parameters and weights

Future enhancements:

1. Use of geolocation data stored in the database to predict spread to neighbouring locations
2. Tracking of disease clusters for isolation and management
3. Improving on security and authorization aspects of the solution
4. Optimizing the database and server aspects for large scale deployment
5. Use of additional data features such as weather, population, cost of living in improving predictions
6. Predicting possible solutions and causes to outbreaks using fuzzy logic

The solution proposed in this project has a lot of potential in terms of automated disease surveillance and even in network surveillance. This project can be evolved to meet different requirements and processing styles. A full fledged deployment can be extremely beneficial to society.

References

[1] Chae, S.; Kwon, S.; Lee, D. Predicting Infectious Disease Using Deep Learning and Big Data. Int. J. Environ. Res. Public Health 2018, 15, 1596.

[2] Spreco A, Timpka T (2016). Algorithms for detecting and predicting influenza outbreaks: metanarrative review of prospective evaluations. BMJ Open 2016;6:e010683. doi: 10.1136/bmjopen-2015-010683

[3] Bracher, Johannes & Held, Leonhard. (2019). Multivariate endemic-epidemic models with higher-order lags and an application to outbreak detection.

[4] Zacher B, Ullrich A, Ghozzi S. Supervised Learning for Automated Infectious Disease - Outbreak Detection. Online J Public Health Inform. 2019;11(1):e261.

[5] Zacher, B., & Czogiel, I. (2019). Supervised learning improves disease outbreak detection. ArXiv, abs/1902.10061.

[6] Ray EL, Reich NG (2018) Prediction of infectious disease epidemics via weighted density ensembles. PLoS Comput Biol 14(2): e1005910.

[7] Suggala, Ravi & Anurag,. (2019). A Survey on Prediction and Detection of Epidemic Diseases Outbreaks.

[8] M. Chen, Y. Hao, K. Hwang, L. Wang and L. Wang, "Disease Prediction by Machine Learning Over Big Data From Healthcare Communities," in IEEE Access, vol. 5, pp. 8869-8879, 2017. doi: 10.1109/ACCESS.2017.2694446

[9] Glosser.sa, Wikipedia.org, Artificial Neural Network, CC-BY-SA 3.0

Appendix A: Code

app.py:

```
from flask import Flask
from datastore import db
from flask_restful import Api
from handlers import *
from models import *
from apscheduler.schedulers.background import BackgroundScheduler
from handlers.generate_prediction import *

def _init_app():
    app = Flask(__name__)
    app.config.from_pyfile("config.py")
    app.secret_key = "secret123"
    return app

app = _init_app()

def _init_db(app):
    db.init_app(app)
    with app.app_context():
        db.create_all()
    # Ensure FOREIGN KEY for sqlite3
    if 'sqlite' in app.config['SQLALCHEMY_DATABASE_URI']:
        def _fk_pragma_on_connect(dbapi_con, con_record):  # noqa
            dbapi_con.execute('pragma foreign_keys=ON')
        with app.app_context():
            from sqlalchemy import event
            event.listen(db.engine, 'connect', _fk_pragma_on_connect)

def _init_routes():
    api = Api(app)
    api.add_resource(Home, "/", methods=["GET"])
    api.add_resource(Login, "/login", methods=["GET", "POST"])
    api.add_resource(Register, "/register", methods=["GET", "POST"])
    api.add_resource(PHCDashboard, "/phcdashboard", methods=["GET"])
    api.add_resource(UploadCSV, "/uploadcsv", methods=["GET", "POST"])
    api.add_resource(Logout, "/logout", methods=["GET"])
    api.add_resource(Reports, "/reports", methods=["GET"])
    api.add_resource(Output, "/output/<path:path>", methods=["GET"])
    api.add_resource(GenRep, "/gen", methods=["GET"])  # Manual Report generation
trigger
    # Generate report for non-default algorithms. Default is ARIMA
    api.add_resource(Algo, "/algo", methods=["GET"])
    api.add_resource(DownloadTemplate, "/download_template", methods=["GET",
"POST"])
    api.add_resource(Admin, "/admin", methods=["GET", "POST"])
    api.add_resource(CoronavirusHandler, "/coronavirus", methods=["GET", "POST"])
    api.add_resource(About, "/about", methods=["GET"])
    api.add_resource(AdminLogin, "/admin_login", methods=["GET","POST"])

if __name__ == "__main__":
    scheduler = BackgroundScheduler(daemon=True)
    scheduler.add_job(generate_report, trigger='cron', day_of_week='mon-sun',
hour='0', minute='0')
    scheduler.add_job(train_all_models,trigger='cron',day_of_week='sun',hour='0'
,minute='0')
    scheduler.start()
    _init_routes()
    _init_db(app)
    app.run(debug=False,threaded=False)
```

reports.py:

```
from flask_restful import Resource
from flask import render_template, make_response, send_from_directory, redirect,
request, Markup, render_template_string
from models import PHCUser, medicaldata, reports, algorithms
from datastore import db
from datetime import datetime
from handlers.generate_report import *
from handlers.generate_prediction import *
from handlers.generate_coronavirus_report import *
from handlers.corona import *
from handlers.generate_coronavirus_report import *

class Reports(Resource):
    def get(self):
        algo=request.args.get('algo',default=None,type=str)
        print(algo)
        if(algo!=None):

list_reports=reports.query.filter_by(Algorithm=algo).order_by(reports.ReportTime.d
esc()).all()
        else:
            list_reports=reports.query.order_by(reports.ReportTime.desc()).all()
        print(list_reports)
        for i in list_reports:
            print(i,"\n",i.ReportLoc,i.ReportTime)
        list_rep=[]
        for i in list_reports:
            rep_name="Report as on "+i.ReportTime.strftime("%d %b %Y")
            tmp={ 'ReportTime':i.ReportTime.strftime("%d %b %Y %H:%M:%S"),
'ReportLoc':i.ReportLoc, 'ReportName':rep_name, 'AlgorithmName': i.Algorithm}
            list_rep.append(tmp)
        headers = {'Content-Type': 'text/html'}
        print(list_rep)
        return
make_response(render_template('reports.html',reports=list_rep),200,headers)

class Output(Resource):
    def get(self,path):
        headers = {'Content-Type': 'text/html'}
        if("reports/" in path):
            print(path)
            return send_from_directory('',path)
        elif("coronavirus_reports/" in path):
            return send_from_directory('',path)
        else:
            return make_response(render_template('error.html', errormsg="Access
Denied"),200,headers)

class GenRep(Resource):
    def get(self):
        algo=request.args.get('algo',default=None,type=str)
        print(algo)
        headers = {'Content-Type': 'text/html'}
        if(algo!=None):
            generate_report(algo,'custom')
        else:
            generate_report()
        return make_response(redirect('/reports'),303,headers)

class Algo(Resource):
    def get(self):
list_algorithms=algorithms.query.order_by(algorithms.DefaultAlgorithm.desc(),algori
thms.AlgorithmName.asc()).all()
        alist=[]
        for i in list_algorithms:
            if(i.AlgorithmName!="MLP"):
                alist.append(i)
        headers = {'Content-Type': 'text/html'}
        return
make_response(render_template('commanderconsole.html',algolist=alist),200,headers
)

def generate_report(algorithm=None,launch_method='auto'):
    if(algorithm=="Coronavirus"):
        get_results()
        gen_coronavirus_report(launch_method,algorithm)
        return
    if (algorithm==None):
        al=algorithms.query.filter_by(DefaultAlgorithm=True).first()
        al=al.AlgorithmName
    else:
        al=algorithm
    al1=algorithms.query.all()
    flag=0
    for i in al1:
        if(i.AlgorithmName==al):
            flag=1
    if(flag==0):
        raise Exception("Algorithm does not exist")

    gen_pred(al,launch_method)
    gen_report(launch_method,al)

    get_results()
    gen_coronavirus_report(launch_method,algorithm)
```

generate_predictions.py:

```
from handlers.ml_algorithm_aditya import *
from handlers.ml_algorithms import *
from handlers.train_models import *

def gen_pred(algorithm=None,launch_method='auto'):
   print("Prediction storage code goes here")
   if(algorithm=="LSTM"):
      LSTM_Aditya(launch_method)
      MLP_Aditya(launch_method)
   elif(algorithm=="Coronavirus"):
      pass #Handled elsewhere
   else:
      pass
   print("Predictions generated and stored")

def train_all_models():
   train_LSTM()
```

ml_algorithm_aditya.py:

```
import numpy as np
from numpy import array
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime,timedelta
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import load_model
import joblib
from models import Diseases,algorithms,Location
from datastore import db

def split_sequence(sequence, n_steps):
        X, y = list(), list()
        for i in range(len(sequence)):
                end_ix = i + n_steps
                if end_ix > len(sequence)-1:
                        break
                seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
```

```
            X.append(seq_x)
            y.append(seq_y)
        return array(X), array(y)

def LSTM_Aditya(launch_method):
    diseases1=Diseases.query.all()
    algorithms1=algorithms.query.filter_by(AlgorithmName="LSTM").first()
    algorithms1=algorithms1.AlgorithmLoc
    locations1 = Location.query.order_by(Location.id).all()
    diseases=[]
    queries=[]
    currtime=datetime.strptime(datetime.now().strftime("%Y %m %d 00 00 00"),"%Y
%m %d %H %M %S")
    lastweek=currtime-timedelta(currtime.weekday())-timedelta(weeks=1,days=1)
    nextweek = currtime+timedelta(weeks=1)
    for i in diseases1:
        diseases.append(i.Disease)
        model = load_model(algorithms1+"_"+i.Disease+".tf")
        scaler = joblib.load(algorithms1+"_"+i.Disease+"_Scaler.gz")
        for j in locations1:
            df=pd.read_sql("""SELECT EntryTime, CentreCode, sum(NoOfCases) as
Count FROM """+i.Disease+""" WHERE
EntryTime>='"""+lastweek.strftime("%Y-%m-%d %H:%M:%S")+"""' AND
EntryTime<='"""+currtime.strftime("%Y-%m-%d %H:%M:%S")+"""' AND
CentreCode="""+str(j.id)+""" GROUP BY EntryTime ORDER BY EntryTime
DESC;""",db.engine)
            print(i.Disease, j.id, df)
            print("""SELECT EntryTime, CentreCode, sum(NoOfCases) as Count FROM
"""+i.Disease+""" WHERE EntryTime>='"""+lastweek.strftime("%Y-%m-%d
%H:%M:%S")+"""' AND EntryTime<='"""+currtime.strftime("%Y-%m-%d
%H:%M:%S")+"""' AND CentreCode="""+str(j.id)+""" GROUP BY EntryTime
ORDER BY EntryTime DESC;""")
            if(len(df.values)==0):
                print("None")
                continue
            else:
                df['EntryTime']=pd.to_datetime(df['EntryTime'])
                idx = pd.date_range(lastweek+timedelta(days=1), currtime)
                df.index=pd.DatetimeIndex(df['EntryTime'])
                df = df.reindex(idx, fill_value=0)
                df=df.drop(labels='EntryTime',axis=1)
                df['CentreCode']=j.id
            print(i.Disease, j.id, df)
            preds=[]
            for k in range(0,7):
                model_inputs=np.ravel(df.iloc[0:7,1:2].values)
                model_inputs=np.append(model_inputs,preds)
                model_inputs=model_inputs[-7:]
                model_inputs=model_inputs.reshape(len(model_inputs),1)
                model_inputs=scaler.transform(model_inputs)
                model_inputs=model_inputs.reshape((1,7,1))
                pr = model.predict(model_inputs)
                pr = scaler.inverse_transform(pr)
                #pr=-pr
                preds.append(np.round(pr,0))
            preds=np.ravel(preds)
            print(preds)
            idx = pd.date_range(currtime,nextweek)
            print(idx)
            for k in range(7):
                db.engine.execute("""INSERT INTO """+i.Disease+"""_Pred
(EntryTime, CentreCode, NoOfCases)
VALUES('"""+str(idx[k])+"""',"""+str(j.id)+""","""+str(int(preds[k]))+""");""")
    print("In LSTM")

def MLP_Aditya(launch_method):
    diseases1=Diseases.query.all()
    algorithms1=algorithms.query.filter_by(AlgorithmName="MLP").first()
    algorithms1=algorithms1.AlgorithmLoc
    locations1 = Location.query.order_by(Location.id).all()
    diseases=[]
    queries=[]
    if(launch_method=='auto'):
        currtime=datetime.strptime(datetime.now().strftime("%Y %m %d 00 00
00"),"%Y %m %d %H %M %S")
    else:
        currtime=datetime.strptime(datetime.now().strftime("%Y %m %d %H %M
%S"),"%Y %m %d %H %M %S")
    lastweek=currtime-timedelta(currtime.weekday())-timedelta(weeks=1,days=1)
    for i in diseases1:
        diseases.append(i.Disease)
        model1 = joblib.load(algorithms1+"_"+i.Disease+".joblib")
        scaler = joblib.load(algorithms1+"_"+i.Disease+"_Scaler.gz")
        for j in locations1:
```

```
        df=pd.read_sql("""SELECT EntryTime, CentreCode, sum(NoOfCases) as
Count FROM """+i.Disease+""" WHERE
EntryTime>='"""+lastweek.strftime("%Y-%m-%d %H:%M:%S")+"""' AND
EntryTime<='"""+currtime.strftime("%Y-%m-%d %H:%M:%S")+"""' AND
CentreCode="""+str(j.id)+""" GROUP BY EntryTime ORDER BY EntryTime
DESC;""",db.engine)
        df2=pd.read_sql("""SELECT EntryTime,CentreCode, sum(NoOfCases) as
Count FROM """+i.Disease+"""_Pred WHERE
EntryTime>='"""+currtime.strftime("%Y-%m-%d %H:%M:%S")+"""' AND
CentreCode="""+str(j.id)+""" GROUP BY EntryTime ORDER BY EntryTime
DESC;""",db.engine)
        df=pd.concat([df,df2],ignore_index=True)
        if(len(df.values)==0):
            print("None")
            continue
        else:
            df['EntryTime']=pd.to_datetime(df['EntryTime'])
            #idx = pd.date_range(lastweek, currtime)
            #df.index=pd.DatetimeIndex(df['EntryTime'])
            #df = df.reindex(idx, fill_value=0)
            #df=df.drop(labels='EntryTime',axis=1)
            df['CentreCode']=j.id
            print(df)
            features=df.iloc[:,2:3].values
            features=np.array(features)
            features=features.reshape(-1,1)
            features=scaler.transform(features)
            preds=model1.predict(features)
            df['OutbreakVal']=np.round(preds,0)
            print(df)
            for k in range(7):
                db.engine.execute("""INSERT INTO outbreak_analysis (EntryTime,
CentreCode, Disease, OutbreakFlag)
VALUES('"""+df.iloc[k,0:1].values[0].strftime('%Y-%m-%d')+"""',"""+str(j.id)+""",
'"""+i.Disease+"""',"""+str(int(preds[k]))+""");""")
```

corona.py:

```
import pandas as pd
import numpy as np
import datetime
import requests
import warnings
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import squarify
import plotly.offline as py
from statsmodels.tsa.arima_model import ARIMA
from IPython.display import Image

def get_table_in_html(input_filename):
    india_covid_19 = pd.read_csv(input_filename)
    india_covid_19['Date'] = pd.to_datetime(india_covid_19['Date'],dayfirst = True)
    state_cases =
india_covid_19.groupby('State/UnionTerritory')['Confirmed','Deaths','Cured'].max().r
eset_index()

state_cases['Active']=state_cases['Confirmed']-(state_cases['Deaths']+state_cases['C
ured'])
    state_cases["Death Rate (per 100)"] =
np.round(100*state_cases["Deaths"]/state_cases["Confirmed"],2)
    state_cases["Cure Rate (per 100)"] =
np.round(100*state_cases["Cured"]/state_cases["Confirmed"],2)
    state_table = (state_cases.sort_values('Confirmed', ascending= False).fillna(0)
    .style.background_gradient(cmap='Blues',subset=["Confirmed"])
    .background_gradient(cmap='Blues',subset=["Deaths"])
    .background_gradient(cmap='Blues',subset=["Cured"])
    .background_gradient(cmap='Blues',subset=["Active"])
    .background_gradient(cmap='Blues',subset=["Death Rate (per 100)"])
    .background_gradient(cmap='Blues',subset=["Cure Rate (per 100)"]))

    state_table_html = state_table.render()
    filename = 'coronavirus_reports/' + datetime.date.today().strftime("%Y-%m-%d")
+ '_00-00-00_' + 'coronavirus-table.html'
    with open(filename,'w') as file:
        file.write(state_table_html)
    filename = 'static/' + str(datetime.date.today()) + '_00-00-00_' +
'coronavirus-table.html'
    with open(filename,'w') as file:
        file.write(state_table_html)
    return

def calc_movingaverage(values ,N):
    cumsum, moving_aves = [0], [0,0]
    for i, x in enumerate(values, 1):
        cumsum.append(cumsum[i-1] + x)
        if i>=N:
            moving_ave = (cumsum[i] - cumsum[i-N])/N
            moving_aves.append(moving_ave)
    return moving_aves

def calc_growthRate(values):
    k = []
    for i in range(1,len(values)):
        summ = 0
        for j in range(i):
            summ = summ + values[j]
        rate = (values[i]/summ)*100
        k.append(float(rate))
    return k

def get_moving_average_growth_rate_and_prediction(input_filename,
state_name='Karnataka'):
    matplotlib.use('Agg')
    india_covid_19 = pd.read_csv(input_filename)#1st problem
    india_covid_19['Date'] = pd.to_datetime(india_covid_19['Date'],dayfirst = True)
    all_state = list(india_covid_19['State/UnionTerritory'].unique())
    all_state.remove('Unassigned')
    latest = india_covid_19[india_covid_19['Date'] > '30-01-20']
    state_cases =
latest.groupby('State/UnionTerritory')['Confirmed','Deaths','Cured'].max().reset_inde
x()
    latest['Active'] = latest['Confirmed'] - (latest['Deaths']- latest['Cured'])
    state_cases = state_cases.sort_values('Confirmed', ascending= False).fillna(0)
    states =list(state_cases['State/UnionTerritory'][0:15])
    states_confirmed = {}
    states_deaths = {}
    states_recovered = {}
    states_active = {}
    states_dates = {}
    for state in states:
        df = latest[latest['State/UnionTerritory'] == state].reset_index()
        k = []
        l = []
        m = []
        n = []
        for i in range(1,len(df)):
            k.append(df['Confirmed'][i]-df['Confirmed'][i-1])
            l.append(df['Deaths'][i]-df['Deaths'][i-1])
            m.append(df['Cured'][i]-df['Cured'][i-1])
            n.append(df['Active'][i]-df['Active'][i-1])
        states_confirmed[state] = k
        states_deaths[state] = l
        states_recovered[state] = m
        states_active[state] = n
        date = list(df['Date'])
        states_dates[state] = date[1:]
    fig = plt.figure(figsize= (25,17))
    plt.suptitle('5-Day Moving Average of Confirmed Cases in Top 15 States',fontsize =
20,y=1.0)
    k=0
    for i in range(1,15):
        ax = fig.add_subplot(5,3,i)
        ax.xaxis.set_major_formatter(mdates.DateFormatter('%d-%b'))
        ax.bar(states_dates[states[k]],states_confirmed[states[k]],label = 'Day wise
Confirmed Cases ')
        moving_aves = calc_movingaverage(states_confirmed[states[k]],5)
        ax.plot(states_dates[states[k]][:-2],moving_aves,color='red',label = 'Moving
Average',linewidth =3)
        plt.title(states[k],fontsize = 20)
        handles, labels = ax.get_legend_handles_labels()
        fig.legend(handles, labels, loc='upper left')
        k=k+1
    plt.tight_layout(pad=3.0)
    moving_average_fig = fig
    filename = 'coronavirus_reports/' + datetime.date.today().strftime("%Y-%m-%d")
+ '_00-00-00_' + 'coronavirus-MovingAverageGraph.png'
    moving_average_fig.savefig(filename)
    filename = 'static/' + str(datetime.date.today()) + '_00-00-00_' +
'coronavirus-MovingAverageGraph.png'
    moving_average_fig.savefig(filename)
    fig = plt.figure(figsize= (25,17))
    plt.suptitle('Growth Rate in Top 15 States',fontsize = 20,y=1.0)
    k=0
    for i in range(1,15):
        ax = fig.add_subplot(5,3,i)
        ax.xaxis.set_major_formatter(mdates.DateFormatter('%d-%b'))
        #ax.bar(states_dates[states[k]],states_confirmed[states[k]],label = 'Day wise
Confirmed Cases ')
        growth_rate = calc_growthRate(states_confirmed[states[k]])
        ax.plot_date(states_dates[states[k]][21:],growth_rate[20:],color =
'#9370db',label = 'Growth Rate',linewidth =3,linestyle='-')
        plt.title(states[k],fontsize = 20)
        handles, labels = ax.get_legend_handles_labels()
        fig.legend(handles, labels, loc='upper left')
        k=k+1
    plt.tight_layout(pad=3.0)
    growth_rate_graph_fig = fig
    filename = 'coronavirus_reports/' + datetime.date.today().strftime("%Y-%m-%d")
+ '_00-00-00_' + 'coronavirus-GrowthRateGraph.png'
    growth_rate_graph_fig.savefig(filename)
    filename = 'static/' + str(datetime.date.today()) + '_00-00-00_' +
'coronavirus-GrowthRateGraph.png'
    growth_rate_graph_fig.savefig(filename)
    k = india_covid_19[india_covid_19['State/UnionTerritory']==
state_name].iloc[:,[1,8]]
    data=k
    arima = ARIMA(data['Confirmed'], order=(5, 1, 0))
    arima = arima.fit(trend='c', full_output=True, disp=True)
    forecast = arima.forecast(steps= 30)
    pred = list(forecast[0])
    start_date = data['Date'].max()
    prediction_dates = []
    for i in range(30):
        date = start_date + datetime.timedelta(days=1)
        prediction_dates.append(date)
        start_date = date
    fig = plt.figure(figsize= (15,10))
    plt.xlabel("Dates",fontsize = 20)
    plt.ylabel('Total cases',fontsize = 20)
    plt.title("Predicted Values for the next 15 Days for " +state_name , fontsize = 20)
    plt.plot_date(y= pred,x= prediction_dates,linestyle ='dashed',color =
'#ff9999',label = 'Predicted');
    plt.plot_date(y=data['Confirmed'],x=data['Date'],linestyle = '-',color = 'blue',label
= 'Actual');
    plt.legend()
    prediction_fig = fig
    filename = 'coronavirus_reports/' + str(datetime.date.today()) + '_00-00-00_' +
'coronavirus_Prediction_' + state_name +'.png'
```

```
    prediction_fig.savefig(filename)
    filename = 'static/' + str(datetime.date.today()) + '_00-00-00_' + 
'coronavirus_Prediction_' + state_name+ '.png'
    prediction_fig.savefig(filename)

def get_results(filename='covid_19_india.csv'):
    get_moving_average_growth_rate_and_prediction(filename)
    get_table_in_html(filename)
```

train_models.py:

```
import numpy as np
from numpy import array
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from joblib import dump, load
from sklearn.preprocessing import MinMaxScaler
from datetime import datetime,timedelta
from tensorflow.keras.models import load_model
from models import Diseases, algorithms, Location
from keras.models import Sequential
from keras.layers import LSTM, Dense
import os
from datastore import db
from keras.metrics import Accuracy

def split_sequence(sequence, n_steps):
        X, y = list(), list()
        for i in range(len(sequence)):
                # find the end of this pattern
                end_ix = i + n_steps
                # check if we are beyond the sequence
                if end_ix > len(sequence)-1:
                        break
                # gather input and output parts of the pattern
                seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
                X.append(seq_x)
                y.append(seq_y)
        return array(X), array(y)

def train_LSTM():
    list_diseases=Diseases.query.order_by(Diseases.Disease).all()
```

```
  for i in list_diseases:
     df=pd.read_sql("""SELECT EntryTime, sum(NoOfCases) as Count FROM
"""+i.Disease+""" GROUP BY EntryTime ORDER BY EntryTime;""",db.engine)
     raw_seq = np.ravel(df.iloc[:,1:2].values)
     raw_seq=raw_seq.reshape(len(raw_seq),1)
     scaler = MinMaxScaler(feature_range=(0, 1))
     scaler = scaler.fit(raw_seq)
     raw_seq=scaler.transform(raw_seq)
     n_steps = 7
     X, y = split_sequence(raw_seq, n_steps)
     n_features = 1
     X = X.reshape((X.shape[0], X.shape[1], n_features))
     model = Sequential()
     model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
     model.add(Dense(1))
     model.compile(optimizer='adam', loss='mse',metrics=[Accuracy()])
     model.fit(X, y, epochs=200, verbose=0)
     fd=os.getcwd()+"/MLModels/LSTM_"+i.Disease+".tf"
     print(fd)
     model.save(fd)
     fd=os.getcwd()+"/MLModels/LSTM_"+i.Disease+"_Scaler.gz"
     print(fd)
     dump(scaler,fd)
     print("Training complete")
```

upload_csv.html:

```
{% extends 'layout.html' %}
{% block body %}
<!-- <a href='/download_template'> Download template </a>
<form method="POST" enctype=multipart/form-data> <input type=file
name=file> <input type='submit' value='upload'>
</form> -->
<br>
<div class="card" style="height:10rem" ;>
 <div class="card-body" style="background-color:#f8f9fa" ;>
   <div class="row">
     <div class="col-sm-6">
      <h6> Health Centre ID:{{location_id}}</h6>
       <br>
       <h6>District:{{district}}</h6>
       <br><br>
     </div>
```

```
      <div class="col-sm-6">
        <h6>Logged In by: {{username}}</h6>
        <br>
        <h6>Date :
          <script>document.write(new Date().getDate())</script>/
          <script>document.write(new Date().getFullYear())</script>
        </h6>
      </div>
    </div>
  </div>
</div><br><br>
<div class="card" style="height:15rem" ;>
  <div class="card-body" style="background-color:#f8f9fa" ;>
    <!-- <div class="container" > -->
    <h5>Download the template for data entry.</h5>
    <h5 class="card-title"><a href='/download_template'> Download template
</a></h5>
    <br>
    <h5>Please upload the file here.</h5>
    <br>
    <form method="POST" enctype=multipart/form-data> <input type=file
name=file> <input type='submit' value='upload'>
    </form>

  </div>
</div>
{% endblock %}
```

admin_login.py:

```
from flask_restful import Resource
from models import AdminModel
from passlib.hash import sha256_crypt
from flask import make_response, render_template, session, flash, request, redirect,
url_for
from datastore import db

class AdminLogin(Resource):
    def get(self):
        if not AdminModel.query.all():
            admin_model =
AdminModel(name='admin',username='admin',password=sha256_crypt.encrypt('pas
sword'))
```

```
            db.session.add(admin_model)
            db.session.commit()

        headers = {'Content-Type': 'text/html'}
        return make_response(render_template('login.html'),200,headers)
    def post(self):
        username = request.form['username']
        password_candidate = request.form['password']  #This is the password the user
has entered.
        admin_user = AdminModel.query.filter_by(username=username).first()

        if admin_user:
            password = admin_user.password
            if sha256_crypt.verify(password_candidate,password):
                session['logged_in'] = True
                session['username'] = username
                session['permission'] = 'admin'
                flash('You are now logged in!', 'success')
                headers = {'Content-Type': 'text/html'}
                return redirect(url_for('admin'))
            else:
                error = "Invalid credentials"
                headers = {'Content-Type': 'text/html'}
                return make_response(render_template("login.html",
error=error),200,headers)
        else:
            error = "Admin not found"
            headers = {'Content-type': 'text/html'}
            return make_response(render_template('login.html', error=error), 200,
headers)
```

admin.py:

```
from flask_restful import Resource
from flask import render_template, make_response, session, request, redirect, url_for
from models import reports
from handlers.logged_in_required import login_required
from models import Location
from flask import render_template, make_response, request, session, flash, redirect,
url_for
from models import PHCUser,reports,Location
from datastore import db
```

```
class Admin(Resource):
   @login_required
   def get(self):
      r=reports.query.order_by(reports.ReportTime.desc()).all()
      print(r)
      if(len(r)>0):
         lr=r[0].ReportTime
      else:
         lr="None"
      headers = {'Content-Type': 'text/html'}

      logged_in_user = session['username']
      return make_response(render_template('admin.html',username=logged_in_user),
200, headers)
   @login_required
   def post(self):
      district = request.form.get('district')
      country = request.form.get('country')
      state = request.form.get('state')
      cordinates = request.form.get('cord')
      location = Location(district=district, country=country, state=state,
location_coord=cordinates)
      db.session.add(location)
      db.session.commit()
      flash("New location created. Use id ="+location.id+" to register", "success")
      return redirect(url_for('admin'))



      return make_response(render_template('admin.html',lastreport=lr), 200,
headers)
   def post(self):
      if('permissions' in session):
         if(session['permissions']=='Admin'):
            l = Location(district=request.form['district'], state=request.form['state'],
country=request.form['country'], location_coord=request.form['cord'])
            db.session.add(l)
            db.session.commit()
            flash('Centre Added!', 'success')
         else:
            return make_response(render_template('admin.html',lastreport="None"),
200, {'Content-Type': 'text/html'})
```

```
    else:
        return make_response(render_template('admin.html',lastreport="None"), 200,
{'Content-Type': 'text/html'})
```

medicaldata.py:

```
from datastore import db
from sqlalchemy import ForeignKey

class medicaldata(db.Model):
  __tablename__='medicaldata'
  id=db.Column(db.Integer, primary_key=True, autoincrement=True)
  EntryTime=db.Column(db.DateTime)
  CentreCode=db.Column(db.Integer,nullable=False)
  Disease=db.Column(db.String(300),nullable=False)
  Age=db.Column(db.Integer)
  NoOfCases=db.Column(db.Integer, default=1, nullable=False)

class medicaldata_pred(db.Model):
  __tablename__='medicaldata_pred'
  id=db.Column(db.Integer,primary_key=True,autoincrement=True)
  EntryTime = db.Column(db.DateTime)
  CentreCode = db.Column(db.Integer,nullable=False)
  Disease = db.Column(db.String(300),nullable=False)
  Age = db.Column(db.Integer)
  NoOfCases = db.Column(db.Integer, default=1, nullable=False)

class reports(db.Model):
  __tablename__='reports'
  id=db.Column(db.Integer,primary_key=True,autoincrement=True)
  ReportTime=db.Column(db.DateTime,nullable=False)
  ReportLoc=db.Column(db.String(1000),nullable=False)
  Algorithm=db.Column(db.String(300))

class algorithms(db.Model):
  __tablename__='algorithms'
  id=db.Column(db.Integer,primary_key=True,autoincrement=True)
  AlgorithmName=db.Column(db.String(100),nullable=False)
  AlgorithmLoc = db.Column(db.String(1000),nullable=False)
  DefaultAlgorithm = db.Column(db.Boolean,nullable=False, default=False)
  Accuracy = db.Column(db.String(20))

class outbreak_analysis(db.Model):
```

```
__tablename__='outbreak_analysis'
id=db.Column(db.Integer,primary_key=True,autoincrement=True)
EntryTime = db.Column(db.DateTime)
CentreCode = db.Column(db.Integer)
Disease = db.Column(db.String(200))
OutbreakFlag = db.Column(db.Integer)
```